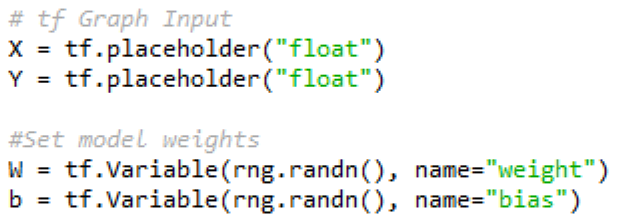
NAME-ISTIAQUE MANNAFEE SHAIKAT
MATRICULATION NUMBER-303527

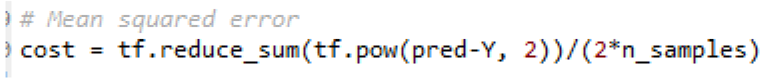
1. Explaining the code: (Univariant Linear Regression)

- For x a random uniform distributed value in created between 1 to 100 of size 1000.
- Then the noise is added in y using the equation(y = 0.5 * x + 2 + noise) where noise = np.random.normal(mu, sigma, [1000,]
- The data is spitted into training and testing data.
- Two placeholders are created X and Y
- Two variables are created W and bias



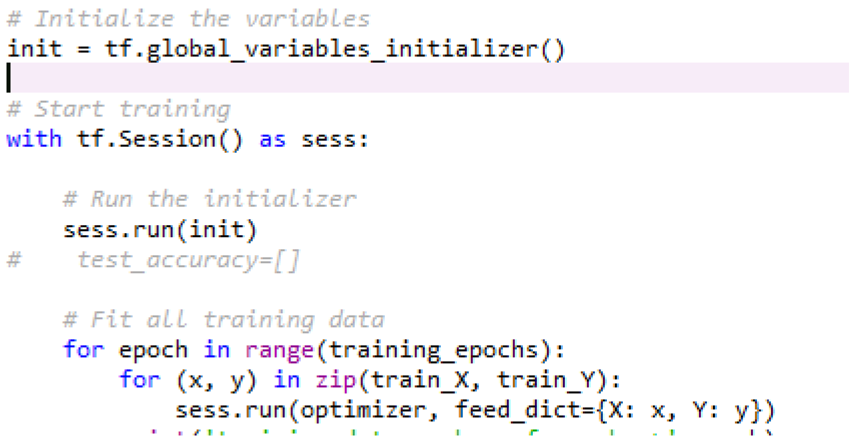
```
# tf Graph Input
X = tf.placeholder("float")
Y = tf.placeholder("float")

#Set model weights
W = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")
```

- Mean squared is used to find the cost and tensorflow GradientDecentOptimizer function is used.



```
# Mean squared error
cost = tf.reduce_sum(tf.pow(pred-Y, 2))/(2*n_samples)
```

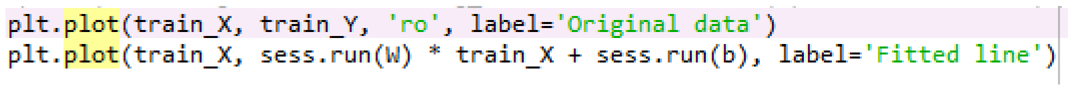
- All the variable in initialized using init = tf.global_variables_initializer() in a session then data is trained in the optimizer



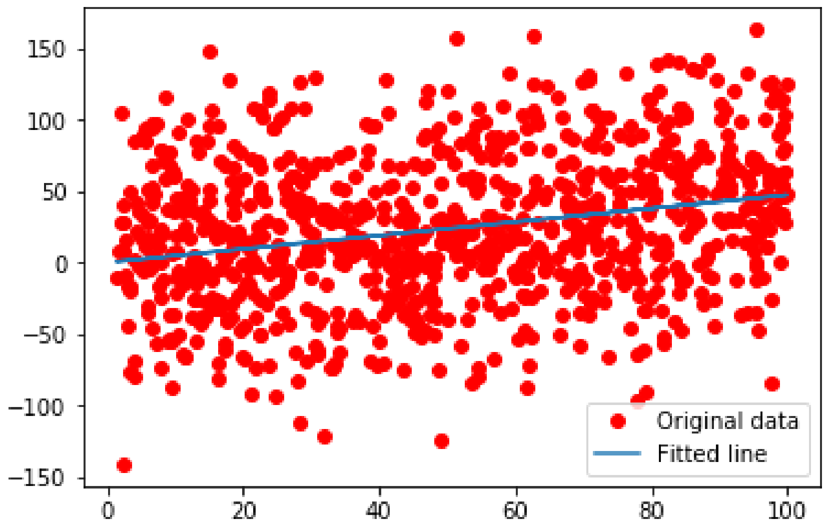
```
# Initialize the variables
init = tf.global_variables_initializer()

# Start training
with tf.Session() as sess:

    # Run the initializer
    sess.run(init)
#    test_accuracy=[]

    # Fit all training data
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})
```

<u>Scatter plot</u>

- Training data is used to train the model to get the best fitted line depending on the data.



```
plt.plot(train_X, train_Y, 'ro', label='Original data')
plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')
```



- The above diagram is for learning rate=0.0001 and epochs=100

2. Explanation: Multivariate Linear Regression

<u>Preprocessing the data</u>

```
#preprocessing the data
df=df.convert_objects(convert_numeric=True)
df=df.dropna()

y= np.array(df[0],dtype='float32')
x=np.array(df.loc[:,1:6],dtype='float32')


x = preprocessing.normalize(x)
```

- The data is read and converted to numeric
- Then the NA values are dropped in the dataset
- The values of x is normalized to get better accuracy
- Then the data is split to 90/10 as training and testing data.

<u>Code explanation</u>

- In order to make it multivariant linear regression exe 1A code is changed by bring changes in the tensor flow graph.

```
X = tf.placeholder(tf.float32, [None,n])
Y= tf.placeholder(tf.float32, [None,1])
#
W = tf.Variable(tf.random_normal([1,n]), name="w")
b = tf.Variable(tf.constant(0.1), name="b")
```

- X is changed to number of features with data type of float.32
- W variable is changed to n number of features
- Three cost functions are used for the optimizer

```
# Mean squared error
cost_mse = tf.reduce_sum(tf.pow(pred-Y, 2))/(2*n)
cost_mae= tf.reduce_mean(tf.abs(Y - pred))
cost_rmse=tf.reduce_mean(tf.sqrt(tf.square(Y - pred)))
```

- Gradients decent optimizer is used to train the data

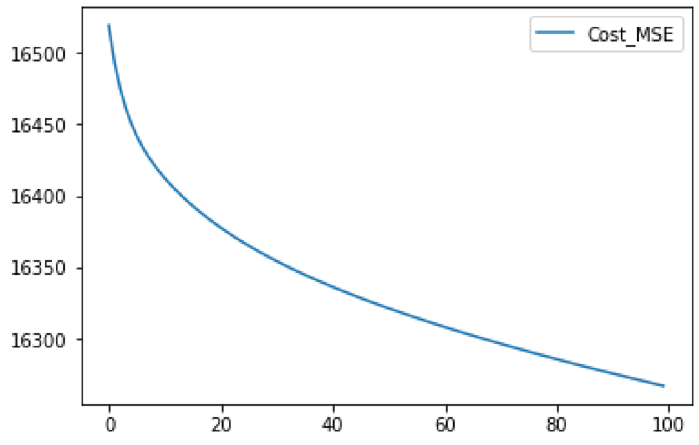
```
## Training using Gradient Descent to minimize cost
with tf.name_scope("train") as scope:
  optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_rmse)
```

- Then a new session is opened to train the data for epoch of 100 after initializing the global variables

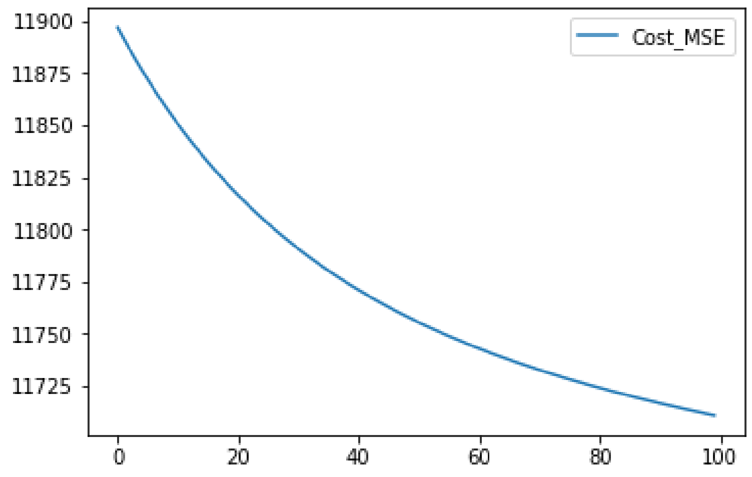
```
init = tf.global_variables_initializer()


with tf.Session() as sess:

    # Run the initializer

    sess.run(init)

#     Fit all training data
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            x = x.reshape(-1,n)
            y = y.reshape(-1,1)
            sess.run(optimizer, feed_dict={X: x, Y: y})
```

MSE VS EPOCH PLOTS

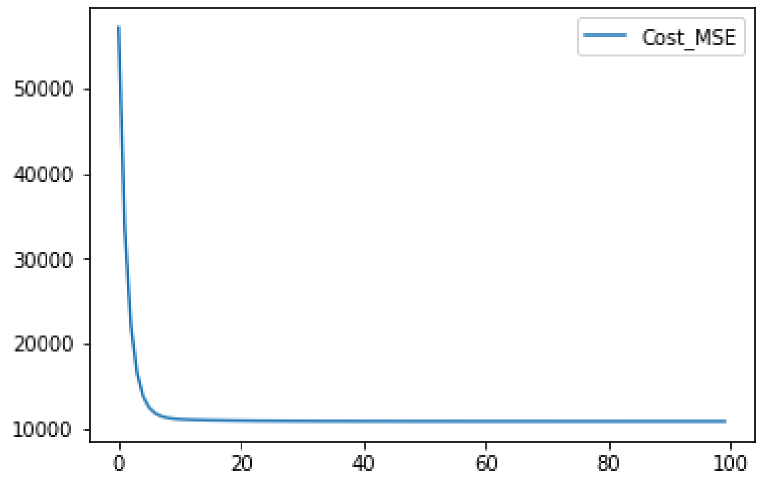
Learning rate=1 and epoch =100



Learning rate=0.1 and epoch =100

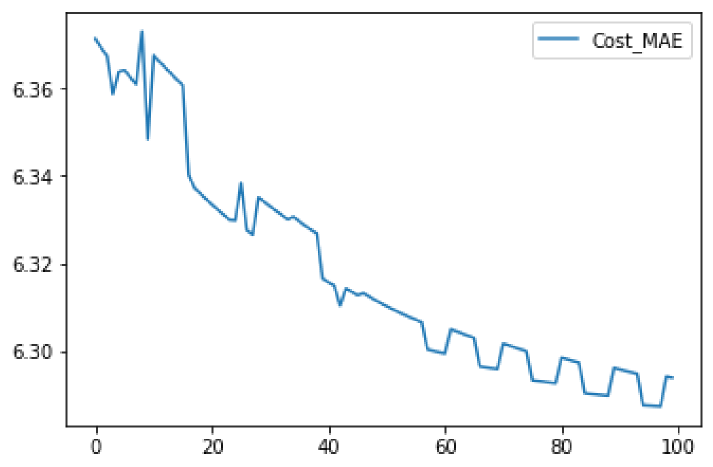


Learning rate=0.001 and epoch =100

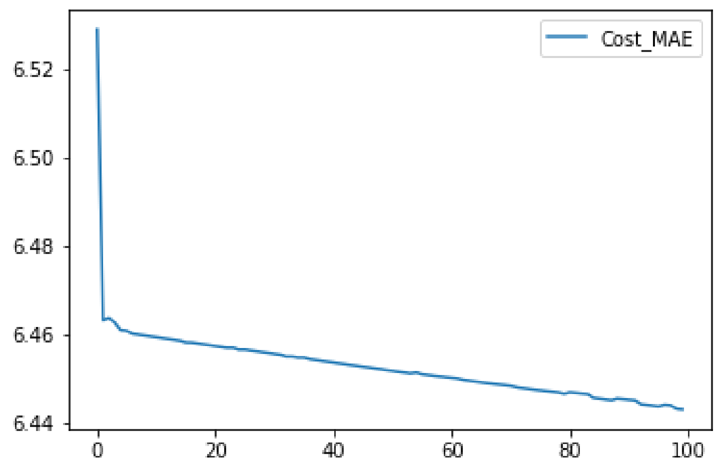


MAE VS EPOCH PLOTS

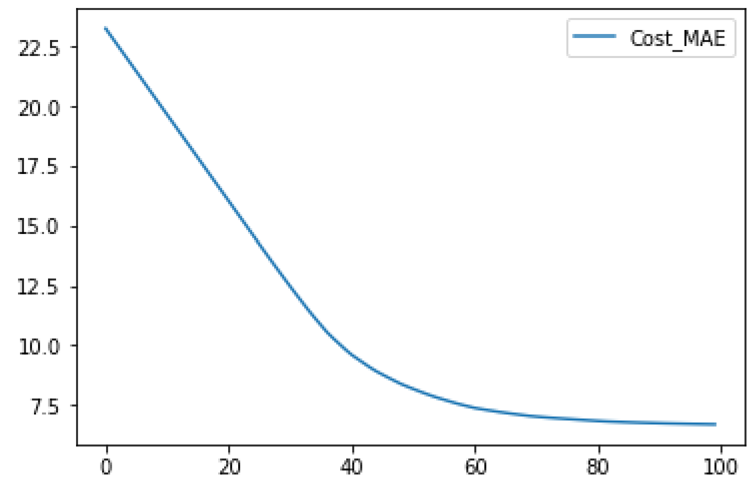
Learning rate=1 and epoch =100



Learning rate=0.1 and epoch =100

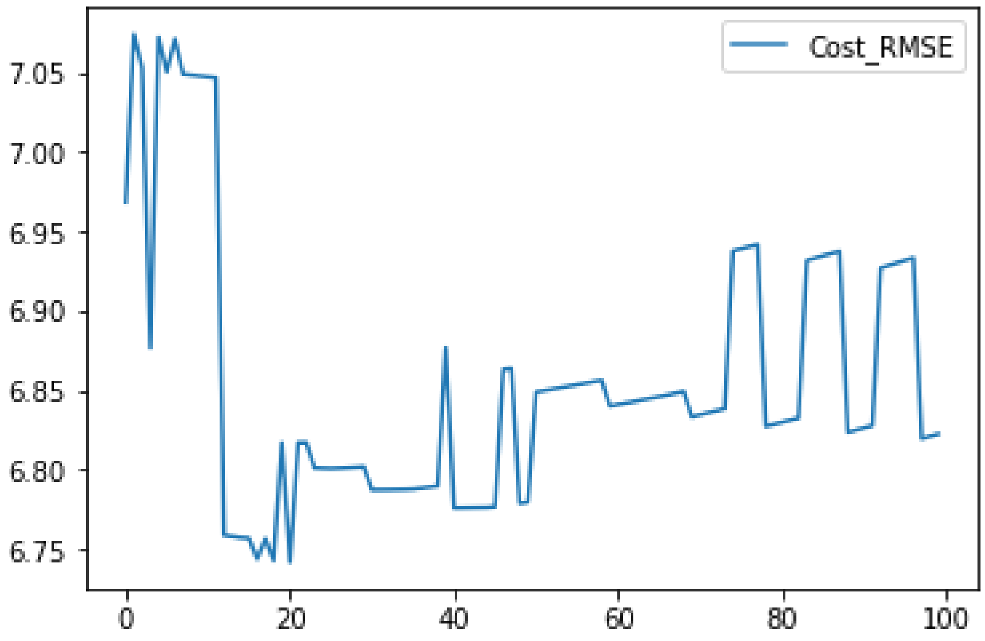


Learning rate=0.001 and epoch =100

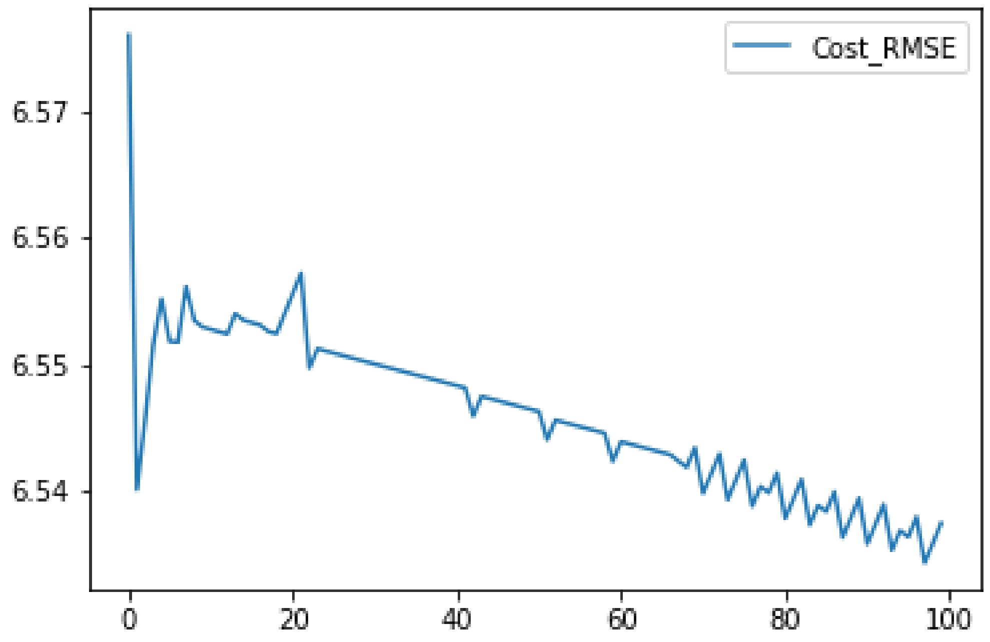


RMSE VS EPOCH PLOTS

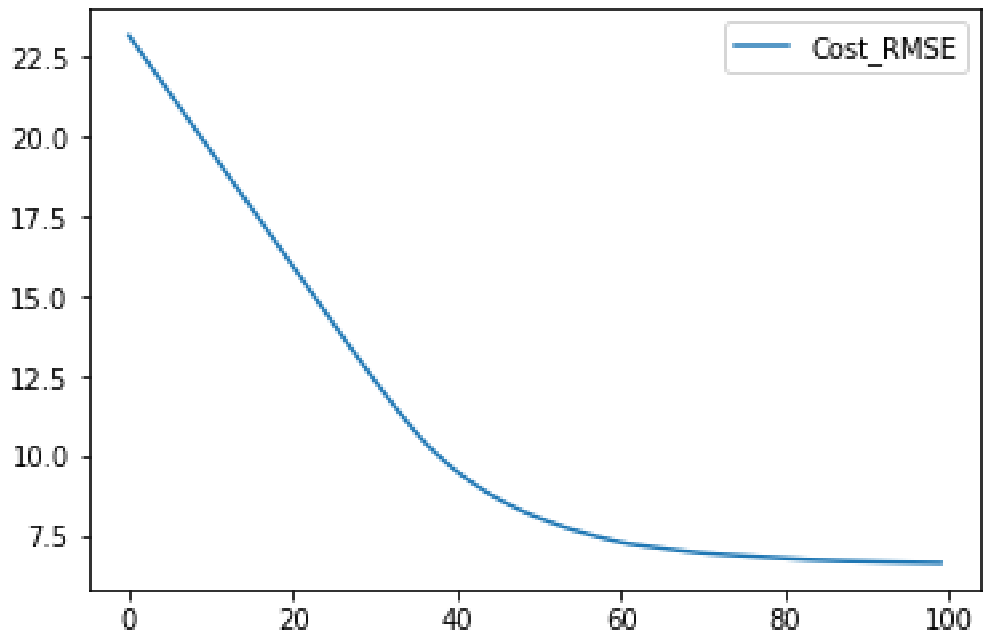
Learning rate=1 and epoch =100



Learning rate=0.1 and epoch =100



Learning rate=0.001 and epoch =100



Exercise 2: Logistic Regression on the Olivetti faces dataset

Preprocessing data

```
dataset = fetch_olivetti_faces(shuffle=True, random_state=rng)
faces = dataset.data
x = dataset.data
y = dataset.target

y = y.reshape(-1,1)
ybefore = np.copy(y)
oneHot = OneHotEncoder()
oneHot.fit(y)
y = oneHot.transform(y).toarray()
```

Code explanation:

- The data set is fetched from the sklearn.datasets
- The features and data is in x and the label or target is in y
- As the label is categorical so it is converted to numerical using one hot encoder and saved to an array
- The data is then spilt to training and testing data which is 90/10 where n is the number of features and m is the number of training examples

```
train_X, test_X, train_Y, test_Y = train_test_split(x, y, train_size=0.9)
m= train_X.shape[0] #number of training examples
n= train_X.shape[1] #features
z=train_Y.shape[1]
train_Y = train_Y.reshape(m,-1)
```

- The tensor Graph is created for X,Y,W,b

```
# There are n columns in the feature matrix

X = tf.placeholder(tf.float32, [None, n])
Y = tf.placeholder(tf.float32, [None, z])
W = tf.Variable(tf.zeros([n,z]), name="w")
b = tf.Variable(tf.zeros([1,z]), name="b")
```

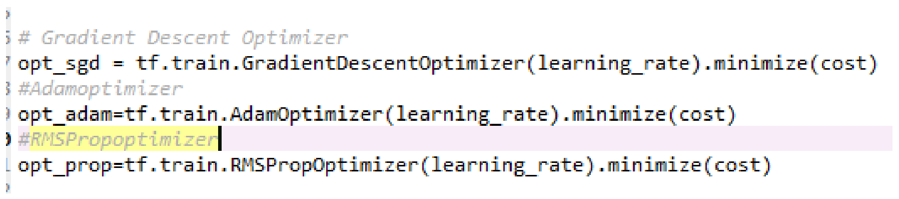
- The model

```
# Hypothesis
Y_hat = tf.nn.softmax(tf.matmul(X, W)+b)
```

- Cross entropy cost function is used for the cost

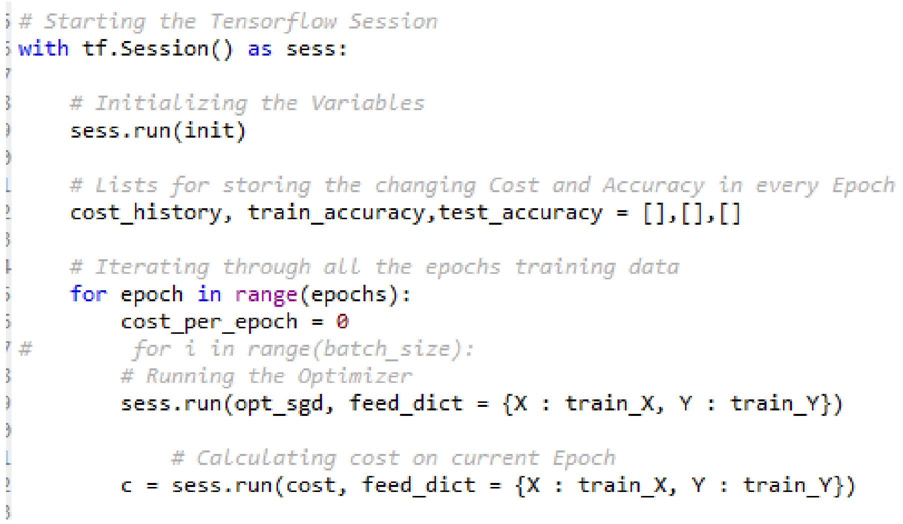
```
#Cross Entropy Cost Function
cost = -tf.reduce_mean(Y*tf.log(Y_hat))
```

- Three different optimizer is used to test that model



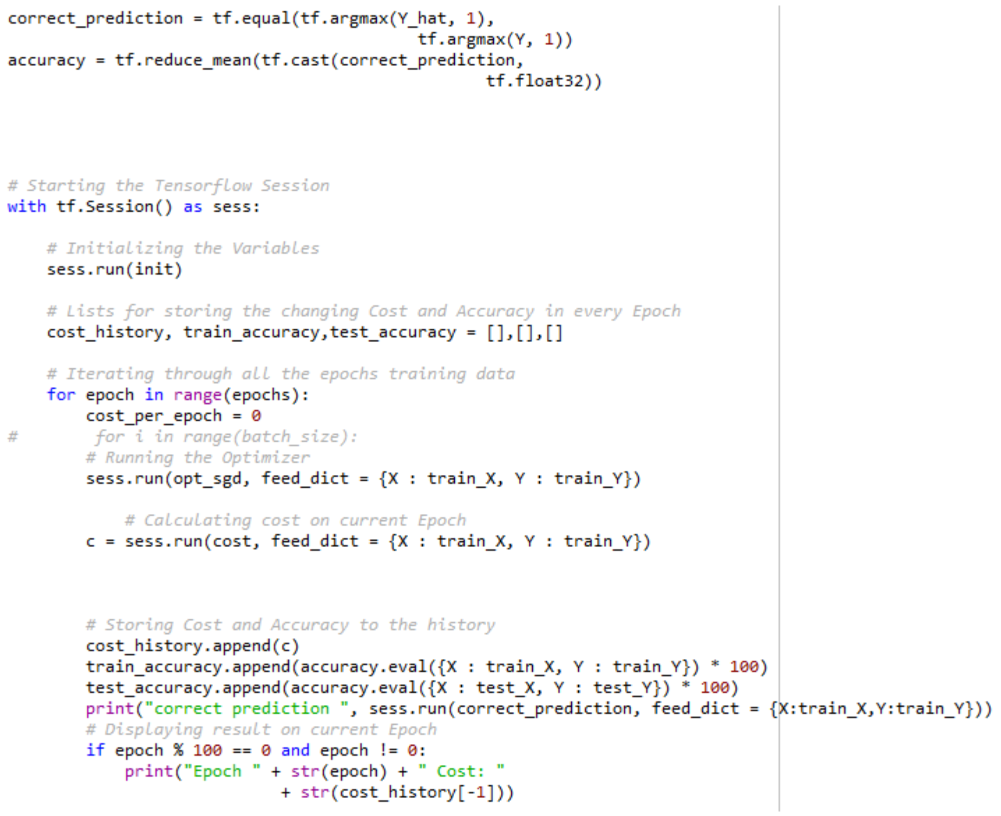
```
# Gradient Descent Optimizer
opt_sgd = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
#Adamoptimizer
opt_adam=tf.train.AdamOptimizer(learning_rate).minimize(cost)
#RMSPropoptimizer
opt_prop=tf.train.RMSPropOptimizer(learning_rate).minimize(cost)
```

- Then a new session is opened to train the data for epoch of 100 after initializing the global variables



```
# Starting the Tensorflow Session
with tf.Session() as sess:

    # Initializing the Variables
    sess.run(init)

    # Lists for storing the changing Cost and Accuracy in every Epoch
    cost_history, train_accuracy,test_accuracy = [],[],[]

    # Iterating through all the epochs training data
    for epoch in range(epochs):
        cost_per_epoch = 0
#          for i in range(batch_size):
        # Running the Optimizer
        sess.run(opt_sgd, feed_dict = {X : train_X, Y : train_Y})

            # Calculating cost on current Epoch
        c = sess.run(cost, feed_dict = {X : train_X, Y : train_Y})
```

- Cost and accuracy is evaluated for using each optimizer



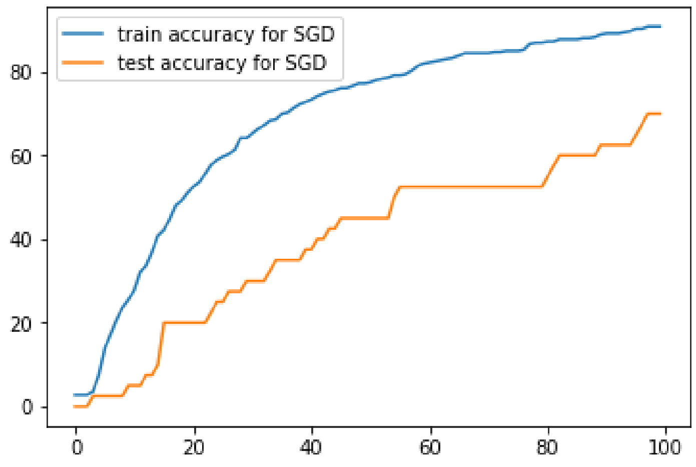
```
correct_prediction = tf.equal(tf.argmax(Y_hat, 1),
                                          tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction,
                                                 tf.float32))



# Starting the Tensorflow Session
with tf.Session() as sess:

    # Initializing the Variables
    sess.run(init)

    # Lists for storing the changing Cost and Accuracy in every Epoch
    cost_history, train_accuracy,test_accuracy = [],[],[]

    # Iterating through all the epochs training data
    for epoch in range(epochs):
        cost_per_epoch = 0
#          for i in range(batch_size):
        # Running the Optimizer
        sess.run(opt_sgd, feed_dict = {X : train_X, Y : train_Y})

            # Calculating cost on current Epoch
        c = sess.run(cost, feed_dict = {X : train_X, Y : train_Y})



        # Storing Cost and Accuracy to the history
        cost_history.append(c)
        train_accuracy.append(accuracy.eval({X : train_X, Y : train_Y}) * 100)
        test_accuracy.append(accuracy.eval({X : test_X, Y : test_Y}) * 100)
        print("correct prediction ", sess.run(correct_prediction, feed_dict = {X:train_X,Y:train_Y}))
        # Displaying result on current Epoch
        if epoch % 100 == 0 and epoch != 0:
            print("Epoch " + str(epoch) + " Cost: "
                            + str(cost_history[-1]))
```

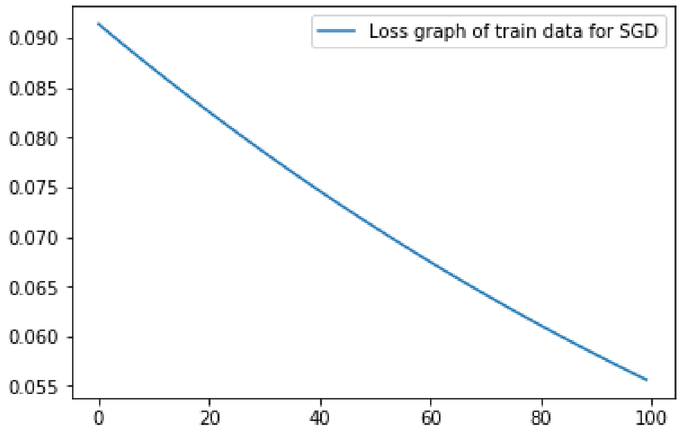
Graph Analysis:

- **Gradient Decent Optimizer (train/test graph) and loss on training set**

Learning rate =1 epoch=100

Training Accuracy: 91.94444417953491 %
Testing Accuracy: 80.0000011920929 %

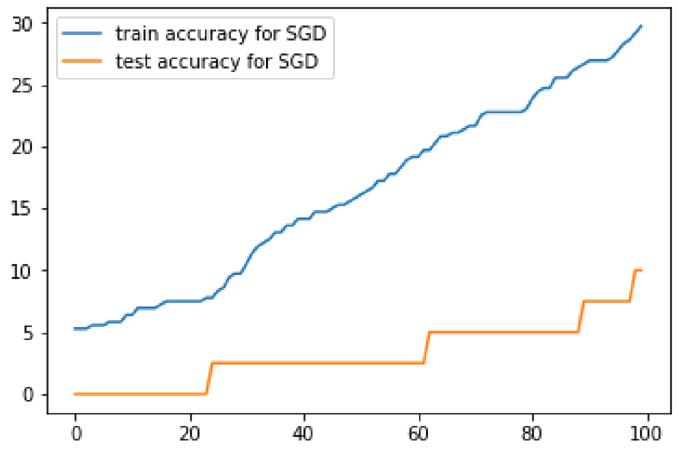


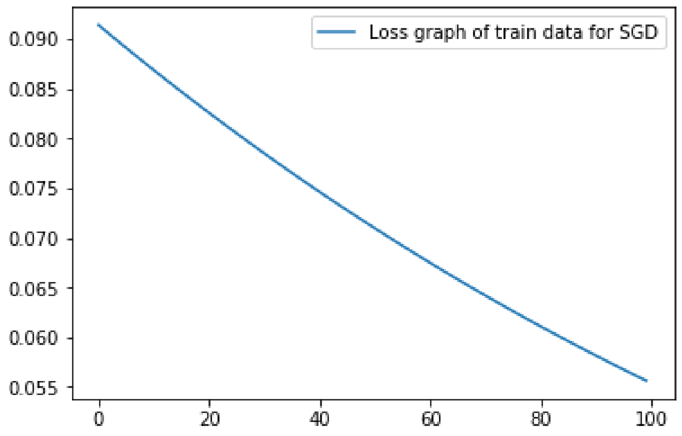


Learning rate =0.1 epoch=100

Training Accuracy: 29.722222685813904 %

Testing Accuracy: 10.000000149011612 %

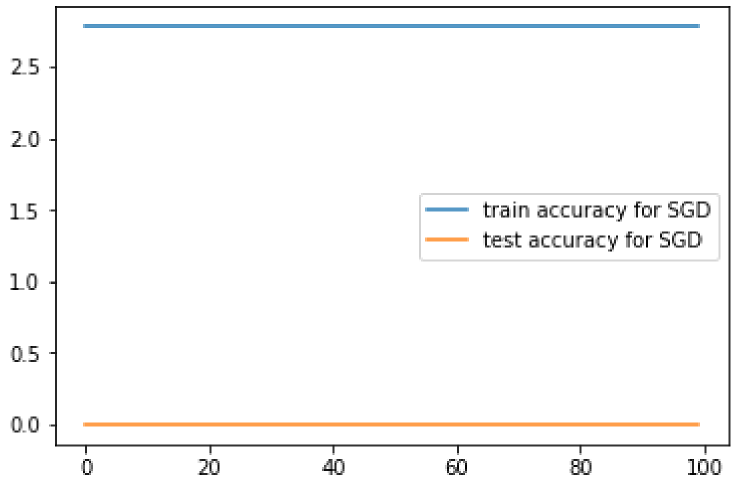


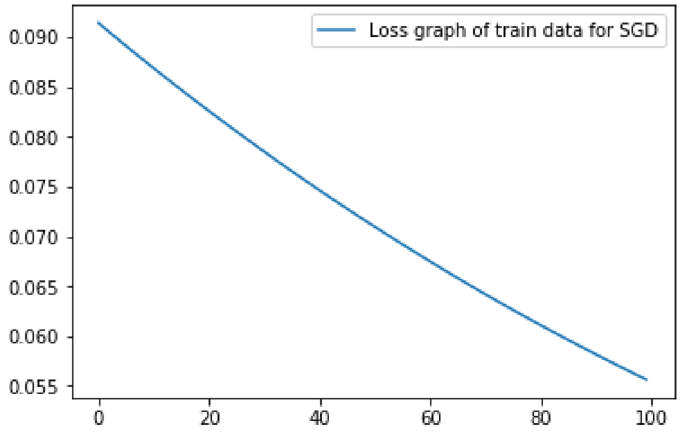


Learning rate =0.01 epoch=100

Training Accuracy: 2.777777798473835 %

Testing Accuracy: 0.0 %

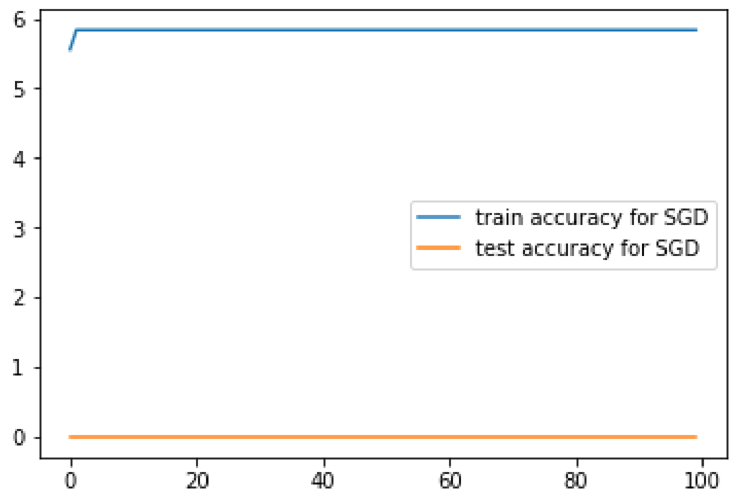


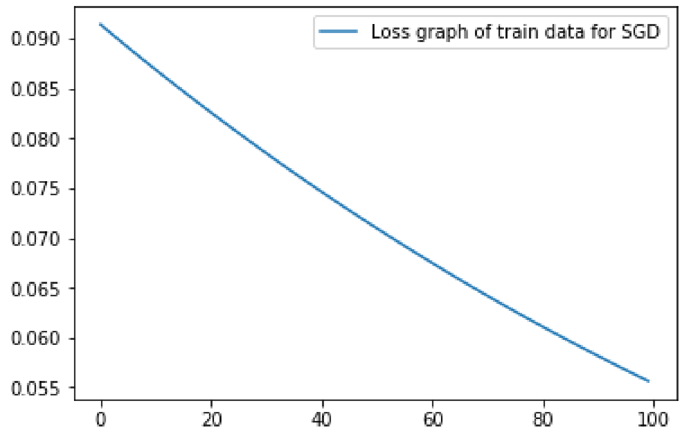


Learning rate =0.0001 epoch=100

Training Accuracy: 5.833333358168602 %

Testing Accuracy: 0.0 %

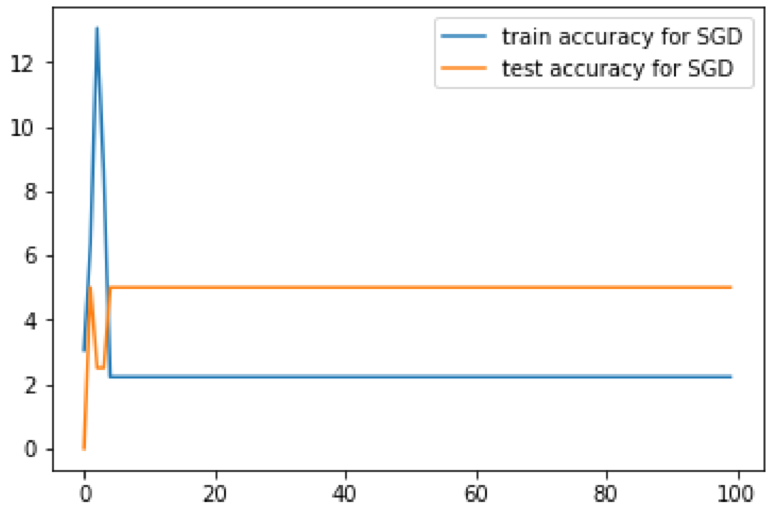


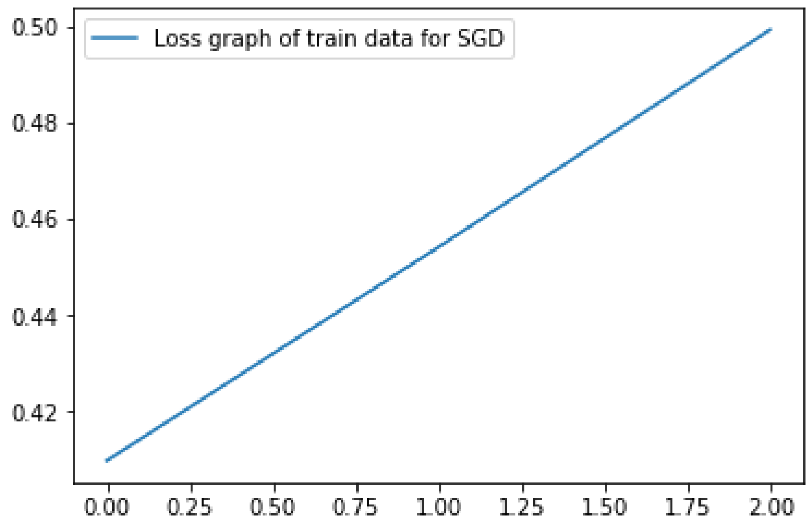


- **Adam optimizer (train/test graph) and loss on training set.**

  Learning rate =0.01 epoch=100

  Training Accuracy: 2.222222276031971 %
  Testing Accuracy: 5.000000074505806 %

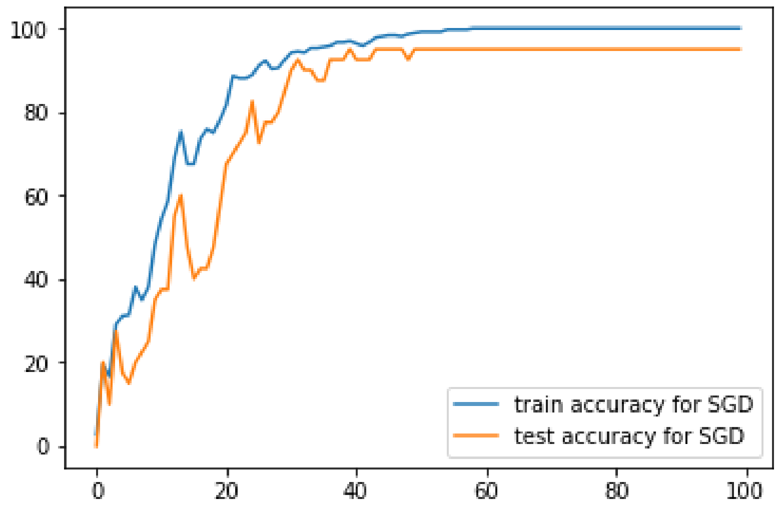


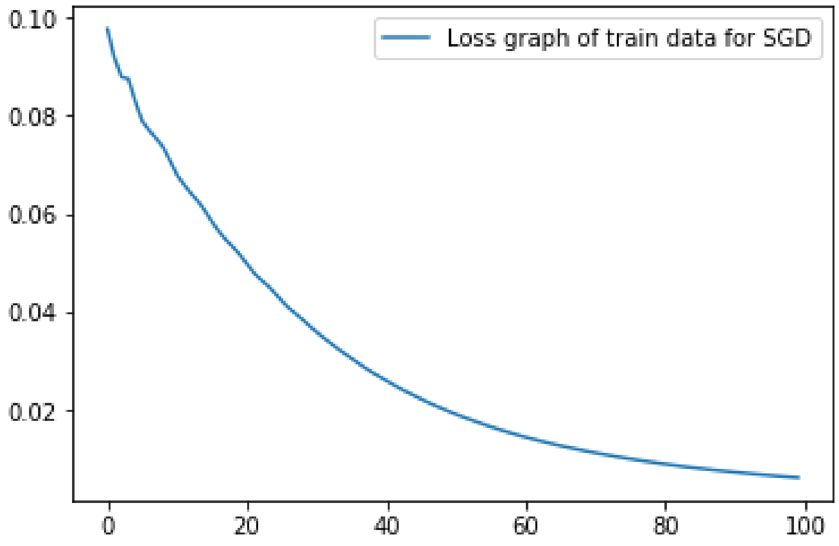


Learning rate =0.001 epoch=100

Training Accuracy: 100.0 %

Testing Accuracy: 94.9999988079071 %

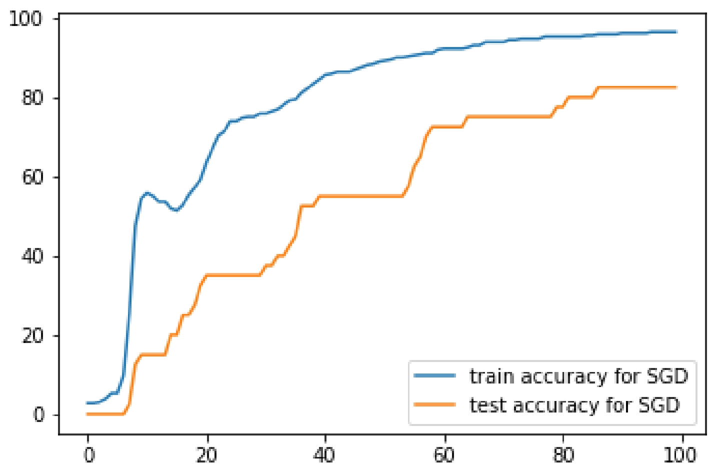


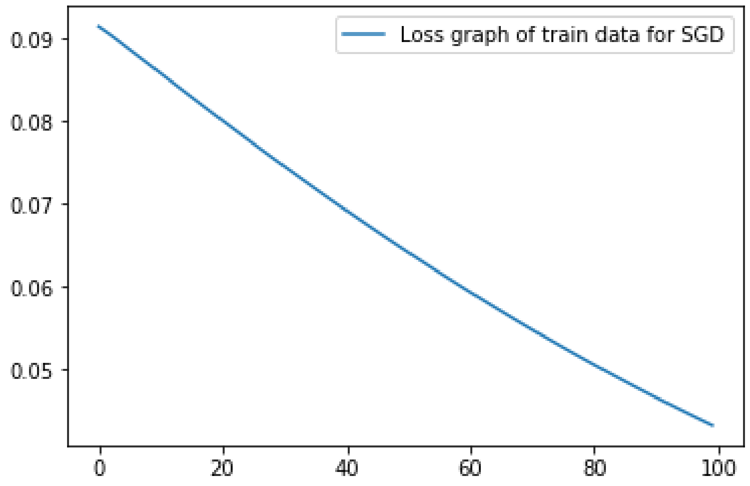


Learning rate =0.0001 epoch=100

Training Accuracy: 96.38888835906982 %

Testing Accuracy: 82.4999988079071 %



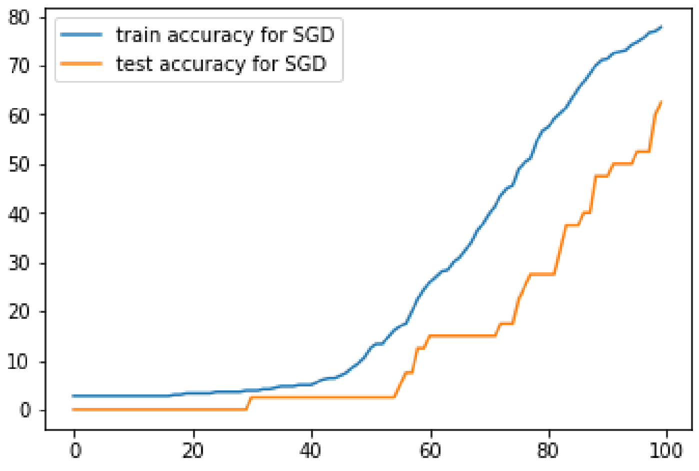


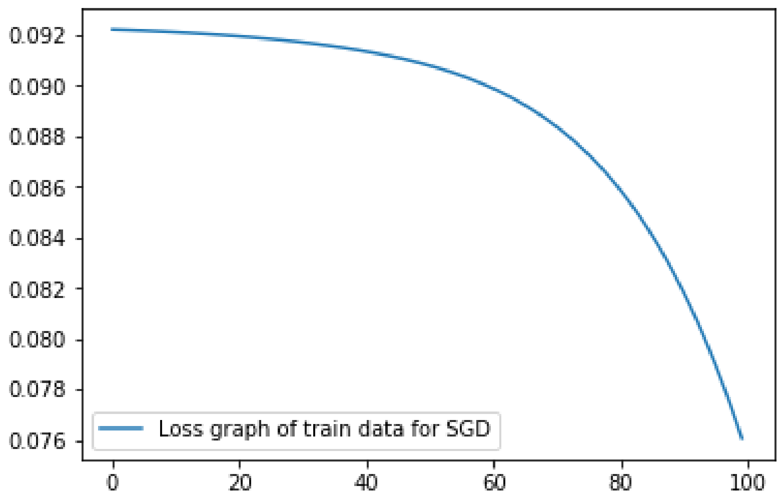
- **RMS Prop Optimizer (train/test graph) and loss on training set**

  Learning rate =0.01 epoch=100

  Training Accuracy: 77.77777910232544 %

  Testing Accuracy: 62.5 %

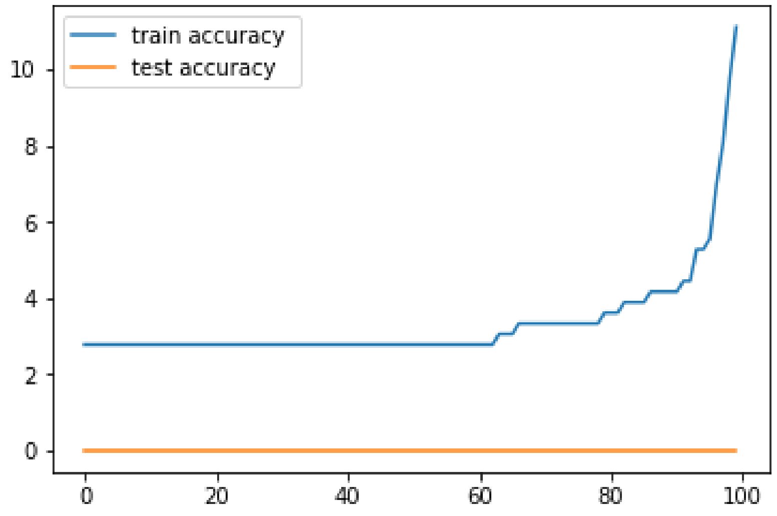


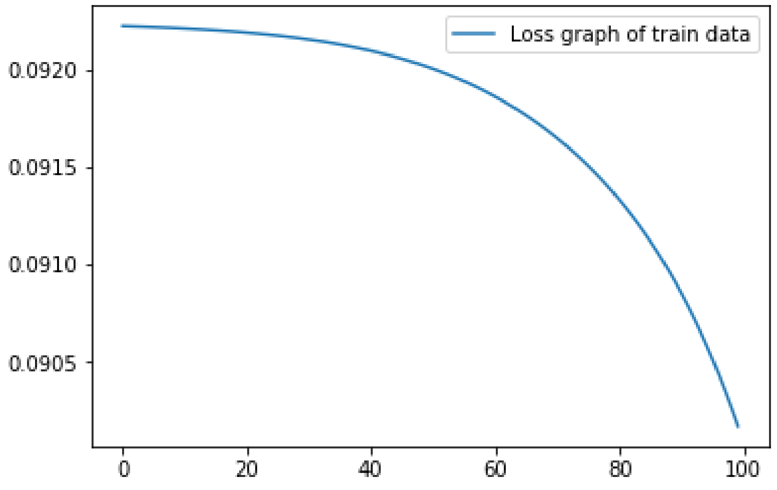


<u>Learning rate =0.001 epoch=100</u>

Training Accuracy: 11.11111119389534 %

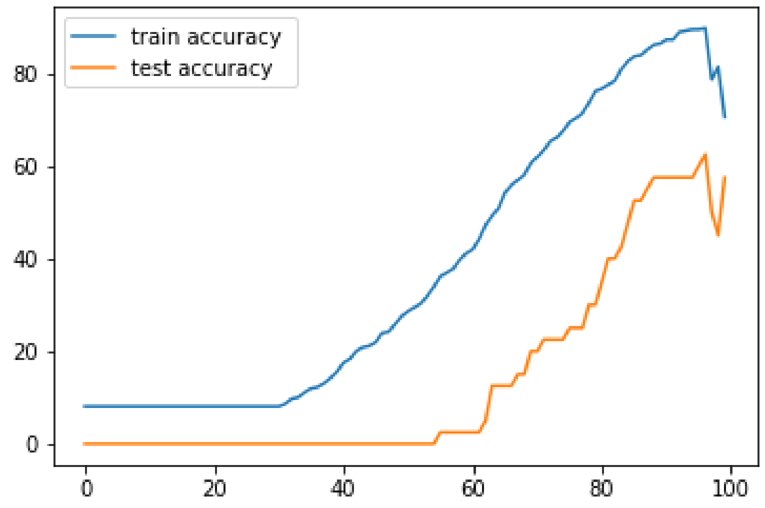
Testing Accuracy: 0.0 %

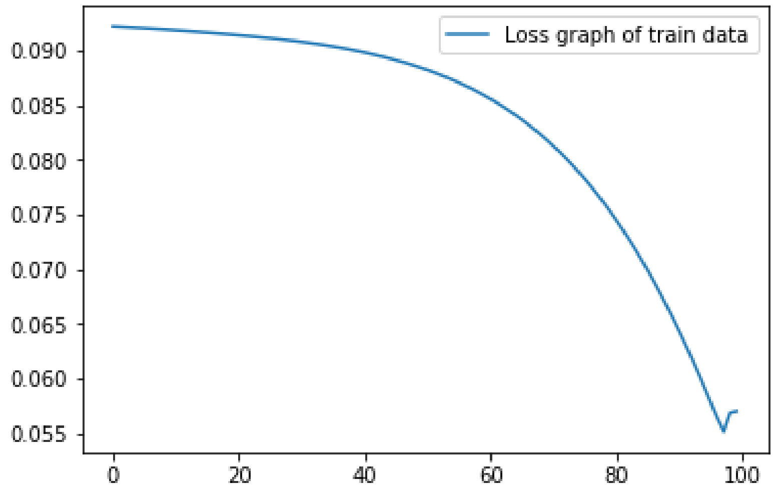




Learning rate =0.03 epoch=100

Training Accuracy: 11.11111119389534 %
Testing Accuracy: 0.0 %


train accuracy
test accuracy


Loss graph of train data