PROJECT 2

CSE 574: Introduction to Machine Learning
(Fall 2019)

Problem Statement: Implement neural network and convolutional neural network for the task of classification on fashion MNIST dataset.

UB Person No. 50321234
UBIT: amrataan
Name: Amrata Anant Raykar

Table of Contents

ABSTRACT

Like human brain functioning to recognize the patterns or images by self-learning, the technology today can train a model to recognize various forms of patterns by rigorous learning. Training, in other words consists of providing inputs and telling the network what the output should be.

Neural Networks are the series of methods which understands of underlying relationship among the data and predict any unseen data of similar type. This model is acceptive to changing input and learns repeatedly to recognize the relationship.

INTRODUCTION

While training the model initially a large amount of data and the answers are fed. Providing the answers allows the model to adjust its internal weights to learn how to do its job. Passing through various mathematical functions in forward propagation and adjusting the weights and reducing the loss in backward propagation rigorously provides the best set of weights which can be used to recognize any unseen pattern.

Neural networks are sometimes described in terms of their depth, including how many layers they have in between the input and output, which are called as hidden layers of the model. This project

DATASET

In this project we have used Fashion-MNIST dataset. The Fashion-MNIST is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples.

Each example is a 28x28 grayscale image, associated with a label from 10 classes. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns.

The first column consists of the class labels and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.

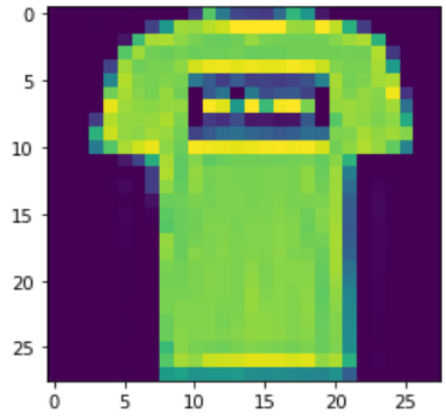
Each training and test example are assigned to one of the labels as shown below table.

1	T-shirt/top
2	Trouser
3	Pullover
4	Dress
5	Coat
6	Sandal
7	Shirt
8	Sneaker
9	Bag
10	Ankle Boot

Sample of an image from the dataset can be seen below:

```
def check_img(i):
    img=X_train[i].reshape((28,28))
    plt.imshow(img)
    plt.show()

#sample of one image in the dataset
check_img(1)
```



DATA PROCESSING

The given raw data is classified into labels(Y) and the features(X). For a specific set of features we have a label. In other words, all the features combinedly define a label.

DATA SCALING

The label set Y has 10 classes to which the features are classified into. This Y is converted into One dimensional hot vector as shown below:

```
y_train[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1.])
```

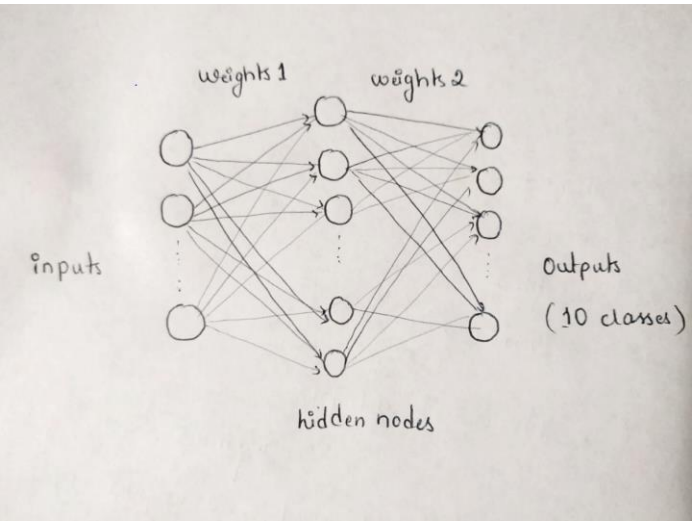
Similarly, the feature set X has values ranging from 0-255, which is normalized by dividing each feature value by 255.

ARCHITECTURE

Neural Network with one hidden layer:

In the first part of the project we have designed a Neural network consist of one hidden layer with 200 hidden nodes and learning rate= 0.001. The entire input is divided into batches of size 1000. Hence, we have 60 batches of data in each epoch.

The view of the model is given below.



LOSS FUNCTION AND GRADIENT:

The hypothesis has theta vector as the weights that need to be refined from a set of values considered randomly. Then we measure how well the algorithm performs on test data using those refined weights. The loss function is defined as

Loss = -sum (log (a2) * Y)

For a better prediction we need to minimize the loss function by finding derivative of loss function with respect to each weight. This is called as Gradient Descent. Given as:

delta_weights2=(a2 - Y)*a1

delta_weights1 = (weights2* (a2-Y))* (a1*(1-a1))*X

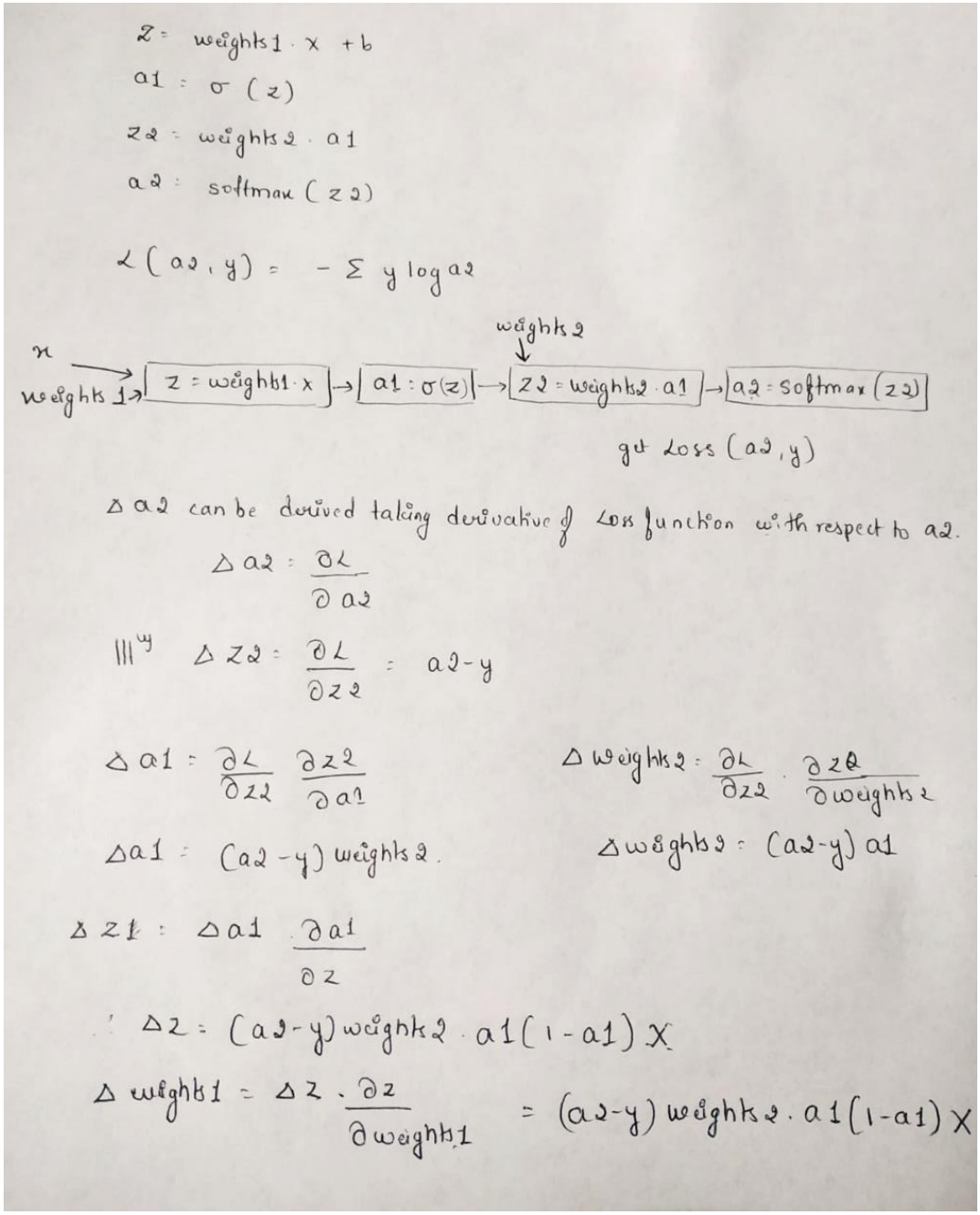
Now subtract the weights by the derivative times the learning rate.

weights2 = weights2 - lr*delta_weights2

weights1 = weights1 - lr*delta_weights1

Passing through various mathematical functions for activation in forward propagation and adjusting the weights and reducing the loss in backward propagation rigorously provides the best set of weights which can be used to recognize any unseen pattern.

Detailed steps of backward propagation is shown in the image below:



Multi-level Neural Network:

Neural networks are sometimes described in terms of their depth, including how many layers they have in between the input and output, which are called as hidden layers of the model. The process involves building, compiling and training the model followed with testing for its accuracy.

We have built a 5-layer neural network. In the first layer we flatten the input data so that 28*28 is flattened out to 784. The second and third layers have sigmoid activation function with 128 and 64 nodes. Dropout layer has been added for regularization which reduces the overfitting. The last dense layer has softmax activation function. Softmax assigns the probability of input to be labeled into each class and we decide the output of the model as the class having maximum probability.

```
#Build a 5-layer neural network
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(128, activation=tf.nn.sigmoid),
    keras.layers.Dense(64, activation=tf.nn.sigmoid),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(10, activation=tf.nn.softmax),
])

model.compile(optimizer='adam',loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Convolution Neural Network:

We have Conv2D layer in 1st layer for the convolution operation to produce a feature map. This is done by sliding a convolution filter over the input. Here we have chosen a feature map with size 3 x 3. 2nd layer we have is a MaxPooling2D layer to reduce the dimensionality of each feature. This helps to shorten training time. Here we have the pooling window with size 2 x 2.

We have Dropout layer as the 3rd layer, a powerful regularization technique. This reduces the over fitting. This disables 30% of the neurons randomly in the phase of learning. Next layer flattens the 3D data to 784 input values of 1D vector. Then we have added 2 more dense layers with relu and softmax activation functions which classifies data to 1 out of 10 classes having maximum probability.

To predict the error rate between the predicted value and the original value we have used Categorical_crossentropy as the loss function and Adam optimizer. It calculates the derivative of the loss function with respect to each weight and subtracts it from the weight.

```
#create the model
con = Sequential()
con.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28,1)))
con.add(MaxPooling2D(pool_size=(2, 2)))
con.add(Dropout(0.3))
con.add(Flatten())
con.add(Dense(128, activation='relu'))
con.add(Dense(10, activation='softmax'))

opt = adam(lr=0.001, decay=1e-6)
con.compile(loss=keras.losses.categorical_crossentropy, optimizer=opt, metrics=['accuracy'])
```

OBSERVATIONS

ACCURACY

Neural Network with one hidden layer:

After training the model we test the accuracy of the prediction made by the model using test data and refined weights.

```
def predict(X_test,y_test):
    z = np.dot(weights1.T, X_test.T)
    a1= sigmoid(z)
    z2= np.dot(weights2.T,a1)
    a2= softmax(z2)
    max_a=np.argmax(a2.T, axis=1)
    return ((y_test==max_a).mean())
```

```
mean_accuracy =predict(X_test,y_test)
print("Test Accuracy=" ,mean_accuracy*100.0 ,"Percent")
```

```
Test Accuracy= 83.87 Percent
```

To verify this, we can use the confusion matrix. Accuracy is given by the ratio of number of correct predictions to the total number of input samples. For this we put the values in confusion matrix and calculate the accuracy for every epoch.

```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test,max_a)
print(confusion_matrix)
N_correct=confusion_matrix.diagonal()
print(" ")
print("Accuracy calculated using Confusion matrix", np.sum(N_correct)/np.sum(confusion_matrix)*100)
```

```
[[930   0  21  25   7   1   6   0  10   0]
 [  8 959   3  23   5   0   0   0   2   0]
 [ 31   3 783  11 162   0   6   0   3   1]
 [ 65   9  16 851  50   0   4   0   5   0]
 [  4   0  97  25 863   0   5   0   6   0]
 [  0   0   0   1   0 938   0  36   1  24]
 [329   0 167  30 225   1 234   0  14   0]
 [  0   0   0   0   0  34   0 927   0  39]
 [ 14   0   9   7   8   5   2   6 949   0]
 [  0   0   0   0   0  10   0  36   1 953]]

Accuracy calculated using Confusion matrix 83.87
```

We can observe that the accuracy is same as the predicted by the model.

Multi-layer neural network:

We evaluate the model against the test data and calculate the accuracy of the prediction using confusion matrix.

```
X_test=np.reshape(X_test,(10000,28,28))
predictions = model.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, np.argmax(predictions, axis=1))
print(confusion_matrix)
N_correct=confusion_matrix.diagonal()
print(" ")
print("Accuracy calculated using Confusion matrix", np.sum(N_correct)/np.sum(confusion_matrix)*100)
```

```
[[782   1   5  94   1   1 106   0  10   0]
 [  1 954   1  36   3   0   3   0   2   0]
 [ 17   1 721  27  65   1 164   0   4   0]
 [ 13   7   5 938  11   1  22   0   3   0]
 [  0   0 103  83 578   0 235   0   1   0]
 [  0   0   0   1   0 946   0  34   1  18]
 [116   1  68  79  32   0 687   0  17   0]
 [  0   0   0   0   0  34   0 936   0  30]
 [  2   1   1   8   2   3  10   4 969   0]
 [  0   0   0   0   0   9   1  34   0 956]]

Accuracy calculated using Confusion matrix 84.67
```

We can also do this using evaluate function of the model as below. Notice the same accuracy given in both the ways.

```
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Loss", loss * 100)
print("Test Accuracy", accuracy * 100)
```

```
10000/10000 [==============================] - 1s 64us/sample - loss: 0.4080 - acc: 0.8467
Test Loss 40.79920841693878
Test Accuracy 84.67000126838684
```

Convolution neural network:

We evaluate the model against the test data and calculate the accuracy of the prediction.

```
#test for the test data and set the accuracy
score = con.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0]*100, "Percent")
print('Test accuracy:', score[1]*100, "Percent")
```

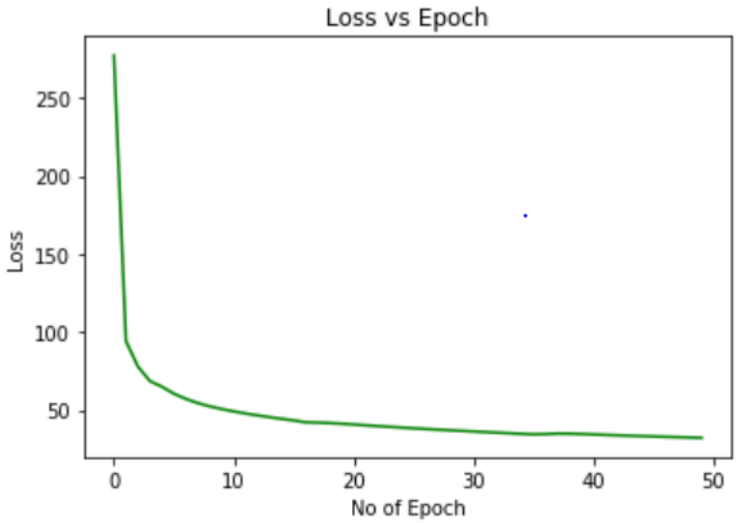
```
Test loss: 35.58673131465912 Percent
Test accuracy: 86.83 Percent
```

LOSS FUNCTION

The main purpose of the gradient descent is to decrease the loss/cost function with its repeated subtraction from the previous loss value. Our model shows that loss values decreases as the number of epochs are increased.

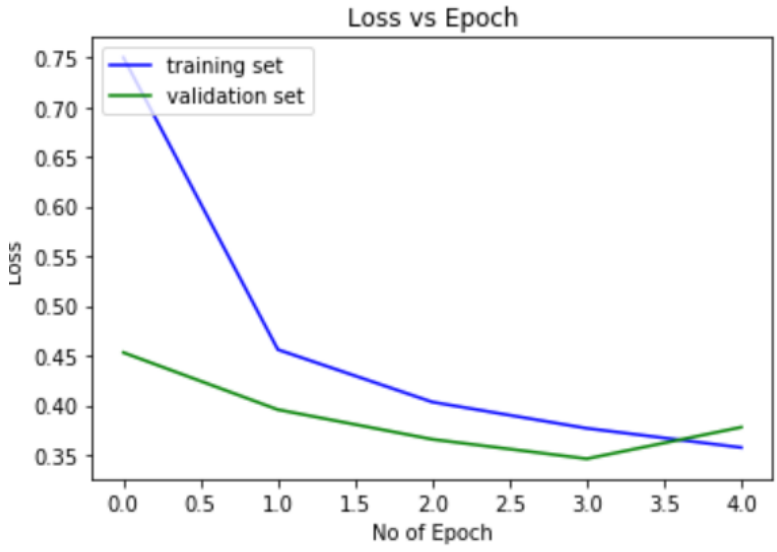
Neural Network with one hidden layer:

The graph of Loss vs No. of epochs for Neural Network with one hidden layer is given below.



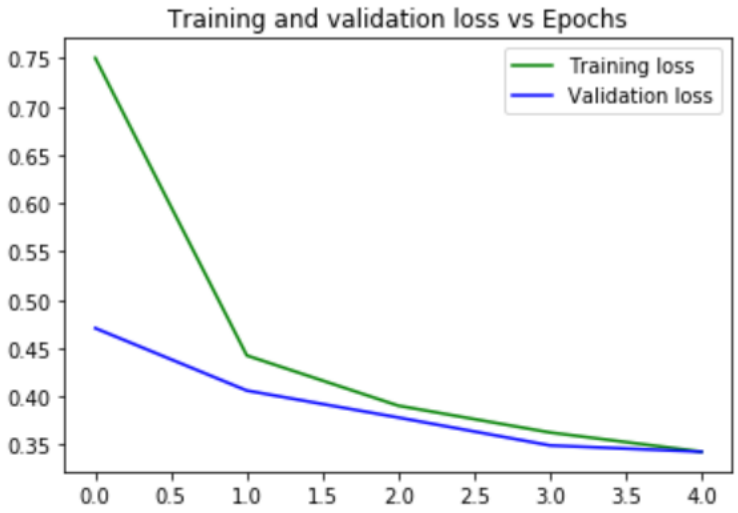
Multi-layer neural network:

The graph of Loss vs No. of epochs for Multi-layer neural network is given below.

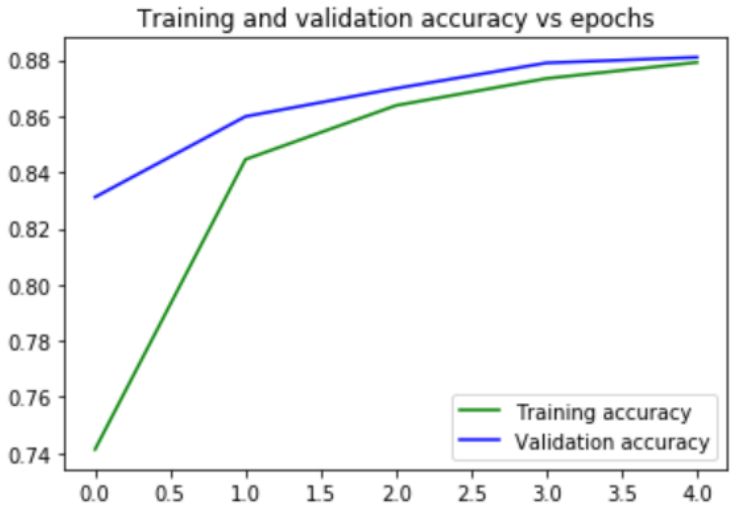


Convolution neural network:

The graph of Loss vs No. of epochs for Convolution neural network is given below.



The graph of Training and validation accuracy is given below:



CONCLUSION

Our project successfully implements neural network and convolutional neural network for the task of classification into 10 classes with the accuracy of 84 percent in one hidden layer NN , 85 percent in multi-layer NN and 88 percent in convolution NN.

Project helps us understand practical use of hidden layers, drop-off layer, different activation functions, convolution neural network, tuning hyper-parameters and visualizing the results along with the theoretical knowledge.