Introduction

Here I have performed logistic regression on the data set that is not used in the class

Objectives

To perform the logistic regression on the given data set, predict the output for the given input and show the graph in Tensor Board

Approaches/Methods

In the beginning, I choose the MNIST dataset which is simple computer vision dataset. Then the dataset is loaded by using the pandas dataframe. For calculating the accuracy of the dataset, we will split the data into two sets. They are training dataset and test dataset. Then the model for logistic regression is built. Then the predictions will be made by using the model and the data. The error will be minimized by using the cross entropy. I created a TensorFlow session to train the logistic regression model with the training data and accuracy of the model is tested using the testing data.

Workflow

1-> Select the dataset

2-> Import the dataset

3-> Split the dataset

4-> Build the logistic regression model

5-> Minimize the loss

6-> Train the model and calculate the accuracy

Dataset

MNIST Dataset

Parameters

- Number of execution: 150 (the size of the dataset)
- Learning rate for optimizer: 0.001

Evaluation

First import the dataset that you have selected by using the pandas library in the Python. Pandas is a Python package providing fast, flexible and expressive data structures designed to make working with “relational” or “labeled” data both easy and Intuitive

```
# Import the data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)


import tensorflow as tf
```

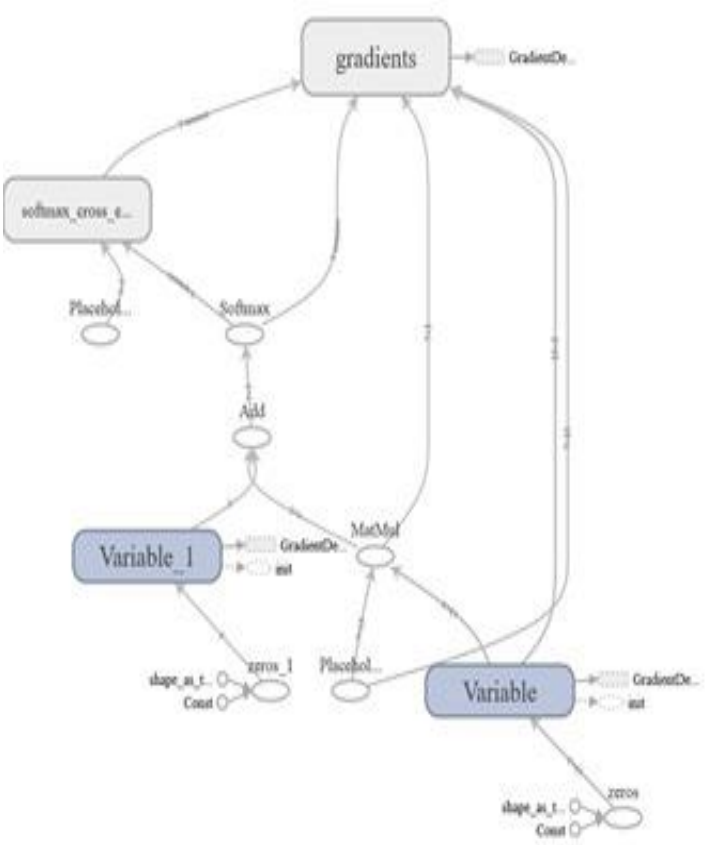
.

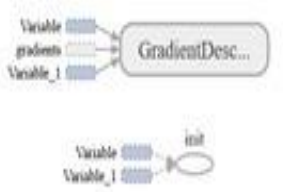
Then the model is built for logistic regression and tested by using the test data and accuracy is calculated.

```
# Import the data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)


import tensorflow as tf

# Set parameters
learning_rate = 0.01
training_iteration = 30
batch_size = 100
display_step = 2

# TF graph input
x_intercept = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28=784
y_intercept = tf.placeholder("float", [None, 10]) # 0-9 digits recognition => 10 classes

# Create a model

# Set model weights
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

# Construct a linear model
model = tf.nn.softmax(tf.matmul(x_intercept, W) + b) # Softmax

# Minimize error using cross entropy
# Cross entropy
cost_function = -tf.reduce_sum(y_intercept*tf.log(model))
# Gradient Descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_function)

# Initializing the variables
init = tf.initialize_all_variables()

# Launch the graph
with tf.Session() as sess:
    sess.run(init)

    writer = tf.summary.FileWriter('./graphs/linear_reg', sess.graph)
    # Training cycle
    for iteration in range(training_iteration):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            # Fit training using batch data
            sess.run(optimizer, feed_dict={x_intercept: batch_xs, y_intercept: batch_ys})
            # Compute average loss
            avg_cost += sess.run(cost_function, feed_dict={x_intercept: batch_xs, y_intercept: batch_ys})/total_batch
        # Display logs per eiteration step
        if iteration % display_step == 0:
            print ("epoch:", '%04d' % (iteration + 1), "cost=", "{:.9f}".format(avg_cost))
```

```
writer.close()
print ("Tuning completed!")

# Test the model
predictions = tf.equal(tf.argmax(model, 1), tf.argmax(y_intercept, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(predictions, "float"))
print ("Accuracy:", accuracy.eval({x_intercept: mnist.test.images, y_intercept: mnist.test.labels}))
```

Graph in Tensor Board:





Output:

```
"C:\Users\sai smaran chinthala\Anaconda3\python.exe" "C:/Users/sai smaran chinthala/PycharmProjects/DL1/venv/Scripts/final.py"
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From C:\Users\sai smaran chinthala\Anaconda3\lib\site-packages\tensorflow\python\util\tf_should_use.py:118: initialize_all_variables (from tensorf
Instructions for updating:
Use `tf.global_variables_initializer` instead.
2018-04-06 23:04:02.073330: I C:\tf_jenkins\workspace\rel-win\M\windows\PY\36\tensorflow\core\platform\cpu_feature_guard.cc:140] Your CPU supports instructions that
epoch: 0001 cost= 30.424720828
epoch: 0003 cost= 21.013457454
epoch: 0005 cost= 20.123671417
epoch: 0007 cost= 19.689746709
epoch: 0009 cost= 19.390671196
epoch: 0011 cost= 19.057297189
epoch: 0013 cost= 18.828820505
epoch: 0015 cost= 18.746175204
epoch: 0017 cost= 18.672458917
epoch: 0019 cost= 18.529580192
epoch: 0021 cost= 18.459544759
epoch: 0023 cost= 18.407405419
epoch: 0025 cost= 18.283036899
epoch: 0027 cost= 18.076609118
epoch: 0029 cost= 18.029298112
Tuning completed!
Accuracy: 0.9229

Process finished with exit code 0
```

Conclusion

The loss depends on the number of iterations. As the iterations increases the loss decreases. The model accuracy is 93%, this shows that our model is 93% accurate in predicting the results for the given inputs for the above dataset.