HERIOT WATT UNIVERSITY

BIOLOGICALLY INSPIRED COMPUTATION

COURSEWORK 1B

Fashion MNIST Dataset Classification with One Hidden Layer

Author
Mohit VAISHNAV

Supervisor
Dr. Marta VALLEJO

Submission Last Date 22^{th}, Oct.

1 Introduction

In the earlier coursework, focus was on binary classification without any use of hidden layer with just one neuron. Moving ahead, now the problem statement is to predict the classification for MNIST Fashion dataset. Its a ten class problem namely, *T-Shirt/Top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot*. Few of them can be visualized as in Fig. 1. Data is composed of 60000 images for training purpose while 10000 images are for the test purpose. Each image is of the size 28×28, contained in a vector format giving it a size of (60000,784) for training images. Correspondingly each image has a label from 0 to 9 each having a meaning representing one class.

Figure 1: Sample Images

2 Procedure

2.1 Loading Dataset

Two kinds of files are provided for the evaluation purpose. One is in *CSV* format while the other is in *MNIST* designated format created by *Zolando* [1]. This dataset is used a benchmark for by many people working in the same domain. To read this file in *python*, *MNIST* library needs to be installed in Jupyter Notebook after which with one command, these files can be loaded from the folder.

Softmax calculates probability distribution for each of the 10 classes in the range of 0 to 1. Next, loss function is to be defined which gives the estimate of working of the neural network. Cross entropy loss is calculated for each prediction w.r.t prediction as:

$$C(S, L) = -\sum_i L_i \log(S_i)$$

L for this purpose has to be one hot encoded, meaning having index 1 at the corresponding index and rest everywhere 0. Cross entropy loss gives the intuition of the class where the label belongs to which is decided by the magnitude of softmax vector. If this increases, there is less likely probability that the index is correctly predicted and vice versa. Average cross entropy loss takes into account average training loss. In any neural network, major part involves the estimation of parameters to ensure this function to be minimum using gradient descent. This is written as:

$$L = -\frac{1}{N}\sum_j C(S(Wx_j + b), L_j)$$

The above shows the task of creating a linear multilabel classification. In case of addition of more artificial neurons to logistic classifier instead passing the result directly to the softmax function [[6]], this is again passed through another artificial neurons which is in general a non linear function. So the new function becomes:

$$\hat{y} = softmax(W_2(nonlin(W_1 x + b_1)) + b_2)$$

This is different from the previous coursework in terms of number of bias and weights. To optimize such functions, there is need of learning the weights, which involves a concept called *back propagation*.

Intermediate steps as mentioned above is also known as hidden layer. All the parameter of this layer can be controlled to make the model work like an optimal ones. Also it is possible to add as many layers as possible and tune all the parameters.

2.2 Implementation

In this implementation, there is just one hidden layer which could be of arbitrary size. So two weights matrix have to be initialized, one which get multiplied with the input while coming towards hidden layer and the other

going to the softmax layer from the output of hidden layer. Now the input data formats are checked for their validity.

One very important step involved in pre-processing is to convert the data to the format of one hot encoding which is essential in the case of multi-class classification of any other similar kind of application. Hence labels are converted from the dimension $N \times 10$ the column is given value at that index where that particular class is found to be present. Randomizing the data has a positive affect on the output of the model so *random.permutation* is used for that purpose.

Now the task is to fit the data in the neural network. Two most important steps are forward and backward propagation. In forward propagation, prediction is carried out while in backward propagation weights are updated based on the prediction error which is averaged over all the training samples.

While carrying out the forward propagation non linearity is introduced by *tanh or ReLU*. Output from this step is passed as an input to the hidden layer after considering weights into factor and the same is done in *softmax* to make the required predictions. These weights and bias are saved for the later stage in back propagation. Using these predictions, made after the softmax and the input of one hot encoded true labels, cost is calculated.

Now that the system could make the prediction and calculate the loss, next comes the task of learning and optimizing which is done using Gradient Descent. Also derivative of the non linear functions are defined with the name *dtanh & dReLU*. It works by calculating the partial derivative of weights w.r.t costs, intuitively showing the direction of fastest increase in loss function. After this, weights are updated taking each step based on the learning rate towards minimum direction.

2.3 Factors to be taken care of:

- Initialization of the weights meaningfully.
- Learning rate has to be defined wisely.
- Can also use mini-batch gradient descent, i.e. randomly selecting subset of training data and update weights.

2.4 Run the code

Jupyter notebook can run all the cells if desired libraries are installed in the system.

3 Observation and Result

First the data set has to be observed because equal distribution of data set in very much essential for obtaining good accuracy of results. Using *seaborn* library of *python* it can be visualized as in Fig. 2. Now we can move on to more intricate parameters to work on. Even for the confusion martix, this library can be used to plot any kind of results which can be seen as in Fig. 3.

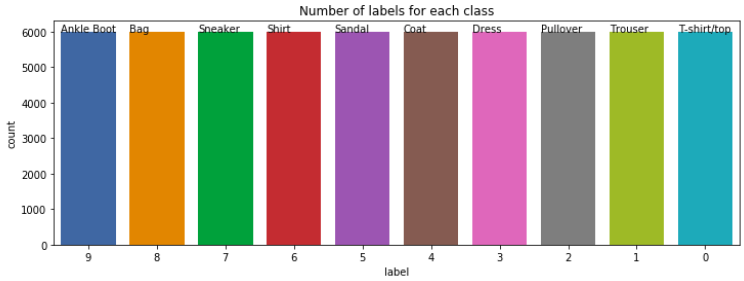


Figure 2: Distribution of data set

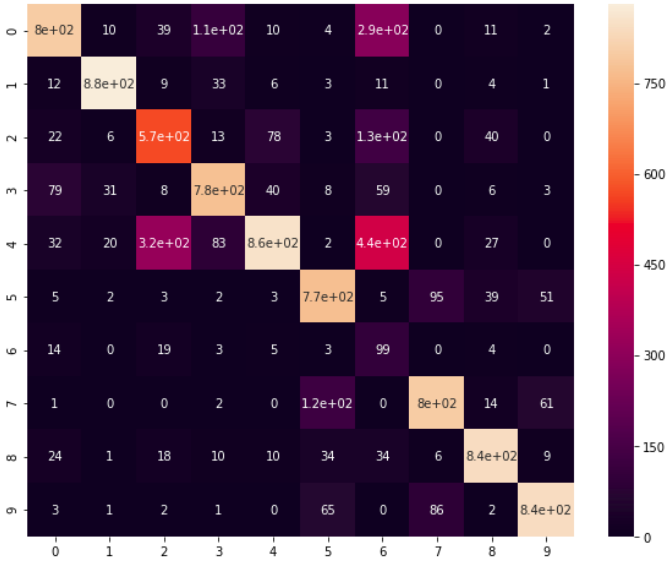


Figure 3: Confusion Matrix using Seaborn Library

Hyper-parameters plays an important role when deciding any model. They have to be tuned in a proper way. In this case, normalized random

values are selected using the function, *randn* which allocates the value in between -1 and 1 while are bias are kept as 0 initially. Now looking towards the degeneracy issue.

One of the issue while running any machine learning problem is the occurrence of *nan*. Here it has been dealt by clipping the value to a minimum value, ie. .000001 which avoids the situation of $\log(0)$ kind of situation.

In some scenario over-fitting of data may happen. It can be minimized using the *L-2* regularization which adds penalty on larger weights value. Dropout is another method proposed to solve the same over fitting problem. Every neuron is usually connected to all other neurons from the next layer (also known as *fully connected*). But in case of dropout, few of these linkages are broken intentionally, hence not passing output from the previous activation function to next level.

At the end, accuracy is the motive. To calculate the accuracy required libraries are imported from *Sklearn.Metrics* like classification_report, confusion_matrix and f1_score. Here we get the number in all understandable formats like confusion matrix, precision, recall, f1 score and average of all those.

So to understand in a better way, different combinations tried are by varying learning rate, different non linear functions and number of neurons in hidden layer. Different combinations of different sets tried are: Number of neurons as 64, 128 and 150; Non linear functions: Sigmoid, ReLU and Tanh; Learning rate: 0.01, 0.5, 0.85 & 1.

Discussing about the non linear activation functions, it can be seen that *Tanh* does not seems to work properly and it deviates from the results very widely (Fig 6,7,10). Even by changing the method of normalizing the values if input images, there occurs issues. Like if the normalization of data is done in $l1$ or $l2$ way using *preprocessing.normalize(X, norm='l1/l2')* from Sklearn library else just diving the data by 255. Amongst all best results were found to be from $l2$.

Next observation for this kind of data set is in terms of learning rate. Unlike other places where the learning rate is kept to be usually very low, here is does not work similarly. Learning rate around 1 seems to work very good. This can be observed from all the Figures included in the report.

Now the parameter considered for altering is number of neurons of hidden layer. Many different values were tried in this section like 64,128 & 150. From the confusion matrix seen in Fig. 11 it is very much clear that just increasing the number of neurons does not make any significant improvement in the performance of the model.

At the end there is no harm in mentioning a point that all this efforts

can be saved using the Keras/Tensorflow structure defined very clearly on internet. They makes the life easy to use and implement any complex neural network we wanted including all the functionality in just a line of code.

4 Conclusion

This coursework helped in gaining a meaningful insight in working of the neural network. Starting from the very basic algorithm where there was requirement to develop a neural network with a hidden layer played an important role in understanding the basics. After trying various combination of experiments it became visible the role of different hyper-parameters involved in a model. To make any model work, it requires a combination of consideration from reading the data, normalizing it, usage of different activation function, learning rate to quote some. Here we see that this data set requires high learning rate to reach to the optimum solution in adequate number of epochs with the data normalized in a proper way and activation function as ReLU/Sigmoid.

5 Self Analysis

Table. 1 shows the markings.

Question	S_Mark	Final
Did you code the entire algorithm?	x	
Did you vectorise your code?	x	
Is your code runnable?	x	
Did you collect data to measure convergence?	x	
Did you create a report explaining what you did?	x	

Table 1: Coursework 1b

```
[[476  23  31 276  16  41 134   0  93  14]
 [ 44 666  16 258  67   0  52   0   3   0]
 [128  17 191  35 204  12 141   1 247   4]
 [ 65 239  72 315 107   9  60   0   5  12]
 [ 57   2 124  23 277   4 289   0  86  10]
 [ 17   9 123   5  38 154  52 233  68  29]
 [178  33 412  79 266  31 214   0  62   5]
 [  1   0   4   0   3 369   7 629  30 101]
 [ 30   9  26   8  21 166  46  27 270  66]
 [  4   2   1   1   1 214   5 110 136 759]]
              precision    recall  f1-score   support

           0       0.48      0.43      0.45      1104
           1       0.67      0.60      0.63      1106
           2       0.19      0.19      0.19       980
           3       0.32      0.36      0.33       884
           4       0.28      0.32      0.30       872
           5       0.15      0.21      0.18       728
           6       0.21      0.17      0.19      1280
           7       0.63      0.55      0.59      1144
           8       0.27      0.40      0.32       669
           9       0.76      0.62      0.68      1233

avg / total       0.42      0.40      0.40     10000
```

(a) (0.01,Sigmoid,l2)

```
[[716   7  32  75  11   5 165   1  15   2]
 [ 13 859   7  47  21  21  16   1   0   2]
 [ 20  18 596  21 118   7 198   0  33   0]
 [ 75  77  11 738  57   2  60   0  14   0]
 [ 15  13 204  38 627   4 193   0  25   0]
 [  7   1  19   5   2 675  12  73  32  57]
 [117  16 128  60 153  12 291   1  25   3]
 [  2   0   4   1   0 133   3 771  23  67]
 [ 33   1  32  10  12  70  39   8 803  26]
 [  0   1   1   0   0 104   1 105  13 868]]
              precision    recall  f1-score   support

           0       0.72      0.70      0.71      1029
           1       0.87      0.87      0.87       987
           2       0.58      0.59      0.58      1011
           3       0.74      0.71      0.73      1034
           4       0.63      0.56      0.59      1119
           5       0.65      0.76      0.70       883
           6       0.30      0.36      0.33       806
           7       0.80      0.77      0.79      1004
           8       0.82      0.78      0.80      1034
           9       0.85      0.79      0.82      1093

avg / total       0.70      0.69      0.70     10000
```

(b) (0.5,ReLU,l2)

```
[[1003 1017  985 1002 1028  988 1009  987  983  998]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]]
              precision    recall  f1-score   support

           0       1.00      0.10      0.18     10000
           1       0.00      0.00      0.00         0
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0
           5       0.00      0.00      0.00         0
           6       0.00      0.00      0.00         0
           7       0.00      0.00      0.00         0
           8       0.00      0.00      0.00         0
           9       0.00      0.00      0.00         0

avg / total       1.00      0.10      0.18     10000
```

(c) (0.5,Tanh,l2)

```
[[ 999 1008 1020  998 1022  963 1023  996  947 1024]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]]
              precision    recall  f1-score   support

           0       1.00      0.10      0.18     10000
           1       0.00      0.00      0.00         0
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0
           5       0.00      0.00      0.00         0
           6       0.00      0.00      0.00         0
           7       0.00      0.00      0.00         0
           8       0.00      0.00      0.00         0
           9       0.00      0.00      0.00         0

avg / total       1.00      0.10      0.18     10000
```

(d) (0.85,Tanh,l2)

Figure 4: Confusion Matrix for 64 neurons in hidden layer in order (learning rate, non linear function, normalization of data)

```
[[761  10  16  68  15   0 183   0  13   1]
 [ 17 897   5  49   3   1   8   2   2   0]
 [ 26   6 625  14 170   5 152   1  35   7]
 [ 72  68   8 724  48   6  55   0  40   0]
 [  9  11 182  32 580   4 217   0  23   0]
 [  4   0   1   5   3 749   5  97  31  31]
 [ 88  34 158  61 134  12 340   0  23   6]
 [  2   0   1   0   0 118   1 835  10  93]
 [ 23   1  23  14  22  39  43   9 819  12]
 [  0   0   3   4   2 101   3  66  12 791]]
              precision    recall  f1-score   support

           0       0.76      0.71      0.74      1067
           1       0.87      0.91      0.89       984
           2       0.61      0.60      0.61      1041
           3       0.75      0.71      0.73      1021
           4       0.59      0.55      0.57      1058
           5       0.72      0.81      0.76       926
           6       0.34      0.40      0.37       856
           7       0.83      0.79      0.81      1060
           8       0.81      0.81      0.81      1005
           9       0.84      0.81      0.82       982

avg / total       0.72      0.71      0.71     10000
```

(a) (0.85,ReLU,l2)

```
[[   0    0    0    0    0    1    0    0    1    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    4    0    0    0    0]
 [ 997 1022  995 1047 1008  682  986  805 1008  994]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   1    2    1    0    0  276    2  158    0   10]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.00      0.00      0.00         0
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0
           5       0.00      1.00      0.01         4
           6       1.00      0.10      0.19      9544
           7       0.00      0.00      0.00         0
           8       0.00      0.00      0.00         0
           9       0.01      0.02      0.01       450

avg / total       0.95      0.10      0.18     10000
```

(b) (0.85,Tanh,l1)

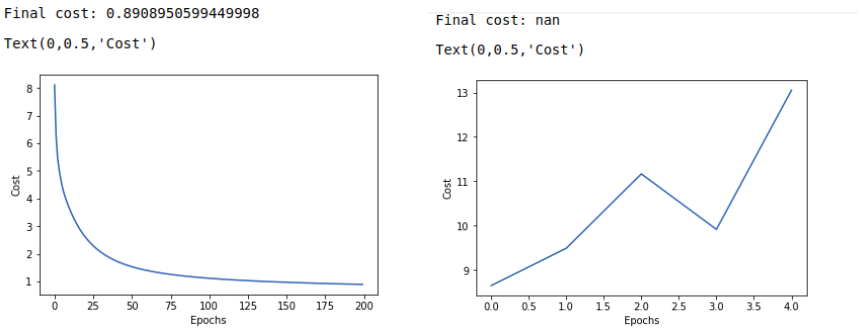
```
[[787   5  16  40   0   0 156   0   2   0]
 [  4 937   1  16   2   0   2   0   1   0]
 [ 15  15 690  17 109   0 129   0   9   0]
 [ 72  34   9 854  43   2  56   0  10   2]
 [  8   5 161  30 756   0 122   0   4   0]
 [  3   0   1   0   2 894   2  33   3  10]
 [ 95   2 108  38  83   0 503   0  23   1]
 [  0   0   0   0   0  58   0 897  10  51]
 [ 16   2  14   5   5  11  30   1 938   1]
 [  0   0   0   0   0  35   0  69   0 935]]
              precision    recall  f1-score   support

           0       0.79      0.78      0.78      1006
           1       0.94      0.97      0.95       963
           2       0.69      0.70      0.70       984
           3       0.85      0.79      0.82      1082
           4       0.76      0.70      0.72      1086
           5       0.89      0.94      0.92       948
           6       0.50      0.59      0.54       853
           7       0.90      0.88      0.89      1016
           8       0.94      0.92      0.93      1023
           9       0.94      0.90      0.92      1039

avg / total       0.82      0.82      0.82     10000
```

(c) (1,Sigmoid,l2)

Figure 5: Confusion Matrix for 64 neurons in hidden layer in order (learning rate, non linear function, normalization of data)



(a) (0.5,ReLU,l2)

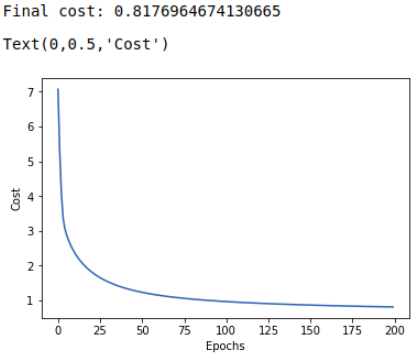
(b) (0.5,Tanh,l2)

Figure 6: Loss plot for 64 neurons in hidden layer in order (learning rate, non linear function, normalization of data)

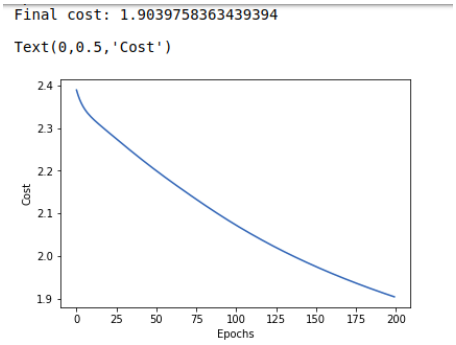
6 Code

6.1 Part I

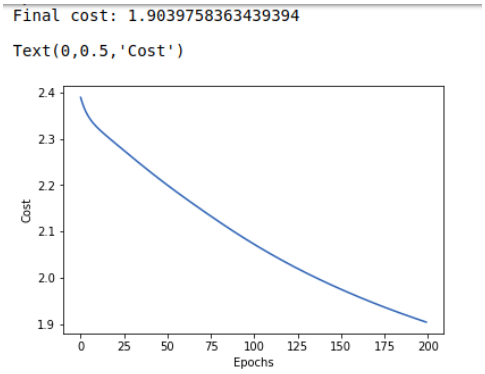
```
1
2
3  # coding: utf-8
4
5  # ## Adding Header files
6
7  # In[24]:
8
9
10 import numpy as np
11 import matplotlib.pyplot as plt
12 from sklearn import preprocessing
13 from sklearn.metrics import classification_report,
       confusion_matrix, f1_score
14 import sklearn.linear_model
15
16
17 # ## Reading and Accessing Data
18
19 # In[72]:
20
21
22 from mnist import MNIST
23 mndata = MNIST('fashionmnist')
24
25 train_images, train_labels = mndata.load_training()
```



(a) (0.85,ReLU,l2)



(b) (0.85,Tanh,l1)



(c) (1,Sigmoid,l2)

Figure 7: Loss plot for 64 neurons in hidden layer in order (learning rate, non linear function, normalization of data)

```
[[566  51  38 119  33   9 201   4  47   1]
 [ 27 806  15 111  46   6  20   1   3   0]
 [ 20  45 381  55 303  25 145   4  59   9]
 [ 86  68  22 615  93  11  55   7  77   2]
 [ 82   8 261  39 298  36 207   0  28   0]
 [ 13   1  19   1   6 533  17 125  36 160]
 [147  15 185  41 127  25 288   4  55   4]
 [  6   3   2   1   2 161  12 677  94  71]
 [ 27   3  70  13  61  74  55  65 592  29]
 [ 18   5  17   6   3 150   4 115  17 700]]
              precision    recall  f1-score   support

           0       0.57      0.53      0.55      1069
           1       0.80      0.78      0.79      1035
           2       0.38      0.36      0.37      1046
           3       0.61      0.59      0.60      1036
           4       0.31      0.31      0.31       959
           5       0.52      0.59      0.55       911
           6       0.29      0.32      0.30       891
           7       0.68      0.66      0.67      1029
           8       0.59      0.60      0.59       989
           9       0.72      0.68      0.70      1035

avg / total       0.55      0.55      0.55     10000
```

(a) (0.01,ReLU,l2)

```
[[323 177  31 467  27  14  84  25  85  21]
 [201 580  80 182 150  88 104   8  12 103]
 [  1   0  18   1  23  16   9   3   1   0]
 [ 43  62 243  32 197 104 140  78 242 198]
 [ 49  57 129  65 237  90 192  38  65  10]
 [ 13  40 117  46 143 121  86 259  93 113]
 [131  59  60  36  47 218 115 279 113 256]
 [ 77  36  43  40  20  62  57  42  61  66]
 [ 46  12 261  31 158  77 194  11  53 113]
 [135   6  11  71   7 215  52 227 259 107]]
              precision    recall  f1-score   support

           0       0.32      0.26      0.28      1254
           1       0.56      0.38      0.46      1508
           2       0.02      0.25      0.03        72
           3       0.03      0.02      0.03      1339
           4       0.23      0.25      0.24       932
           5       0.12      0.12      0.12      1031
           6       0.11      0.09      0.10      1314
           7       0.04      0.08      0.06       504
           8       0.05      0.06      0.05       956
           9       0.11      0.10      0.10      1090

avg / total       0.20      0.16      0.18     10000
```

(b) (0.01,ReLU,l2)

Figure 8: Confusion Matrix for 128 neurons in hidden layer in order (learning rate, non linear function, normalization of data)

```
test_images, test_labels = mndata.load_testing()


### Converting data to Numpy format

In[73]:


train_images = np.asarray(train_images)
train_labels = np.asarray(train_labels)
test_images = np.asarray(test_images)
test_labels = np.asarray(test_labels)


### Normalizing data

In[4]:


def normalize255(X):
    X_new = X/255

    return X_new

def normalizeL1(X):
    X_normalized = preprocessing.normalize(X, norm='l1')

    return X_normalized

```

```
[[712  13  31  56  15  12 162   0  11   3]
 [ 17 905   5  45  12   3  13   0   1   1]
 [ 15  10 614  26 199   7 155   1  20   1]
 [ 83  24   9 787  60   5  47   1  21  11]
 [ 21  14 215  36 571   2 136   0  13   1]
 [  7   0   5   3   1 739   3  72  36  64]
 [146  11 141  51 103   6 400   0  44   1]
 [  1   1   1   1   0  89   2 801  20  79]
 [ 21   0  19   9  12  58  47  11 880  19]
 [  9   0   2   0   0  84   2  90   9 778]]
              precision    recall  f1-score   support

           0       0.69      0.70      0.70      1015
           1       0.93      0.90      0.91      1002
           2       0.59      0.59      0.59      1048
           3       0.78      0.75      0.76      1048
           4       0.59      0.57      0.58      1009
           5       0.74      0.79      0.76       930
           6       0.41      0.44      0.43       903
           7       0.82      0.81      0.81       995
           8       0.83      0.82      0.83      1076
           9       0.81      0.80      0.81       974

avg / total       0.72      0.72      0.72     10000
```

(a) (0.5,ReLU,l2)

```
[[265  71  76 124 160  12 200   9  46  45]
 [ 54 687  79  56  95  13  39   1  42   3]
 [ 58  64 255  82  91  40 196  24  92  69]
 [ 53  58  55 536 164  43  64   5  34  28]
 [151   7 309 112 234  85 106  12  56   0]
 [  7   2  21   5   7 409  21 177  67 131]
 [332  46 106  39 119  49 304   0   8   2]
 [  5  39   3   9   1 179   8 592  47 103]
 [ 37  10  67  13 160  64  64  35 546  14]
 [ 40  35   4  18   2 102  19 130  22 620]]
              precision    recall  f1-score   support

           0       0.26      0.26      0.26      1008
           1       0.67      0.64      0.66      1069
           2       0.26      0.26      0.26       971
           3       0.54      0.52      0.53      1040
           4       0.23      0.22      0.22      1072
           5       0.41      0.48      0.44       847
           6       0.30      0.30      0.30      1005
           7       0.60      0.60      0.60       986
           8       0.57      0.54      0.55      1010
           9       0.61      0.62      0.62       992

avg / total       0.45      0.44      0.45     10000
```

(b) (0.05,ReLU,l2)

```
[[515  47  83  65  48  13 134   2  25  10]
 [ 46 681  20 121  40   2  22   0  10   9]
 [ 46 122 222  79 381  15 282   1 159   0]
 [214  26 139 432  85  23 151  20 149   6]
 [ 79  11 328  18 268   3 142   0  43   1]
 [  7   4  32   1  17 403  26 130  84  75]
 [ 81   9 123 179 102  80 233  11 104  85]
 [  2   7   5  11   3 147   3 619  31  46]
 [ 42  78  54 109  52 163  22  63 289  53]
 [  2   7  11  13   3 104   7 168  72 690]]
              precision    recall  f1-score   support

           0       0.50      0.55      0.52       942
           1       0.69      0.72      0.70       951
           2       0.22      0.17      0.19      1307
           3       0.42      0.35      0.38      1245
           4       0.27      0.30      0.28       893
           5       0.42      0.52      0.47       779
           6       0.23      0.23      0.23      1007
           7       0.61      0.71      0.66       874
           8       0.30      0.31      0.31       925
           9       0.71      0.64      0.67      1077

avg / total       0.43      0.44      0.43     10000
```

(c) (0.05,Tanh,l2)

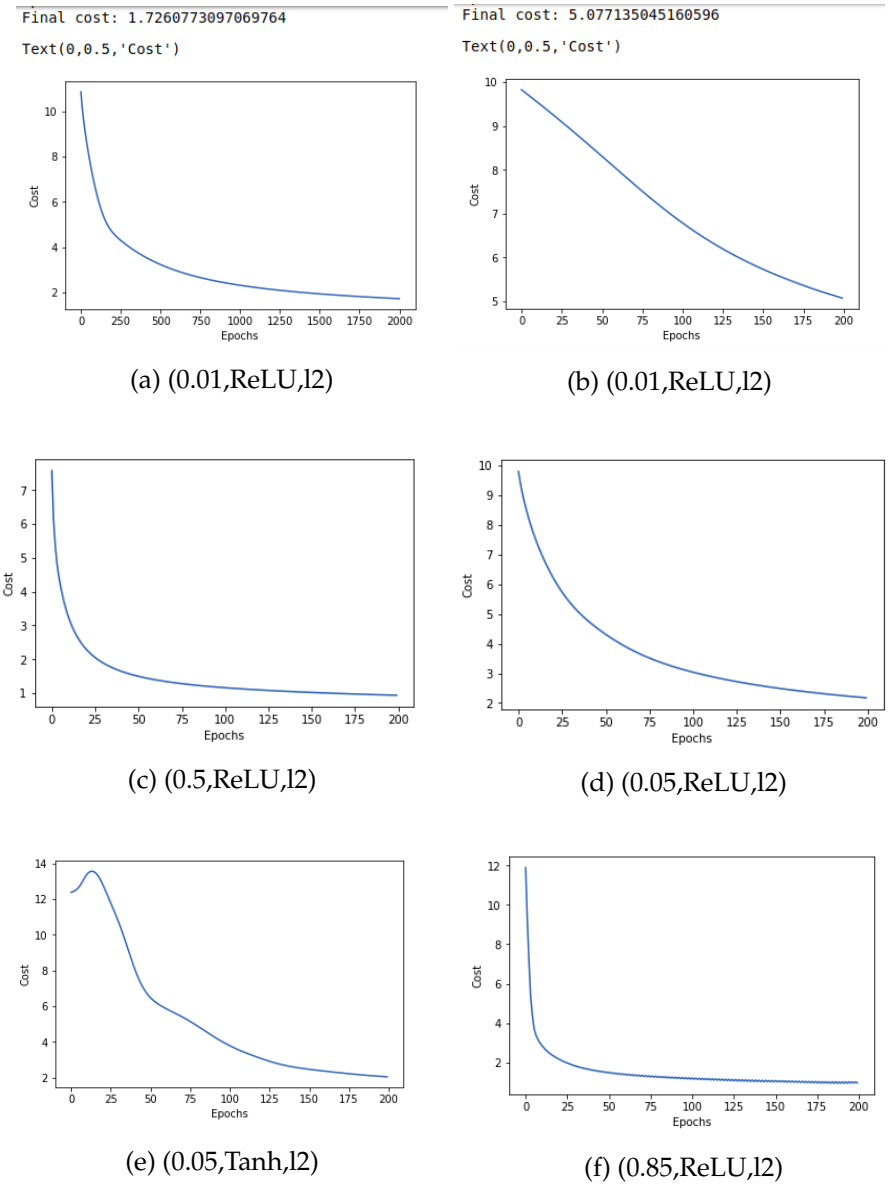
```
[[800  10  39 109  10   4 288   0  11   2]
 [ 12 881   9  33   6   3  11   0   4   1]
 [ 22   6 571  13  78   3 134   0  40   0]
 [ 79  31   8 775  40   8  59   0   6   3]
 [ 32  20 320  83 856   2 439   0  27   0]
 [  5   2   3   2   3 773   5  95  39  51]
 [ 14   0  19   3   5   3  99   0   4   0]
 [  1   0   0   2   0 124   0 799  14  61]
 [ 24   1  18  10  10  34  34   6 840   9]
 [  3   1   2   1   0  65   0  86   2 840]]
              precision    recall  f1-score   support

           0       0.81      0.63      0.71      1273
           1       0.93      0.92      0.92       960
           2       0.58      0.66      0.62       867
           3       0.75      0.77      0.76      1009
           4       0.85      0.48      0.61      1779
           5       0.76      0.79      0.77       978
           6       0.09      0.67      0.16       147
           7       0.81      0.80      0.80      1001
           8       0.85      0.85      0.85       986
           9       0.87      0.84      0.85      1000

avg / total       0.80      0.72      0.75     10000
```

(d) (0.85,ReLU,l2)

Figure 9: Confusion Matrix for 128 neurons in hidden layer in order (learning rate, non linear function, normalization of data)



(a) (0.01,ReLU,l2)

(b) (0.01,ReLU,l2)

(c) (0.5,ReLU,l2)

(d) (0.05,ReLU,l2)

(e) (0.05,Tanh,l2)

(f) (0.85,ReLU,l2)

Figure 10: Loss plot for 128 neurons in hidden layer in order (learning rate, non linear function, normalization of data)

```
[[360  56  66 169 167  14 173   1  17   3]
 [ 26 775  11 114  31   1  18   0  13   3]
 [ 80  13 425  17 210  12 224   3  34   1]
 [273 122   8 553  25   7 113   4   7   0]
 [ 68  18 332  20 407   4 182   0  17   6]
 [  3   0   6   2   2 312   3 251  83 159]
 [137  10 125  92 121  92 218   1  43  19]
 [  6   1   1   6   0 318   1 657   7  61]
 [ 38   5  21  25  30  69  51  27 567 104]
 [  9   0   5   2   7 171  17  56 212 644]]
              precision    recall  f1-score   support

           0       0.36      0.35      0.36      1026
           1       0.78      0.78      0.78       992
           2       0.42      0.42      0.42      1019
           3       0.55      0.50      0.52      1112
           4       0.41      0.39      0.40      1054
           5       0.31      0.38      0.34       821
           6       0.22      0.25      0.23       858
           7       0.66      0.62      0.64      1058
           8       0.57      0.61      0.59       937
           9       0.64      0.57      0.61      1123

avg / total       0.50      0.49      0.50     10000
```

(a) (0.01,Sigmoid,l2)

```
[[734   2   9  34   1   0 115   0   2   0]
 [  3 940   5  16   1   0   1   0   1   0]
 [ 13   9 638  11  91   0  99   0   5   0]
 [ 61  33   8 850  37   1  45   0  10   1]
 [  3   6 142  27 730   0 100   0   1   0]
 [  3   0   2   0   2 908   2  39   7   6]
 [167   7 184  57 132   1 609   0  38   1]
 [  0   0   0   0   0  52   0 900   6  46]
 [ 16   3  12   5   6   7  29   0 930   2]
 [  0   0   0   0   0  31   0  61   0 944]]
              precision    recall  f1-score   support

           0       0.73      0.82      0.77       897
           1       0.94      0.97      0.96       967
           2       0.64      0.74      0.68       866
           3       0.85      0.81      0.83      1046
           4       0.73      0.72      0.73      1009
           5       0.91      0.94      0.92       969
           6       0.61      0.51      0.55      1196
           7       0.90      0.90      0.90      1004
           8       0.93      0.92      0.93      1010
           9       0.94      0.91      0.93      1036

avg / total       0.82      0.82      0.82     10000
```

(b) (1,Sigmoid,l2)

Figure 11: Loss plot for 150 neurons in hidden layer in order (learning rate, non linear function, normalization of data)

```
55 def normalizeL2(X):
56     X_normalized = preprocessing.normalize(X, norm='l2')
57 
58     return X_normalized
59 
60 
61 # In[53]:
62 
63 
64 ## Calling different normalization function
65 
66 #train_images = normalize255(train_images)
67 #test_images = normalize255(test_images)
68 
69 train_images = normalizeL1(train_images)
70 test_images = normalizeL1(test_images)
71 
72 #train_images = normalizeL2(train_images)
73 #test_images = normalizeL2(test_images)
74 
75 
76 # ### Defining classes:
77 
78 # In[6]:
79 
80 
81 class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', '
       Coat',
82                'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

```
83
84
85  # ### Plotting few examples
86
87  # In[7]:
88
89
90  plt.figure(figsize=(10,10))
91  for i in range(25):
92      plt.subplot(5,5,i+1)
93      plt.xticks([])
94      plt.yticks([])
95      plt.grid(False)
96      plt.imshow(train_images[i].reshape((28, 28)), cmap = 'gray')
97      plt.xlabel(class_names[train_labels[i]])
98
99
100 # ## Main Program
101
102 # ### Activation Function
103
104 # In[13]:
105
106
107 def sigmoid(z):
108     s = 1 / (1 + np.exp(-z))
109     return s
110
111 def tanh(x):
112     return np.tanh(x)
113
114 def dtanh(x):
115     return 1. - x * x
116
117 def ReLU(x):
118     return x * (x > 0)
119
120 def dReLU(x):
121     return 1. * (x > 0)
122
123
124 # #### Training
125
126 # In[54]:
127
128
129 X, y = train_images, train_labels
130 classes = 10
131 minval = .0000001
```

```
m = 60000
n_h = 128 # number of neurons in hidden layer
learning_rate = .85

samples = y.shape[0]

y = y.reshape(1, samples)

Y_new = np.eye(classes)[y.astype('int32')]
Y_new = Y_new.T.reshape(classes, samples)

X_train = X.T
Y_train = Y_new

rand_idx = np.random.permutation(m)
X_train, Y_train = X_train[:, rand_idx], Y_train[:, rand_idx]

def compute_multiclass_loss(Y, Y_hat):

    L_sum = np.sum(np.multiply(Y, np.log(Y_hat.clip(min=minval))))
    m = Y.shape[1]
    L = -(1/m) * L_sum

    return L

n_x = X_train.shape[0]

W1 = np.random.randn(n_h, n_x)
b1 = np.zeros((n_h, 1))
W2 = np.random.randn(classes, n_h)
b2 = np.zeros((classes, 1))

X = X_train
Y = Y_train

error_cost = []

for i in range(200):

    Z1 = np.matmul(W1,X) + b1
    # Change for the non linearity
    A1 = sigmoid(Z1)
    Z2 = np.matmul(W2,A1) + b2
    A2 = np.exp(Z2) / np.sum(np.exp(Z2), axis=0)

    cost = compute_multiclass_loss(Y, A2)

    dZ2 = A2-Y
    dW2 = (1./m) * np.matmul(dZ2, A1.T)
```

```
    db2 = (1./m) * np.sum(dZ2, axis=1, keepdims=True)

    dA1 = np.matmul(W2.T, dZ2)

    # Change for the non linearity with derivative
    dZ1 = dA1 * sigmoid(Z1) * (1 - sigmoid(Z1))
    dW1 = (1./m) * np.matmul(dZ1, X.T)
    db1 = (1./m) * np.sum(dZ1, axis=1, keepdims=True)

    W2 = W2 - learning_rate * dW2
    b2 = b2 - learning_rate * db2
    W1 = W1 - learning_rate * dW1
    b1 = b1 - learning_rate * db1

    error_cost.append(cost)
    if (i % 100 == 0):
        print("Epoch", i, "cost: ", cost)

print("Final cost:", cost)
plt.plot(error_cost)
plt.xlabel("Epochs")
plt.ylabel("Cost")


# #### Testing

# In[55]:


samples = test_labels.shape[0]

y = test_labels.reshape(1, samples)

# Converting to one hot encoder
Y_new = np.eye(classes)[test_labels.astype('int32')]
Y_new = Y_new.T.reshape(classes, samples)

X_test = test_images.T
Y_test = Y_new

# Shuffling data
rand_idx = np.random.permutation(samples)
X_test, Y_test = X_train[:, rand_idx], Y_train[:, rand_idx]

Z1 = np.matmul(W1, X_test) + b1

# Change for the non linearity
A1 = ReLU(Z1)
Z2 = np.matmul(W2, A1) + b2
```

```
230 A2 = np.exp(Z2) / np.sum(np.exp(Z2), axis=0)
231
232 predictions = np.argmax(A2, axis=0)
233 labels = np.argmax(Y_test, axis=0)
234
235 plt.subplots(figsize=(10,8))
236 sns.heatmap(confusion_matrix(predictions, labels),annot=True)
237
238 print(confusion_matrix(predictions, labels))
239 print(classification_report(predictions, labels))
240
241
242 # In[39]:
243
244
245 plt.subplots(figsize=(10,8))
246 sns.heatmap(confusion_matrix(predictions, labels),annot=True)
```

References

[1] https://github.com/zalandoresearch/fashion-mnist

[2] http://rohanvarma.me/Neural-Net/

[3] https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

[4] http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/

[5] https://sebastianraschka.com/faq/docs/softmax_regression.html

[6] https://jonathanweisberg.org/post/A%20Neural%20Network%20from%20Scratch%20-%20Part%201/