Midterm Project

Classify Iris using ANN(DNN)

Dept. of CSE, Yonsei University

2016323246 Junyoung Jang

In this report, I used Tensorflow Library which easily apply Neural Network to a data set. (I agree with that it is necessary manually to make the algorithm and code, but we can expect more exact result as using Library.) This report introduce each steps that are involved to Computing Environment, Problem Setting, Compared Simulation result using each method and Python Code.

0. Computing Environment

Unlike MATLAB, Tensorflow is sensitive to computing environment. If you want to run my code, you should set the same environment at least.

- Python version : 3.6.0
- IDE : Pycharm Community 2016.3
- Library : Anaconda, Tensorflow 1.0.0

1. Problem Setting

It used Iris data of UCI in this report. The data set contains 3 classes (Setosa, Versicolor and Virginica) of 50 instances each, where each class refers to a type of iris plant. It also include 4 Features (Sepal Length, Sepal Width, Petal Length and Petal Width) at each class. Compared each features of Classes is represented as belows :

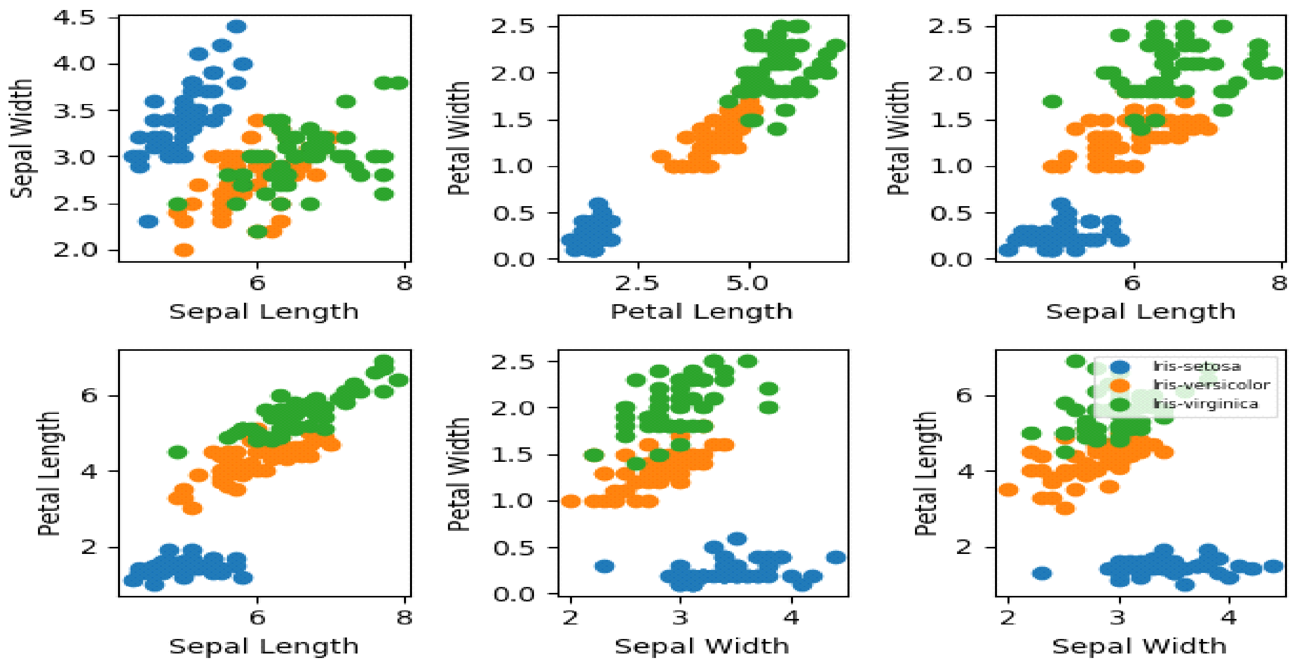

Figure 1. Correlated figure of each features of Classes
(Blue: Sentos, Red: Versicolor and Green: Virginica)

We have to know each class has classified characters. For example, Petal Width and Sepal Length, Petal Width and Petal Length etc. To classify the Iris problem, I simply set as below:

- Main Algorithm : ANN(DNN) and Softmax regression
- Fix random seed because it make stable result.
- Default #Hidden Layer : 5
- #Neuron of each Hidden Layer : 100
- Training Epoch : 1000
- Learning Rate : 0.01

2. Simulation Result

Before simulation result, It has many methods to classify data. For example, what use optimizer method(Adagradient, Adam) or what use initialization(Xavier and He) or what use activation function etc. Therefore, in this section, I analyze advantage and disadvantage of each method. First, I deal with activation functions which are “Sigmoid” and “ReLu” function.

2-1. Activation function

It exist many activation function to apply Neural Network. However, in this report, I only deal with “Sigmoid” and “ReLu” function. Basically, Equation of “Sigmoid” and “ReLu” function is as follows:

$$Sigmoid : f(x) = \frac{1}{1+e^{-x}}, \quad f'(x) = f(x)(1-f(x))$$

$$ReLu : f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}, \quad f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

I simulated each activation functions increasing the number of hidden layer (#0 ~ #10) of Neural Network. In the weight initialization case, I used He’s initialization. I will introduce Initialization in next step. Simulation result is as follows:

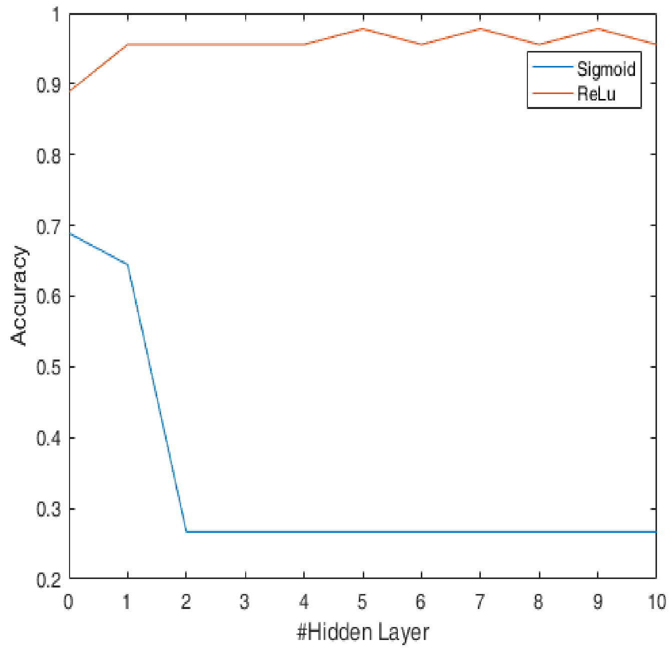

Figure 2. Compared Accuracy of Sigmoid and ReLu (Blue: Sigmoid, Red: ReLu, x-axis is the number of hidden layer and y-axis is accuracy)

#Hidden	Accuracy	
	Sigmoid	ReLu
0	0.688889	0.888889
1	0.644444	0.955556
2	0.266667	0.955556
3	0.266667	0.955556
4	0.266667	0.955556
5	0.266667	0.977778
6	0.266667	0.955556
7	0.266667	0.977778
8	0.266667	0.955556
9	0.266667	0.977778
10	0.266667	0.955556

“ReLu” function has better performance than “Sigmoid” function. Because “Sigmoid” function tend to vanish gradient when x grows to inifinite large. (ex. $f'(x) = f(x)(1-f(x)) = 1\times(1-1)=0$). But “ReLu” function do not tend to vanish gradient. We can also check advantage of “ReLu” function through cost value as bellows:

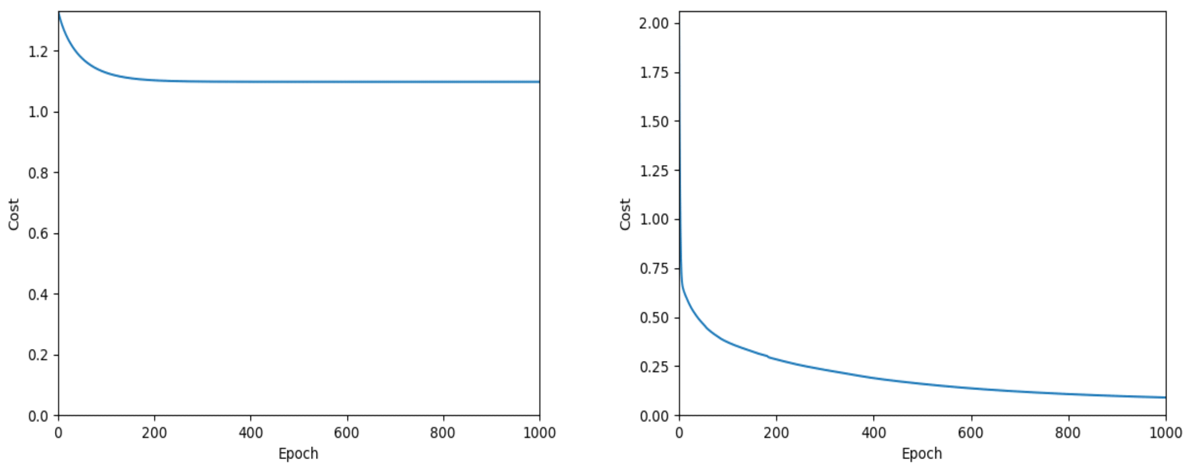

Figure 3. Compared cost(loss) of Sigmoid and ReLu (Left: Sigmoid, Right: ReLu, x-axis is the epoch and y-axis is cost)

2-2. Initialization

We knew "ReLu" function is better than "Sigmoid" function in previous step. In this step, I introduce two initialization method. Of course, in this simulation, I set "ReLu" function as activation function. Originally, Neuron Network is constructed by zero value of initial weight. But these problem will be blow up. To solve the problem, many initialization methods are invented. For example, RBM, Xavier and He's initialization. In this report, I introduce Xavier and He's intialization. Xavier and He's Initialization are very simple as follows: (I wrote python code using numpy library)

$$Xavier : W = np.random.randn(in,out)/np.sqrt(in)$$

$$He \quad : W = np.random.randn(in,out)/np.sqrt(in/2)$$

Simulation result is as belows:

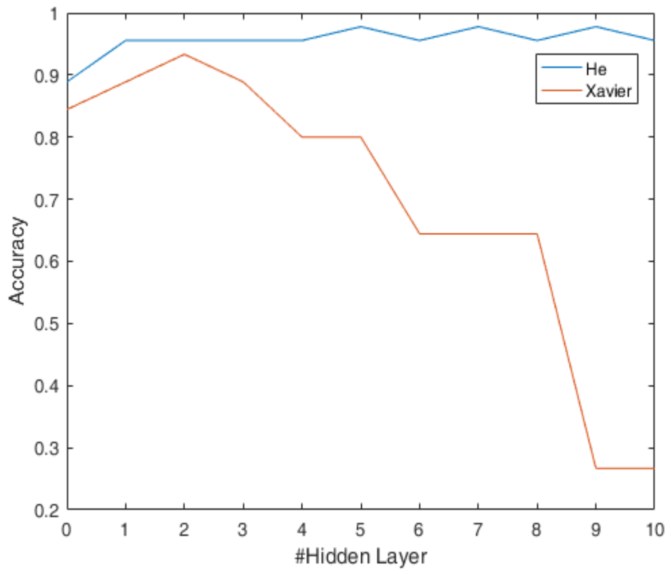

Figure 4. Compared Accuracy of Xavier and He (Blue: He, Red: Xavier, x-axis is the number of hidden layer and y-axis is accuracy)

#Hidden	Accuracy	
	He	Xavier
0	0.888889	0.844444
1	0.955556	0.888889
2	0.955556	0.933333
3	0.955556	0.888889
4	0.955556	0.800000
5	0.977778	0.800000
6	0.955556	0.644444
7	0.977778	0.644444
8	0.955556	0.644444
9	0.977778	0.266667
10	0.955556	0.266667

And Cost value of each initialization methods is as follows:

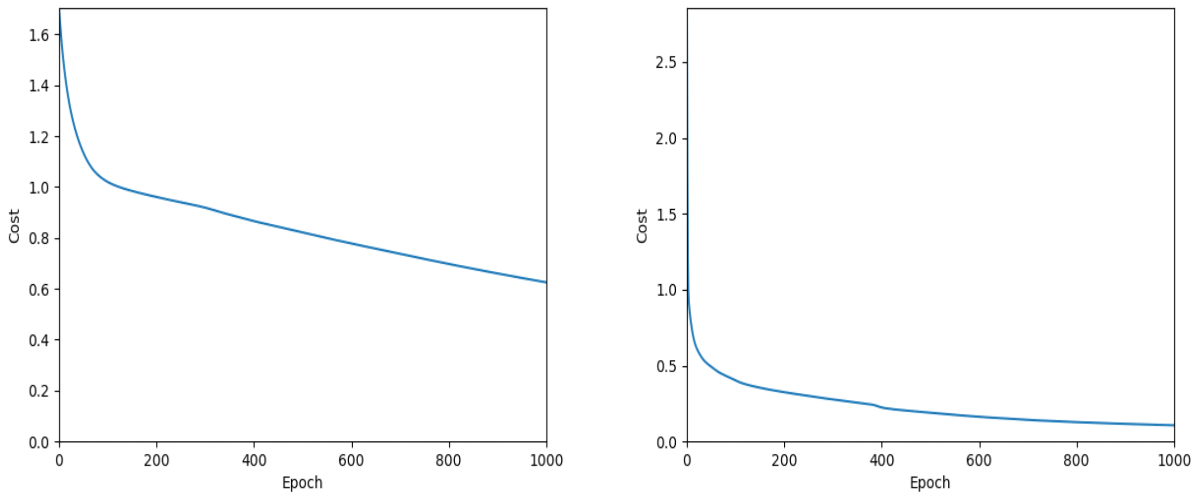

Figure 5. Compared cost(loss) of Xavier and He
(Left: Xavier, Right: He, x-axis is the epoch and y-axis is cost)

We can know He's initialization is better performance than Xaiver initialization.

Reference

: Xavier initialization

X. Glorot and Y.Bengio, "Understanding the difficulty of training deep feedforward neural networks", in International conference on artificial intelligence and statistics, 2010

: He's initialization

K. He, X. Zhang, S.Ren, and J.Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification" 2015

2-3. Optimizer method

Finally, We compare optimizer methods that are Adagradient and Adapative Mometn Estimation(Adam). Two method is popular method in Neural Network. Adagradient has advantage which is fast convergence and exact in sparse data. Also Adam has compensated for the disadvantage that the learning rate drops too quickly. In this section, I compare Adagradient to Adam through cost value and accuracy :

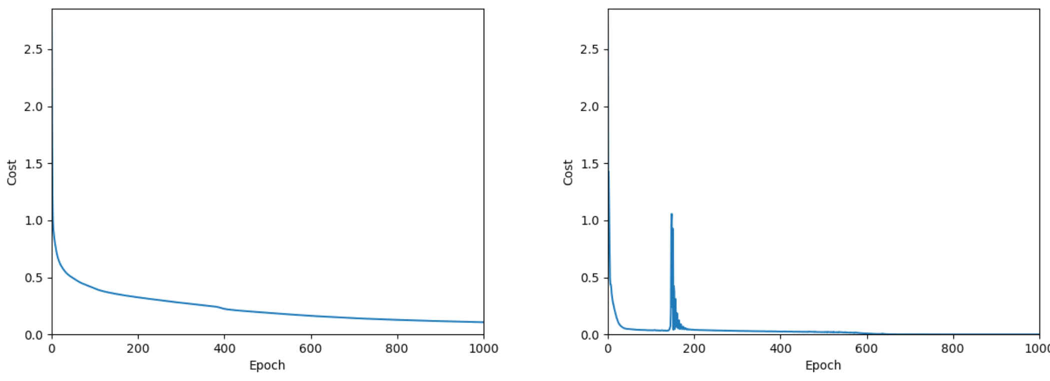

Figure 6. Compared cost(loss) of Adagrad and Adam
(Left: Adagrad, Right: Adam, x-axis is the epoch and y-axis is cost)

Also, Accuracy of Adagrad(about 98%) is better than it of Adam(about 93%). I think Adagrad is more suitable to apply to Iris data which are sparse data. And I just conjecture Adagrad is good optimizer to apply general data mining because most of the data is in the form of sparse.

Additionally, I also simulated ANN(DNN) applied “Dropout”. “Dropout” is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. (from WIKIPEDIA). In the end of this report, I make comparison between Non-Dropout ANN and Dropout ANN with applying the good methods discussed above. (Activation ReLu function, He’s Initialization, Optimizer Adagradient)

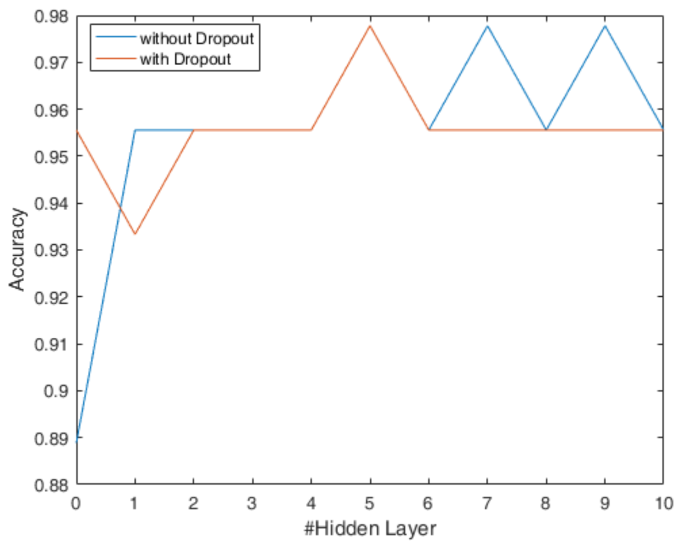

Figure 7. Compared Accuracy of Non-Dropout and Dropout
(Blue: He, Red: Xavier, x-axis is the number of hidden layer and y-axis is accuracy)

#Hidden	Accuracy	
	Non-Drop	Drop
0	0.888889	0.955556
1	0.955556	0.933333
2	0.955556	0.955556
3	0.955556	0.955556
4	0.955556	0.955556
5	0.977778	0.977778
6	0.955556	0.955556
7	0.977778	0.955556
8	0.955556	0.955556
9	0.977778	0.955556
10	0.955556	0.955556

"Dropout" is not much affect in this problem. If you deal with big data and nonlinear data, It will be a very useful way.

In this report, we can know a good method of each step to analyze Iris data. However, I do not argue It is equally applied to other problems. I think we have to find a way to optimize the problem each time in the field of Neural Network.

[Appendix] Python Code

Step1. Load Library

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Step2. Define initial weight function

```
# Create Initial variable for weight value.
def init_variable(shape):
    return(tf.Variable(tf.truncated_normal(shape = shape)))
```

Step3. Fix random seed to get stable result

```
# Fix random variable
seed = 0
tf.set_random_seed(seed)
np.random.seed(seed)
```

Step4. Import data using pandas(temporary variable DF)

```
# Import data
DF = pd.read_csv(
filepath_or_buffer='https://archive.ics.uci.edu/ml/machine-learning-databas
es/iris/iris.data',header=None, sep=',')
```

Step5. Rearrange data to input ANN machine

Raw data has ’String’ of each classes. To use data, I transform ’String’ to vector. (ex. ’Iris-setosa' -> [1, 0, 0]). And Data is shuffled using np.random.shuffle(df) in the views of non-bias.

```
Features = DF.ix[:, 0:3].values
ClassStr = DF.ix[:, 4].values
df = np.zeros((len(ClassStr), 7))
for i in range(len(ClassStr)):              # Transform String to Number.
    df[i, 0:4] = Features[i, :]
    if ClassStr[i] == 'Iris-setosa':
        df[i, 4] = 1                         # [1, 0, 0]
    elif ClassStr[i] == 'Iris-versicolor':
        df[i, 5] = 1                         # [0, 1, 0]
    elif ClassStr[i] == 'Iris-virginica':
        df[i, 6] = 1                         # [0, 0, 1]
np.random.shuffle(df)                         # Shuffle row of matrix for test.
```

Step6. Draw Figure (in the report, page2 figure1)

```
def Problem_Plot():
    DF.columns = ['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal
Width', 'Class']
    ProblemFig = plt.figure()
    plt.subplot(2, 3, 1)
    for key, val in DF.groupby('Class'):
        plt.plot(val['Sepal Length'], val['Sepal Width'], 'o', label=key)
    plt.xlabel('Sepal Length')
    plt.ylabel('Sepal Width')

    plt.subplot(2, 3, 2)
    for key,val in DF.groupby('Class'):
        plt.plot(val['Petal Length'], val['Petal Width'], 'o', label=key)
    plt.xlabel('Petal Length')
    plt.ylabel('Petal Width')
    plt.subplot(2, 3, 3)
    for key,val in DF.groupby('Class'):
        plt.plot(val['Sepal Length'], val['Petal Width'], 'o', label=key)
    plt.xlabel('Sepal Length')
    plt.ylabel('Petal Width')

    plt.subplot(2, 3, 4)
    for key,val in DF.groupby('Class'):
        plt.plot(val['Sepal Length'], val['Petal Length'], 'o', label=key)
    plt.xlabel('Sepal Length')
    plt.ylabel('Petal Length')

    plt.subplot(2, 3, 5)
    for key,val in DF.groupby('Class'):
        plt.plot(val['Sepal Width'], val['Petal Width'], 'o', label=key)
    plt.xlabel('Sepal Width')
    plt.ylabel('Petal Width')

    plt.subplot(2, 3, 6)
    for key,val in DF.groupby('Class'):
        plt.plot(val['Sepal Width'], val['Petal Length'], 'o', label=key)
    plt.xlabel('Sepal Width')
    plt.ylabel('Petal Length')
    plt.legend(loc='best', prop={'size':6})
    plt.tight_layout()
    ProblemFig.savefig('Result_IRIS_Problem.png',dpi=100)
    return
Problem_Plot()
```

Step7. Seperate Training Set and Test Set.

```
# Training Set 70% / Test Set 30%
# 150 * 0.7 = 105 / 150 * 0.3 = 45
Training_Features = df[0:105, 0:4]  # Training Set(Features).
Training_Classes = df[0:105, 4:7]   # Training Set(Classes).
Test_Features = df[105:, 0:4]       # Test Set(Features).
Test_Classes = df[105:, 4:7]        # Test Set(Classes).
```

Step8. Set parameters and variables

```
# Control panel
LEARNING_RATE = 0.01
TRAINING_EPOCHS = 1000

# Placeholder variables
x = tf.placeholder(tf.float32)  # Input variable of Features
y = tf.placeholder(tf.float32)  # Input variable of Class

NF = 4                          # the Number of Features.
NC = 3                          # the Number of Class.
NN = 100                        # the Number of Neuron(Node).
NUM_LAYER = 5
```

Step9. Set Input Layer

```
# First Layer
#w0 = init_variable(shape=[NF, NN])/np.sqrt(NF)      # Xavier's Initialization
w0 = init_variable(shape=[NF, NN])/np.sqrt(NF/2)    # He's Initialization
b0 = init_variable(shape=[NN])
layer = tf.nn.relu(tf.matmul(x, w0) + b0)
# layer = tf.nn.sigmoid(tf.matmul(x, w0) + b0)
```

Step10. Set Hidden Layer

```
for Iter in range(NUM_LAYER):
    #w = init_variable(shape=[NN, NN])/np.sqrt(NN)   # Xavier's Initialization
    w = init_variable(shape=[NN, NN])/np.sqrt(NN/2) # He's Initialization
    b = init_variable(shape=[NN])
    layer = tf.nn.relu(tf.matmul(layer, w) + b)
    # layer = tf.nn.sigmoid(tf.matmul(layer, w) + b)
```

Step11. Set Output Layer

```
# Final Layer
#w = init_variable(shape=[NN, NC])/np.sqrt(NN)       # Xavier's Initialization
w = init_variable(shape=[NN, NC])/np.sqrt(NN/2)      # He's Initialization
b = init_variable(shape=[NC])
score = tf.matmul(layer, w) + b
```

Step12. Set Run Optimization method for train

```
# Optimization method
cost    =    tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,
logits=score))
train = tf.train.AdagradOptimizer(LEARNING_RATE).minimize(cost)
#train = tf.train.AdamOptimizer(LEARNING_RATE).minimize(cost)
hypothesis = tf.nn.softmax(score)
```

Step13. Calculate accuracy

```
# Performance measures
prediction = tf.argmax(score, 1)
correct_prediction = tf.equal(prediction, tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Step14. Run session of TensorFlow

#Plot part is Figure 3,5,6 in this report

```
# Create TensorFlow session
with tf.Session() as sess:
    # Initialize variables
    tf.global_variables_initializer().run()

    # train
    cost_history = []
    for epoch in range(TRAINING_EPOCHS):
        _, cost_this_batch, hypo = sess.run([train, cost, hypothesis],
            feed_dict={x: Training_Features, y: Training_Classes})
        cost_history = np.append(cost_history, cost_this_batch)

    # Plot
    CostFig = plt.figure()
    plt.plot(range(len(cost_history)), cost_history)
    plt.xlabel('Epoch')
    plt.ylabel('Cost')
    plt.axis([0, TRAINING_EPOCHS, 0, np.max(cost_history)])
    CostFig.savefig('Result_Cost.png', dpi=100)

    print("Optimization Finished!")
    print("Accuracy: ", sess.run(accuracy, feed_dict={x: Test_Features, y: Test_Classes}))
```