

HW 3-1

Loading the dataset

Load the digits dataset from scikit learn.

In [1]:

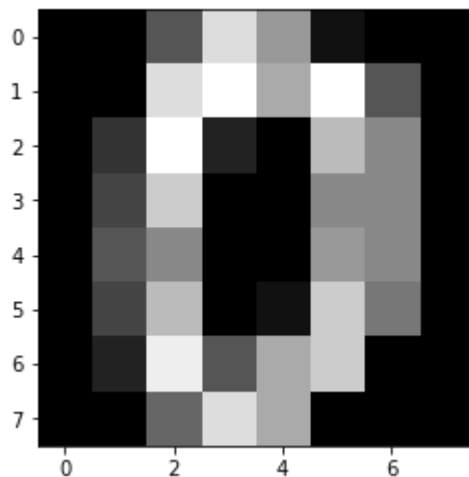
```
from sklearn.datasets import load_digits
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
digits= load_digits()
```

Understanding data

To check how data looks like.

In [2]:

```
plt.imshow(np.reshape(digits.data[0], (8,8)), cmap=plt.cm.gray)
plt.show()
```



Splitting data

Split data into 20% validation data and 80% training data

In [3]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.2)
```

Training logistic regression model

To train a logistic regression model. Use it to predict test set and listing accuracy score for both.

In [4]:

```
from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression()
logisticRegr.fit(x_train, y_train)
predictions = logisticRegr.predict(x_test)
score = logisticRegr.score(x_train, y_train)
print("Training accuracy :",score*100,"%")
score = logisticRegr.score(x_test, y_test)
print("Testing accuracy :",score*100,"%")
```

Training accuracy : 99.5824634656 %

Testing accuracy : 95.0 %

Confusion Matrix

Printing confusion matrix

In [5]:

```
import seaborn as sns
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, predictions)
print(cm)
```

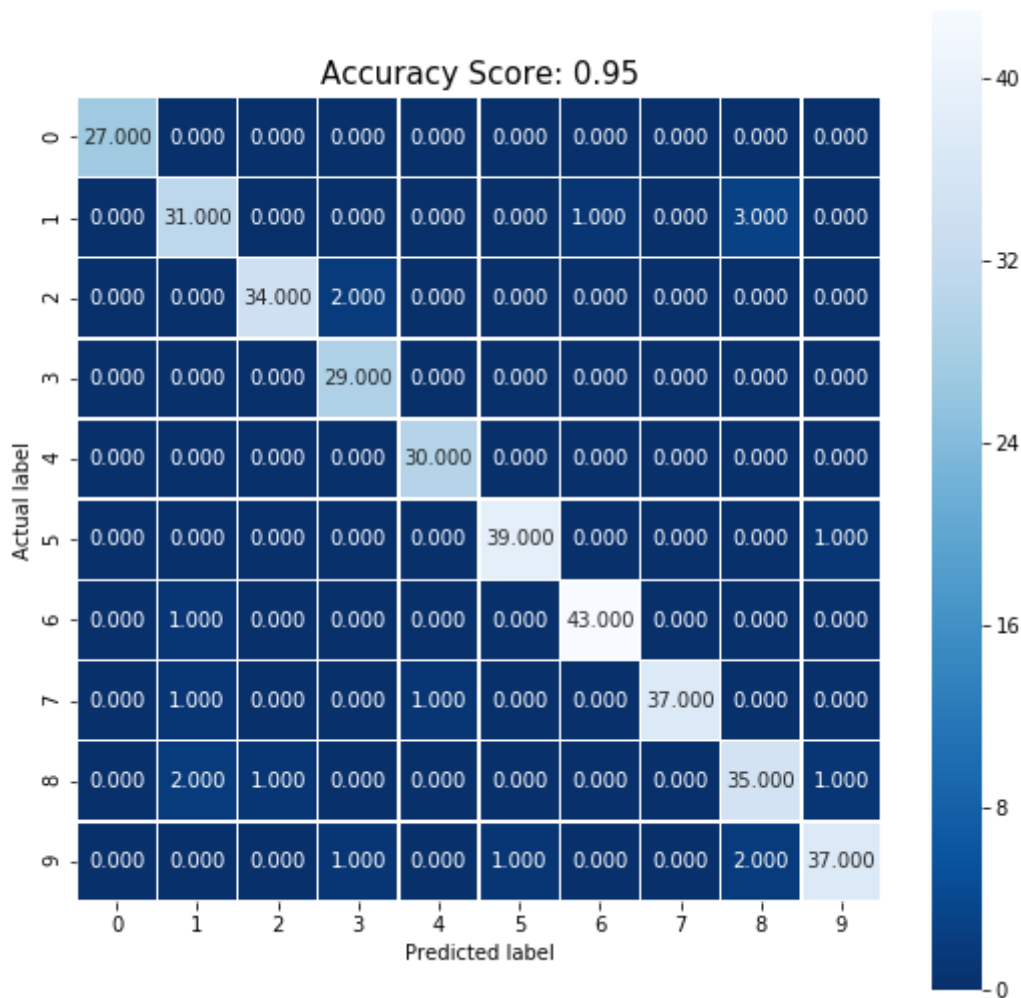
```
[[27  0  0  0  0  0  0  0  0  0]
 [ 0 31  0  0  0  0  1  0  3  0]
 [ 0  0 34  2  0  0  0  0  0  0]
 [ 0  0  0 29  0  0  0  0  0  0]
 [ 0  0  0  0 30  0  0  0  0  0]
 [ 0  0  0  0  0 39  0  0  0  1]
 [ 0  1  0  0  0  0 43  0  0  0]
 [ 0  1  0  0  1  0  0 37  0  0]
 [ 0  2  1  0  0  0  0  0 35  1]
 [ 0  0  0  1  0  1  0  0  2 37]]
```

Graphical Confusion Matrix

Better looking confusion matrix

In [6]:

```
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
plt.show()
```



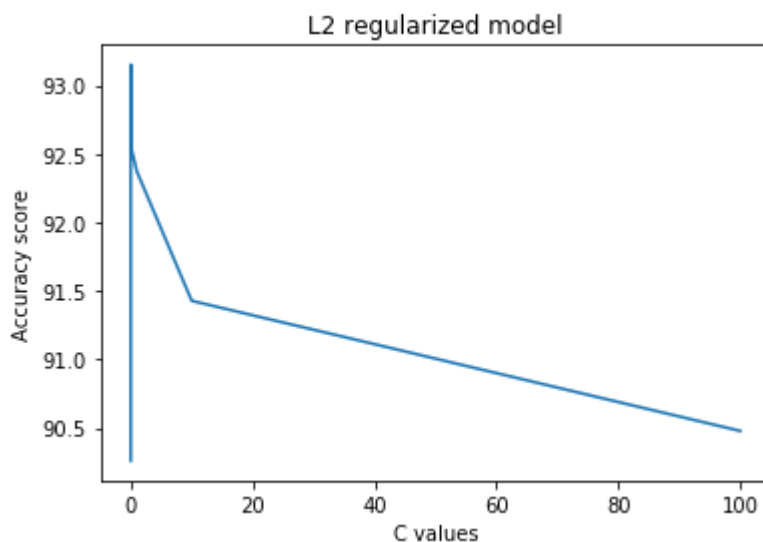
Training L2 regularized model and tuning hyper parameter "C" using K-fold Cross validation

Setting C value - [100.0, 10.0, 1.0, 0.1, 0.01, 0.001, 0.0001] and using L2 regularized model and K fold cross validation technique having 7 splits.

In [7]:

```
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=7)
skf.get_n_splits(digits.data, digits.target)
arr = []
brr = []
for i in range (0,7):
    arr.append((100/10**i)) #Storing value of C passed
    sco = 0;
    for train_index, test_index in skf.split(digits.data, digits.target):
        x_train, x_test = digits.data[train_index], digits.data[test_index]
        y_train, y_test = digits.target[train_index], digits.target[test_index]
        logisticRegr = LogisticRegression(C=(100/10**i),penalty="l2") #L2 regularized model
        logisticRegr.fit(x_train, y_train)
        predictions = logisticRegr.predict(x_test)
        score = logisticRegr.score(x_test, y_test)
        sco = sco + score
    brr.append(sco/7*100) # Taking average of all cross validation scores
    print("When C = ",(100/10**i),"accuracy reported = ", (sco/7*100))
plt.plot(arr, brr)
plt.xlabel('C values')
plt.ylabel('Accuracy score')
plt.title('L2 regularized model')
plt.show()
```

When C = 100.0 accuracy reported = 90.4782306988
 When C = 10.0 accuracy reported = 91.4276214486
 When C = 1.0 accuracy reported = 92.3716288337
 When C = 0.1 accuracy reported = 92.5381249959
 When C = 0.01 accuracy reported = 93.1507706674
 When C = 0.001 accuracy reported = 92.4793560721
 When C = 0.0001 accuracy reported = 90.2615882301



Training L1 regularized model and tuning hyper parameter "C" using K-fold Cross validation

Setting C value - [100.0, 10.0, 1.0, 0.1, 0.01, 0.001, 0.0001] and using L1 regularized model and K fold cross validation technique having 7 splits.

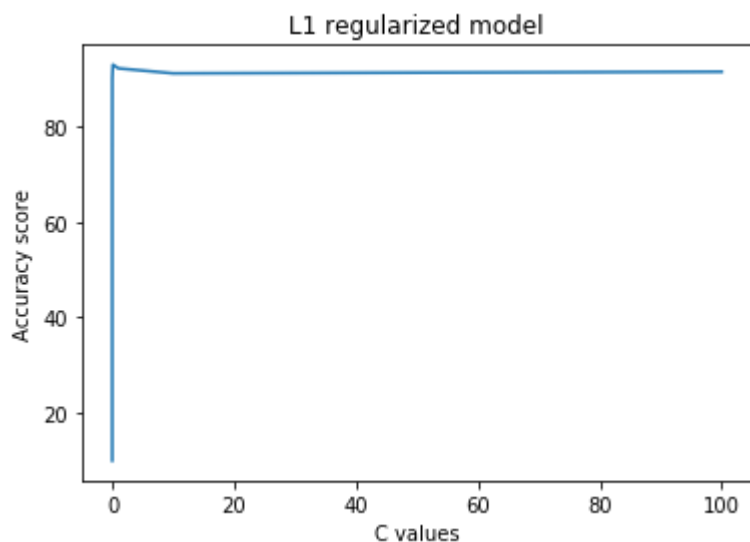
In [8]:

```

skf = StratifiedKFold(n_splits=7)
skf.get_n_splits(digits.data, digits.target)
arr = []
brr = []
for i in range (0,7):
    arr.append((100/10**i))#Storing value of C passed
    sco = 0;
    for train_index, test_index in skf.split(digits.data, digits.target):
        x_train, x_test = digits.data[train_index], digits.data[test_index]
        y_train, y_test = digits.target[train_index], digits.target[test_index]
        logisticRegr = LogisticRegression(C=(100/10**i),penalty="l1") #L1 regularized model
        logisticRegr.fit(x_train, y_train)
        predictions = logisticRegr.predict(x_test)
        score = logisticRegr.score(x_test, y_test)
        sco = sco + score
    brr.append(sco/7*100)# Taking average of all cross validation scores
    print("When C = ",(100/10**i),"accuracy reported = ", (sco/7*100))
plt.plot(arr, brr)
plt.xlabel('C values')
plt.ylabel('Accuracy score')
plt.title('L1 regularized model')
plt.show()

```

When C = 100.0 accuracy reported = 91.5942588543
 When C = 10.0 accuracy reported = 91.2611659685
 When C = 1.0 accuracy reported = 92.3132241029
 When C = 0.1 accuracy reported = 93.0978497258
 When C = 0.01 accuracy reported = 90.252266317
 When C = 0.001 accuracy reported = 76.0734990316
 When C = 0.0001 accuracy reported = 9.90484819527



HW 3-2 (a)

Import tensorflow and reset graph

In [9]:

```
import tensorflow as tf
tf.reset_default_graph()
```

Functions from Alex's notebook to run tensorboard

In [10]:

```
from IPython.display import clear_output, Image, display, HTML

def strip_consts(graph_def, max_const_size=32):
    """Strip large constant values from graph_def."""
    strip_def = tf.GraphDef()
    for n0 in graph_def.node:
        n = strip_def.node.add()
        n.MergeFrom(n0)
        if n.op == 'Const':
            tensor = n.attr['value'].tensor
            size = len(tensor.tensor_content)
            if size > max_const_size:
                tensor.tensor_content = "<stripped %d bytes>"%size
    return strip_def

def show_graph(graph_def, max_const_size=32):
    """Visualize TensorFlow graph."""
    if hasattr(graph_def, 'as_graph_def'):
        graph_def = graph_def.as_graph_def()
    strip_def = strip_consts(graph_def, max_const_size=max_const_size)
    code = """
    <script src="//cdnjs.cloudflare.com/ajax/libs/polymer/0.3.3/platform.js"></script>
    <script>
      function load() {{
        document.getElementById("{id}").pbtxt = {data};
      }}
    </script>
    <link rel="import" href="https://tensorboard.appspot.com/tf-graph-basic.build.html"
    <div style="height:600px">
      <tf-graph-basic id="{id}"></tf-graph-basic>
    </div>
    """.format(data=repr(str(strip_def)), id='graph'+str(np.random.rand()))

    iframe = """
    <iframe seamless style="width:1200px;height:620px;border:0" srcdoc="{}"></iframe>
    """.format(code.replace("'", '"'))
    display(HTML(iframe))
```

Creating computational graph in tensorflow

As image is 8x8 = 64 pixel so have 64 input neurons(placeholders)

Weight = 64x10 weights

Bias = 10

Output layer y => Equation for logistic regression

y_ => Placeholder to input true labels of images

In [11]:

```
x = tf.placeholder(tf.float32, [None, 64])
W = tf.Variable(tf.zeros([64, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b
y_ = tf.placeholder(tf.int64, [None])
cross_entropy = tf.losses.sparse_softmax_cross_entropy(labels=y_, logits=y)
show_graph(tf.get_default_graph())
```



Fit to screen

Run

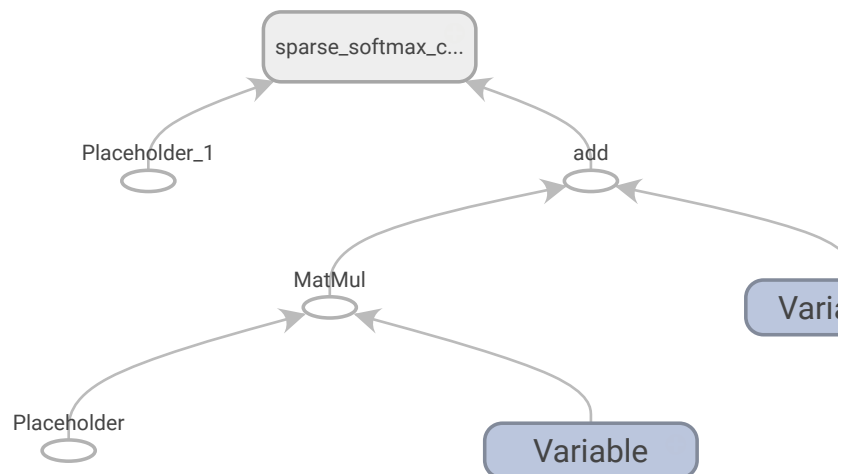
Upload

Choose File

Color

Structure

color: same substructure
gray: unique substructure



Training and testing

Learning rate - 0.001

epochs - 200

batch size - 100

x_arr => stores all values of epochs

y_arr => stores accuracy result per epoch
data

In [12]:

```

train_step = tf.train.GradientDescentOptimizer(0.001).minimize(cross_entropy)
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()
correct_prediction = tf.equal(tf.argmax(y, 1), y_)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
#training...
x_arr=[]
y_arr=[]
y_arr1=[]
epoch_size = 200
for epoch in range(0,epoch_size):    #epoch
    x_arr.append(epoch)
    batch_total = int(len(x_train)/100)
    for batch_number in range(0,batch_total+1):    #iterations
        batch_xs, batch_ys = x_train[0+(100*batch_number):100+(100*batch_number)],y_train[0+(100*batch_number):100+(100*batch_number)]
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    y_arr.append((sess.run(accuracy, feed_dict={x: x_train, y_: y_train}))*100)
    y_arr1.append((sess.run(accuracy, feed_dict={x: x_test, y_: y_test}))*100)

```

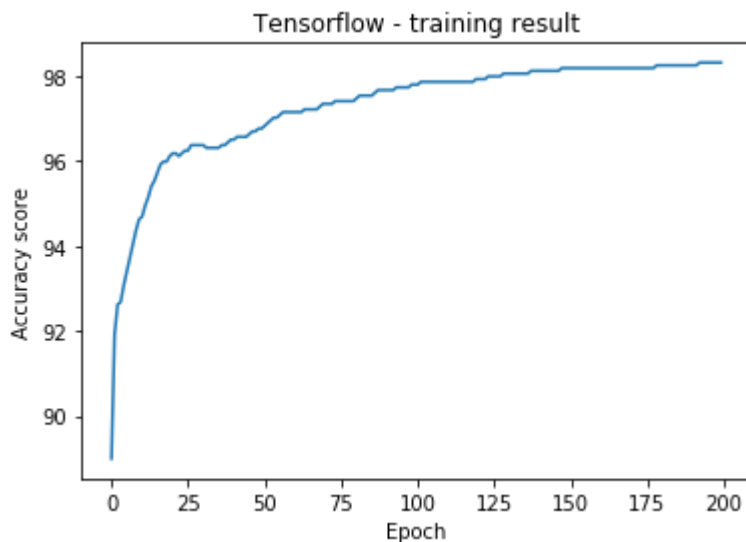
Plot for Epoch V/s Accuracy score for training data

In [13]:

```

plt.plot(x_arr,y_arr)
plt.xlabel('Epoch')
plt.ylabel('Accuracy score')
plt.title('Tensorflow - training result')
plt.show()

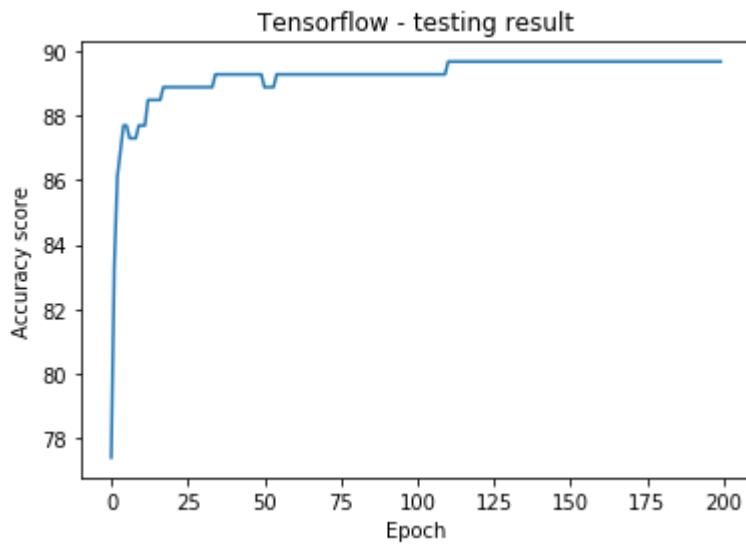
```



Plot for Epoch V/s Accuracy score for testing data

In [14]:

```
plt.plot(x_arr,y_arr1)
plt.xlabel('Epoch')
plt.ylabel('Accuracy score')
plt.title('Tensorflow - testing result')
plt.show()
```



Tensorboard

In [15]:

```
show_graph(tf.get_default_graph())
```

HW 3-2 (b)

Reset graph

In [16]:

```
tf.reset_default_graph()
```

Creating deep neural net

In [17]:

```
# Define hyperparameters and input size
n_inputs = 8*8
n_hidden1 = 300
n_hidden2 = 200
n_hidden3 = 100
n_outputs = 10
```

In [18]:

```
# Placeholders for data (inputs and targets)
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int64, shape=(None), name="y")
```

In [19]:

```
# Define neuron layers (ReLU in hidden layers)
# We'll take care of Softmax for output with Loss function

def neuron_layer(X, n_neurons, name, activation=None):
    # X input to neuron
    # number of neurons for the layer
    # name of layer
    # pass in eventual activation function

    with tf.name_scope(name):
        n_inputs = int(X.get_shape()[1])

        # initialize weights to prevent vanishing / exploding gradients
        stddev = 2 / np.sqrt(n_inputs)
        init = tf.truncated_normal((n_inputs, n_neurons), stddev=stddev)

        # Initialize weights for the layer
        W = tf.Variable(init, name="weights")
        # biases
        b = tf.Variable(tf.zeros([n_neurons]), name="bias")
        # Output from every neuron
        Z = tf.matmul(X, W) + b

        if activation is not None:
            z = activation(Z)
            drop_out = tf.nn.dropout(z, keep_prob = 0.90)
            return drop_out
        else:
            return Z
```

In [20]:

```
# Define the hidden layers
with tf.name_scope("dnn"):
    hidden1 = neuron_layer(X, n_hidden1, name="hidden1", activation=tf.nn.relu)
    hidden2 = neuron_layer(hidden1, n_hidden2, name="hidden2", activation=tf.nn.relu)
    hidden3 = neuron_layer(hidden2, n_hidden3, name="hidden3", activation=tf.nn.relu)
    logits = neuron_layer(hidden3, n_outputs, name="outputs")
```

In [21]:

```
# Define loss function (that also optimizes Softmax for output):
with tf.name_scope("loss"):
    # logits are from the last output of the dnn
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
```

In [22]:

```
# Training step with Gradient Descent
learning_rate = 0.001
with tf.name_scope("train"):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    training_op = optimizer.minimize(loss)
```

In [23]:


```
# Evaluation to see accuracy


with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
```

Show Graph

In [24]:


```
show_graph(tf.get_default_graph())
```

 Fit to screen

Run 


Upload


Choose File


Color Structure 


color: same substructure
gray: unique substructure


Graph (* = expandable)


 Namespace*


 OpNode


 Unconnected series*


 Connected series*

 Constant

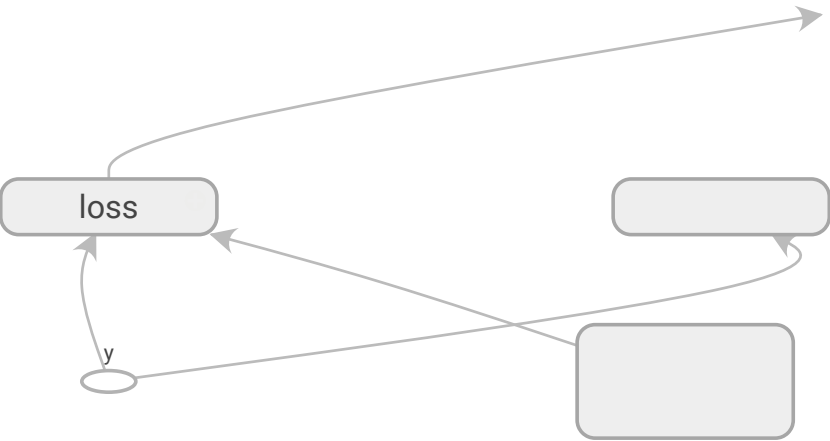
 Summary

 Dataflow edge

 Control dependency edge

 Reference edge

Main Graph



EVALUATION PHASE

Train steps

In [25]:

```

init = tf.global_variables_initializer()
saver = tf.train.Saver()
x_arr=[]
y_arr=[]
y_arr1=[]
n_epochs = 1000
batch_total = int(len(x_train)/100)
with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        x_arr.append(epoch)
        for batch_number in range(0,batch_total+1): #iterations
            X_batch, y_batch = x_train[0+(100*batch_number):100+(100*batch_number)],y_train[0+(100*batch_number):100+(100*batch_number)]
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
            acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
            acc_val = accuracy.eval(feed_dict={X: x_test, y: y_test})
            y_arr.append(acc_train*100)
            y_arr1.append(acc_val*100)

```

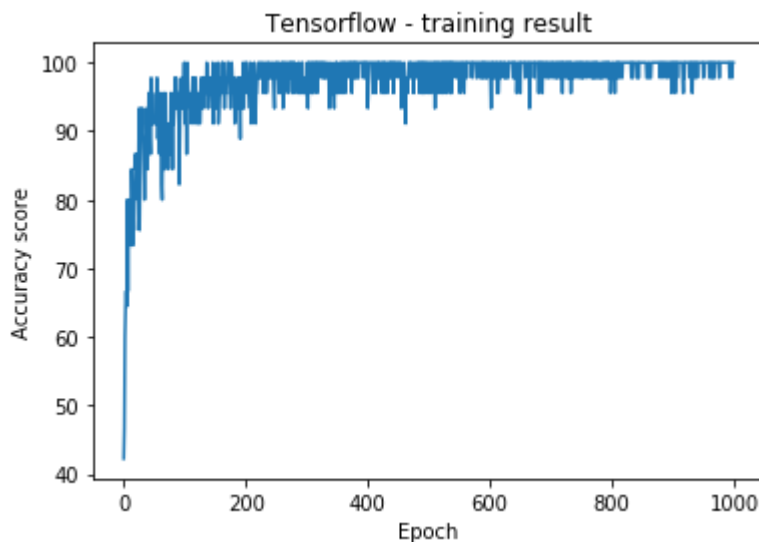
Plot for Epoch V/s Accuracy score for training data

In [26]:

```

plt.plot(x_arr,y_arr)
plt.xlabel('Epoch')
plt.ylabel('Accuracy score')
plt.title('Tensorflow - training result')
plt.show()

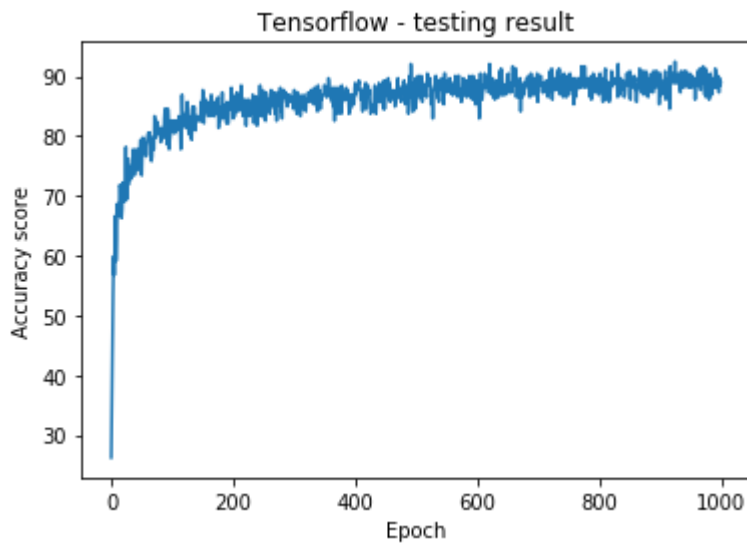
```



Plot for Epoch V/s Accuracy score for testing data

In [27]:

```
plt.plot(x_arr,y_arr1)
plt.xlabel('Epoch')
plt.ylabel('Accuracy score')
plt.title('Tensorflow - testing result')
plt.show()
```



Show Graph

In [29]:

```
show_graph(tf.get_default_graph())
```

Difference in model complexity between the Dense Neural Network model and the multiclass logistic regression model in terms of number of (trainable) model parameters -- i.e., weights + biases

In case of multiclass logistic regression model we had total **10** biases and **640**($64*10$) weights to train.

In case of Dense Neural Network Model we had total **610**($300+200+100+10$) biases and **81000**($(300*200)+(200*100)+(100*10)$) weights to train.

In []: