Name: Hong Hai Tran
Student ID: 20174583

EE488 – Deep Learning with Alpha Go
Project #1

Task 1. CLASSIFICATION

- Source code (TensorFlow v1.3.0, NumPy v1.13.3, MatPlotLib v1.5.1)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from math import pi

# Neural network settings
hidNeus = 20
epochNum = 1000
lr = 0.0001

# Generating training set
trainingSetSize = 10000
x_train = np.zeros((trainingSetSize, 2), dtype = np.float32)
y_train = np.zeros((trainingSetSize, 1), dtype = np.float32)
r_train = np.random.normal(size = trainingSetSize)
t_train = np.random.uniform(0, 2*pi, trainingSetSize)
for i in range(trainingSetSize):
    y_train[i] = np.random.randint(2)   # 0 or 1
    if ( y_train[i] == 0 ):  # x = [r*cos(t), r*sin(t)]
        x_train[i,:] = [r_train[i] * np.cos(t_train[i]), r_train[i] * np.sin(t_train[i])]

    else:   # x = [(r+5)*cos(t), (r+5)*cos(t)]
        x_train[i,:] = [(r_train[i] + 5) * np.cos(t_train[i]), (r_train[i] + 5) * np.sin(t
_train[i])]

# Generating validation set
validSetSize = 1000
x_valid = np.zeros((validSetSize, 2), dtype = np.float32)
y_valid = np.zeros((validSetSize, 1), dtype = np.float32)
r_valid = np.random.normal(size = validSetSize)
t_valid = np.random.uniform(0, 2*pi, validSetSize)
for i in range(validSetSize):
    y_valid[i] = np.random.randint(2)   # 0 or 1
    if ( y_valid[i] == 0 ):  # x = [r*cos(t), r*sin(t)]
        x_valid[i,:] = [r_valid[i] * np.cos(t_valid[i]), r_valid[i] * np.sin(t_valid[i])]

    else:   # x = [(r+5)*cos(t), (r+5)*cos(t)]
        x_valid[i,:] = [(r_valid[i] + 5) * np.cos(t_valid[i]), (r_valid[i] + 5) * np.sin(t
_valid[i])]

# Generating test set
testSetSize = 1000
x_test = np.zeros((testSetSize, 2), dtype = np.float32)
y_test = np.zeros((testSetSize, 1), dtype = np.float32)
r_test = np.random.normal(size = testSetSize)
t_test = np.random.uniform(0, 2*pi, testSetSize)
for i in range(testSetSize):
    y_test[i] = np.random.randint(2)   # 0 or 1
    if ( y_test[i] == 0 ):  # x = [r*cos(t), r*sin(t)]
        x_test[i,:] = [r_test[i] * np.cos(t_test[i]), r_test[i] * np.sin(t_test[i])]
```

```
    else:   # x = [(r+5)*cos(t), (r+5)*cos(t)]
        x_test[i,:] = [(r_test[i] + 5) * np.cos(t_test[i]), (r_test[i] + 5) * np.sin(t_tes
t[i])]

# Places to hold data
x_ = tf.placeholder(dtype = tf.float32, shape = [None, 2])
y_ = tf.placeholder(dtype = tf.float32, shape = [None, 1])

# Params initialization
W = tf.Variable(tf.truncated_normal([2, hidNeus], stddev = 0.1))
c = tf.Variable(tf.constant(0, shape = [hidNeus], dtype = tf.float32))
w = tf.Variable(tf.truncated_normal([hidNeus, 1], stddev = 0.1))
b = tf.Variable(tf.constant(0, shape = [1], dtype = tf.float32))

# Hidden layer implementation
h_relu = tf.nn.relu(tf.matmul(x_, W) + c)

# Output layer implementation
z = tf.matmul(h_relu, w) + b
y_hat = tf.nn.sigmoid(z)

# Cost function
cost = tf.nn.sigmoid_cross_entropy_with_logits(labels = y_, logits = z)

# Training settings
optimizer = tf.train.GradientDescentOptimizer(lr)
train = optimizer.minimize(cost)

# Evaluation of the model
y_pred = tf.round(y_hat)
correct_pred = tf.equal(y_pred, y_)
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Learning
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

miniBatchSize = 1000
miniBatchNum = trainingSetSize/miniBatchSize
print "========================="
print "|Epoch\t|Train\t|Val\t|"
print "|=======================|"
for i in range(epochNum):
    # Extract mini batch data and start training
    for j in range(miniBatchNum):
        trainingData = x_train[(miniBatchSize*j) : (miniBatchSize*(j+1)), :]
        trainingLabels = y_train[(miniBatchSize*j) : (miniBatchSize*(j+1)), :]
        train.run(feed_dict={x_: trainingData, y_: trainingLabels})
    # Print train and validation accuracy during training
    if i%50 == 49:
        trainAccuracy = accuracy.eval(feed_dict = {x_:trainingData, y_: trainingLabels})
        validAccuracy = accuracy.eval(feed_dict = {x_: x_valid, y_: y_valid})
        print "|%d\t|%.4f\t|%.4f\t|" %(i+1, trainAccuracy, validAccuracy)
print '|=======================|'

testAccuracy = accuracy.eval(feed_dict = {x_: x_test, y_: y_test})
print "Test accuracy is %.4f" %(testAccuracy)

print 'W =', W.eval()
print 'c =', c.eval()
print 'w =', w.eval()
```

```
print 'b =', b.eval()

plt.figure(1)
# Plot test data
for i in range(testSetSize):
    if ( y_test[i] == 0 ):
        plt.scatter(x_test[i,0], x_test[i,1], s = 50, c = 'b', marker = 'o')
    else:
        plt.scatter(x_test[i,0], x_test[i,1], s = 50, c = 'g', marker = 'v')

W_val = W.eval()
c_val = c.eval()

# Plot line dividing active and inactive region for hidden ReLU layer
for i in range(hidNeus):
    a = W_val[0,i]; b = W_val[1,i]; c = c_val[i]
    x_plot = np.arange(-10, 11)
    plt.plot(x_plot, (-a*x_plot-c)/b, linewidth = 1, color = 'r')

xmin=-10.
xmax= 10.
ymin=-10.
ymax= 10.
plt.axis([xmin,xmax,ymin,ymax])
plt.grid(True)
plt.show()
```

- Plot

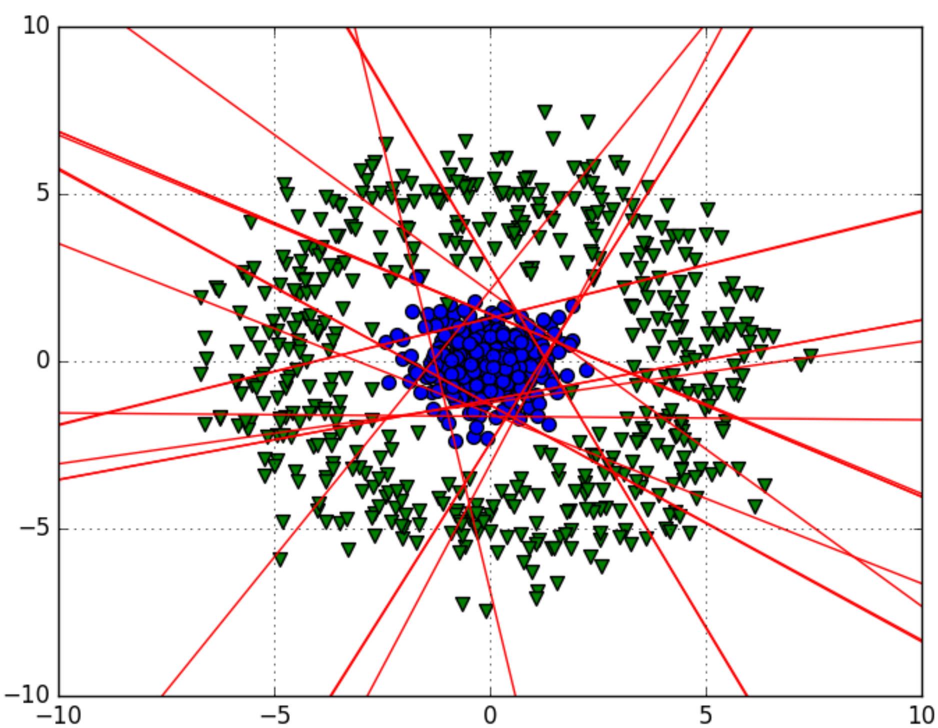

Figure 1. Test data and diving lines

- Output of the code
  - Training and validation accuracies during training and final test accuracy

```
========================
|Epoch  |Train  |Val    |
|======================|
|10     |0.9900 |0.9870 |
|20     |0.9890 |0.9870 |
|30     |0.9890 |0.9860 |
|40     |0.9890 |0.9860 |
|50     |0.9890 |0.9860 |
|60     |0.9900 |0.9860 |
|70     |0.9900 |0.9860 |
|80     |0.9900 |0.9860 |
|90     |0.9900 |0.9860 |
|100    |0.9910 |0.9860 |
|======================|
Test accuracy is 0.9970
```

  - Trained parameters

```
W = [[ 0.7235046   0.39629221  0.36658379 -1.68910551  1.33046544  1.07475519
   1.28344595 -1.65421963 -0.64146078  0.62569875  0.09673909 -0.20202288
   0.42336509  0.36339635  0.1036927  -0.00341243  0.47704419 -0.04880916
  -0.4683789  -0.79431009]
 [ 0.76992285 -1.65734363  0.67143053 -0.31463367 -0.6486817  -0.52609867
   0.59336025  1.036461   -1.26214266  1.14695108 -0.52910078  0.63645577
   0.19683278  0.67775023 -0.43500555 -0.33433065 -0.1961306  -0.0689773
   1.46089911 -1.13063121]]
c = [-1.59895909 -1.91320348 -0.94656873 -2.14913797 -1.59920752 -1.28871453
 -1.69258273 -2.19574785 -1.96769583 -1.61884499 -0.65564042 -0.81731778
 -0.55672014 -0.95140052 -0.50136495 -0.55099249 -0.60734504 -0.08997842
 -1.8839606  -1.47423291]
w = [[ 1.63439631]
 [ 1.95125616]
 [ 1.01585841]
 [ 2.26866007]
 [ 1.76528895]
 [ 1.4235121 ]
 [ 1.87913728]
 [ 2.31844783]
 [ 1.72891271]
 [ 1.73055518]
 [ 0.64795226]
 [ 0.85314786]
 [ 0.62173533]
 [ 1.01779056]
 [ 0.50995439]
 [ 0.50079238]
 [ 0.65859044]
 [ 0.09405661]
 [ 1.82854557]
 [ 1.5147388 ]]
b = [-11.2029705]
```

- Explanation on code design
  - First, I generate training, validation, test data and labels. I generate labels using `np.random.randint(2)` to have a label of value 0 or 1. Depending on the generated value of label, I choose value of the corresponding data:
    - For label ‘0’, `(x1, x2) = (r*cos(t), r*sin(t))`
    - For label ‘1’, `(x1, x2) = ((r+5)*cos(t), (r+5)*sin(t))`

`r` values are generated using `np.random.normal()` and `t` values are generated using `np.random.uniform()`. The numbers of examples for training set, validation set and test set are 10000, 1000 and 1000 respectively.
  - After that, I initialize parameters of the neural network, which are W (matrix of shape 2x20), c (row vector of length 20), w (column vector of length 20) and b (a scalar) as Variable classes.
  - Next, I start to construct structure of the neural network, which consists of an input layer of 2 neurons, a hidden layer with 20 ReLU hidden neurons, and an output layer of 1 neuron.
  - Then, I define the cost function and the optimizer to optimize the network parameters, as well as the accuracy to evaluate the model.
  - After everything has been set up, I start to train the model. I use mini-batch gradient descent to train the model, with mini batch size of 1000. The number of epoch I used is 100. Every 10 epochs, I will print the corresponding training and validation accuracy. After training, I print the test accuracy to see the result.
  - After training, I plot the test set as well as the lines dividing active and inactive regions for hidden-layer neurons. There are 20 pairs of value of W and b, so there are 20 corresponding lines of the format `ax+by+c=0`, where `a=W[0,i]`, `b = W[1,i]` and `c = c[i]`, where i is in the range [0, 20).
- Interpretation of the results
  - The test accuracy of the model is very high, at 99.7%, which means the model classified correctly 997 out of 1000 examples.
  - After training, the trained parameters have changed from their initial values, which proves that the learning process happened and made impact on the parameters.
  - Look at Figure 1, we can figure out that neural network performs classification by using the lines constructed by parameters W and b, which **divides the data labeled '0' and data labeled '1' into two different regions**. If a pair of values (x1, x2) lies on these lines, the input to the corresponding hidden unit, which is `W[0,i]*x0+W[1,i]*x1+c[i]`, will be equal to 0.

Task 2. TRANSFER LEARNING

- Source code (TensorFlow v1.3.0, NumPy v1.13.3, MatPlotLib v1.5.1)

```python
import tensorflow as tf
import numpy as np

# Get MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('./MNIST_data', one_hot=True)

# Parameters
epochNum = 20
lr = 1e-4

# Remove number '9' from training set
filterTrainImages = []
filterTrainLabels = []
for i in range(mnist.train.labels.shape[0]):
    if ( np.argmax(mnist.train.labels[i,:]) != 9 ):
        filterTrainImages.append(mnist.train.images[i,:])
        filterTrainLabels.append(mnist.train.labels[i,0:9])
filterTrainImages = np.array(filterTrainImages)
filterTrainLabels = np.array(filterTrainLabels)

# Remove number '9' from validation set
filterValidImages = []
filterValidLabels = []
for i in range(mnist.validation.labels.shape[0]):
    if ( np.argmax(mnist.validation.labels[i,:]) != 9 ):
        filterValidImages.append(mnist.validation.images[i,:])
        filterValidLabels.append(mnist.validation.labels[i,0:9])
filterValidImages = np.array(filterValidImages)
filterValidLabels = np.array(filterValidLabels)

# Remove number '9' from test set
filterTestImages = []
filterTestLabels = []
for i in range(mnist.test.labels.shape[0]):
    if ( np.argmax(mnist.test.labels[i,:]) != 9 ):
        filterTestImages.append(mnist.test.images[i,:])
        filterTestLabels.append(mnist.test.labels[i,0:9])
filterTestImages = np.array(filterTestImages)
filterTestLabels = np.array(filterTestLabels)

# Tensorflow session
sess = tf.InteractiveSession()

# Place to hold data
x_ = tf.placeholder(tf.float32, shape=[None, 784])
y_ = tf.placeholder(tf.float32, shape=[None, 9])

# First convolutional layer
x_image = tf.reshape(x_, [-1,28,28,1])
W_conv = tf.Variable(tf.truncated_normal([5, 5, 1, 30], stddev=0.1))
b_conv = tf.Variable(tf.constant(0.1, shape=[30]))
h_conv = tf.nn.conv2d(x_image, W_conv, strides=[1, 1, 1, 1], padding='VALID')
h_relu = tf.nn.relu(h_conv + b_conv)
h_pool = tf.nn.max_pool(h_relu, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')


# Second convolutional layer
```

```python
W_conv2 = tf.Variable(tf.truncated_normal([3, 3, 30, 50], stddev=0.1))
b_conv2 = tf.Variable(tf.constant(0.1, shape=[50]))
h_conv2 = tf.nn.conv2d(h_pool, W_conv2, strides=[1, 1, 1, 1], padding='VALID')
h_relu2 = tf.nn.relu(h_conv2 + b_conv2)
h_pool2 = tf.nn.max_pool(h_relu2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME'
)

# Fully-connected layer
W_fc1 = tf.Variable(tf.truncated_normal([5 * 5 * 50, 500], stddev=0.1))
b_fc1 = tf.Variable(tf.constant(0.1, shape=[500]))
h_pool_flat = tf.reshape(h_pool2, [-1, 5*5*50])
h_fc1 = tf.nn.relu(tf.matmul(h_pool_flat, W_fc1) + b_fc1)

# Dropout
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# Output layer
W_fc2 = tf.Variable(tf.truncated_normal([500, 9], stddev=0.1))
b_fc2 = tf.Variable(tf.constant(0.1, shape=[9]))
y_hat = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

# Train and Evaluate the Model
crossEntropy = - tf.reduce_sum(y_*tf.log(y_hat))
optimizer = tf.train.AdamOptimizer(lr)
training = optimizer.minimize(crossEntropy)
correctPred = tf.equal(tf.argmax(y_hat,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correctPred, tf.float32))

# Transfer Learning: Place holder for labels
y_tl_ = tf.placeholder(tf.float32, shape=[None, 10])
# Transfer Learning: New output layer
W_fc2_tl = tf.Variable(tf.truncated_normal([500, 10], stddev=0.1))
b_fc2_tl = tf.Variable(tf.constant(0.1, shape=[10]))
y_hat_tl = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2_tl) + b_fc2_tl)
# Transfer Learning: Train and Evaluate the Model with new output layer
crossEntropyTL = - tf.reduce_sum(y_tl_*tf.log(y_hat_tl))
trainingTL = optimizer.minimize(crossEntropyTL, var_list = (W_fc2_tl, b_fc2_tl))
correctPredTL = tf.equal(tf.argmax(y_hat_tl, 1), tf.argmax(y_tl_, 1))
accuracyTL = tf.reduce_mean(tf.cast(correctPredTL, tf.float32))

# Initialize all declared tensors
tf.global_variables_initializer().run()

# Training model
trainingSetSize = filterTrainImages.shape[0]
miniBatchSize = 100
miniBatchNum = int(np.ceil(trainingSetSize/miniBatchSize))
print '================================='
print '|Epoch\t|MnBatch|Train\t|Val\t|'
print '|===============================|'
for j in range(epochNum):
    for i in range(miniBatchNum):
        # Extract mini batch data
        if ( i < miniBatchNum - 1 ):
            trainingImages = filterTrainImages[(miniBatchSize*i) : (miniBatchSize*(i+1)),
:]
            trainingLabels = filterTrainLabels[(miniBatchSize*i) : (miniBatchSize*(i+1)),
:]
        else:   # i = miniBatchNum - 1
            restTrainingExps = trainingSetSize - miniBatchSize*i
```

```
            trainingImages = filterTrainImages[(miniBatchSize*i) : (miniBatchSize*i + rest
TrainingExps), :]
            trainingLabels = filterTrainLabels[(miniBatchSize*i) : (miniBatchSize*i + rest
TrainingExps), :]
        # Start training
        training.run(feed_dict={x_: trainingImages, y_: trainingLabels, keep_prob: 0.5})
        # Print train and validation accuracy during training
        if ((i%100 == 99) or (i == miniBatchNum - 1)):
            trainAccuracy = accuracy.eval(feed_dict={x_:trainingImages, y_: trainingLabels
, keep_prob: 1.})
            valAccuracy = accuracy.eval(feed_dict=\
                {x_: filterValidImages, y_: filterValidLabels, keep_prob: 1.})
            print '|%d\t|%d\t|%.4f\t|%.4f\t|' % (j+1, i+1, trainAccuracy, valAccuracy)
print '|===============================|'

# Test accuracy of pre-trained model
testAccuracy = accuracy.eval(feed_dict=\
    {x_: filterTestImages, y_: filterTestLabels, keep_prob: 1.})
print 'Test accuracy of pre-trained model (9 outputs) is %.4f' % (testAccuracy)

# Transfer Learning
print '================================='
print '|Epoch\t|MnBatch|Train\t|Val\t|'
print '|===============================|'
for j in range(epochNum):
    for i in range(550):
        # Extract mini batch data
        batch = mnist.train.next_batch(100)
        # Start training
        trainingTL.run(feed_dict={x_: batch[0], y_tl_: batch[1], keep_prob: 0.5})
        # Print training and validation accuracy during training
        if (i%100 == 99) or (i == 549):
            trainAccuracy = accuracyTL.eval(feed_dict={x_:batch[0], y_tl_: batch[1], keep_
prob: 1.})
            valAccuracy = accuracyTL.eval(feed_dict=\
                {x_: mnist.validation.images, y_tl_:mnist.validation.labels, keep_prob: 1.
})
            print '|%d\t|%d\t|%.4f\t|%.4f\t|' % (j+1, i+1, trainAccuracy, valAccuracy)
print '|===============================|'

# Test accuracy of transfer learning model
testAccuracyNew = accuracyTL.eval(feed_dict=\
    {x_: mnist.test.images, y_tl_:mnist.test.labels, keep_prob: 1.})
print 'Test accuracy of transfer learning model (10 outputs) is %.4f' % (testAccuracyNew)
```

- Output of the code
    - Pre-trained model (9 outputs)

Epoch	MnBatch	Train	Val
1	100	0.7800	0.8606
1	200	0.8600	0.9250
1	300	0.9300	0.9407
1	400	0.9800	0.9514
1	495	0.9863	0.9556
2	100	0.9300	0.9598
2	200	0.9300	0.9663
2	300	0.9900	0.9687
2	400	0.9900	0.9716
2	495	0.9932	0.9727
3	100	0.9700	0.9745
3	200	0.9200	0.9756
3	300	0.9900	0.9771
3	400	0.9900	0.9774
3	495	0.9932	0.9782
4	100	0.9800	0.9805
4	200	0.9500	0.9807
4	300	1.0000	0.9818
4	400	0.9900	0.9820
4	495	0.9932	0.9836
5	100	0.9900	0.9838
5	200	0.9600	0.9842
5	300	1.0000	0.9853
5	400	0.9900	0.9865
5	495	0.9932	0.9858
6	100	0.9900	0.9869
6	200	0.9800	0.9871
6	300	1.0000	0.9882
6	400	1.0000	0.9880
6	495	0.9932	0.9880
7	100	0.9900	0.9887
7	200	0.9700	0.9887
7	300	1.0000	0.9887
7	400	1.0000	0.9885
7	495	0.9932	0.9889
8	100	0.9900	0.9891
8	200	0.9800	0.9889
8	300	1.0000	0.9909
8	400	1.0000	0.9911
8	495	0.9932	0.9896
9	100	1.0000	0.9909
9	200	0.9900	0.9891
9	300	1.0000	0.9922
9	400	1.0000	0.9902
9	495	0.9932	0.9905
10	100	1.0000	0.9922
10	200	0.9900	0.9916
10	300	1.0000	0.9925
10	400	1.0000	0.9929
10	495	0.9932	0.9909
11	100	1.0000	0.9920
11	200	0.9900	0.9909
11	300	1.0000	0.9929
11	400	1.0000	0.9927
11	495	0.9932	0.9909
12	100	1.0000	0.9931
12	200	0.9900	0.9925
12	300	1.0000	0.9931
12	400	1.0000	0.9931
12	495	0.9932	0.9920
13	100	1.0000	0.9922
13	200	0.9900	0.9929
13	300	1.0000	0.9931
13	400	1.0000	0.9936
13	495	0.9932	0.9911
14	100	1.0000	0.9938
14	200	1.0000	0.9927
14	300	1.0000	0.9925
14	400	1.0000	0.9933
14	495	0.9932	0.9920
15	100	1.0000	0.9936
15	200	0.9900	0.9920
15	300	1.0000	0.9936
15	400	1.0000	0.9938
15	495	0.9932	0.9922
16	100	1.0000	0.9933
16	200	1.0000	0.9931
16	300	1.0000	0.9931
16	400	1.0000	0.9938
16	495	0.9932	0.9922
17	100	1.0000	0.9938
17	200	0.9900	0.9929
17	300	1.0000	0.9933
17	400	1.0000	0.9940
17	495	0.9932	0.9920
18	100	1.0000	0.9938
18	200	1.0000	0.9931
18	300	1.0000	0.9942
18	400	1.0000	0.9938
18	495	0.9932	0.9936
19	100	1.0000	0.9931
19	200	1.0000	0.9938
19	300	1.0000	0.9931
19	400	1.0000	0.9940
19	495	0.9932	0.9927
20	100	1.0000	0.9933
20	200	1.0000	0.9940
20	300	1.0000	0.9938
20	400	1.0000	0.9933
20	495	0.9932	0.9922

Test accuracy of pre-trained model is 0.9924

- Transfer learning model (10 outputs):

Epoch	MnBatch	Train	Val
1	100	0.9400	0.9374
1	200	0.9600	0.9590
1	300	0.9900	0.9648
1	400	0.9900	0.9700
1	500	0.9700	0.9724
1	550	0.9900	0.9730
2	100	0.9900	0.9738
2	200	0.9800	0.9746
2	300	1.0000	0.9752
2	400	0.9800	0.9760
2	500	0.9900	0.9764
2	550	0.9700	0.9768
3	100	0.9700	0.9772
3	200	0.9700	0.9778
3	300	0.9800	0.9792
3	400	0.9600	0.9798
3	500	0.9700	0.9796
3	550	0.9600	0.9800
4	100	0.9500	0.9802
4	200	0.9800	0.9810
4	300	0.9900	0.9812
4	400	0.9800	0.9810
4	500	0.9800	0.9814
4	550	1.0000	0.9814
5	100	1.0000	0.9822
5	200	0.9900	0.9822
5	300	0.9600	0.9828
5	400	0.9900	0.9832
5	500	0.9900	0.9828
5	550	0.9900	0.9830
6	100	0.9900	0.9834
6	200	1.0000	0.9832
6	300	0.9900	0.9834
6	400	0.9900	0.9844
6	500	0.9800	0.9850
6	550	1.0000	0.9844
7	100	0.9900	0.9850
7	200	1.0000	0.9854
7	300	0.9900	0.9856
7	400	1.0000	0.9860
7	500	0.9900	0.9854
7	550	1.0000	0.9856
8	100	0.9900	0.9854
8	200	0.9900	0.9858
8	300	0.9900	0.9864
8	400	1.0000	0.9866
8	500	0.9700	0.9868
8	550	0.9800	0.9868
9	100	0.9600	0.9866
9	200	1.0000	0.9870
9	300	1.0000	0.9872
9	400	0.9900	0.9874
9	500	1.0000	0.9876
9	550	0.9900	0.9874
10	100	0.9800	0.9874
10	200	0.9900	0.9876
10	300	1.0000	0.9876
10	400	0.9900	0.9876
10	500	0.9900	0.9876
10	550	0.9900	0.9876
11	100	0.9700	0.9878
11	200	1.0000	0.9880
11	300	0.9800	0.9882
11	400	0.9900	0.9878
11	500	0.9900	0.9888
11	550	1.0000	0.9884
12	100	0.9900	0.9880
12	200	0.9900	0.9880
12	300	0.9900	0.9884
12	400	0.9900	0.9884
12	500	0.9800	0.9880
12	550	1.0000	0.9886
13	100	0.9800	0.9886
13	200	1.0000	0.9882
13	300	1.0000	0.9886
13	400	1.0000	0.9888
13	500	1.0000	0.9882
13	550	0.9700	0.9884
14	100	0.9900	0.9884
14	200	0.9800	0.9892
14	300	1.0000	0.9892
14	400	0.9900	0.9896
14	500	1.0000	0.9896
14	550	0.9800	0.9896
15	100	1.0000	0.9894
15	200	1.0000	0.9894
15	300	0.9900	0.9892
15	400	1.0000	0.9892
15	500	1.0000	0.9896
15	550	0.9800	0.9900
16	100	0.9900	0.9890
16	200	0.9900	0.9898
16	300	1.0000	0.9896
16	400	0.9900	0.9904
16	500	0.9700	0.9902
16	550	0.9700	0.9898
17	100	1.0000	0.9890
17	200	0.9800	0.9892
17	300	1.0000	0.9900
17	400	0.9900	0.9896
17	500	0.9900	0.9896
17	550	1.0000	0.9898
18	100	0.9800	0.9896
18	200	1.0000	0.9896
18	300	1.0000	0.9900
18	400	1.0000	0.9896
18	500	0.9800	0.9900
18	550	0.9900	0.9902
19	100	1.0000	0.9900
19	200	1.0000	0.9900
19	300	0.9900	0.9896
19	400	1.0000	0.9900
19	500	1.0000	0.9904
19	550	0.9800	0.9902
20	100	0.9900	0.9906
20	200	0.9900	0.9904
20	300	0.9900	0.9902
20	400	1.0000	0.9900
20	500	1.0000	0.9900
20	550	1.0000	0.9902

Test accuracy of transfer learning model is 0.9881

- Explanation on the code design
    - Firstly, I create new data sets which remove a digit from the original training, validation and test set of MNIST data. In the code, I manually remove data and label for digit '9'.
    - I made some changes to the default structure of the neural network in mnist_conv1.py file to make sure the classification accuracy is higher than 98%. The changes are:
        - One more convolutional layer is added after the first convolutional layer
        - Add dropout layer to drop half of the neurons of the first fully-connected layer
        - Change the output layer to have 9 outputs instead of 10 outputs
    - To train the network, I keep using Adam optimizer and mini-batch approach, where the mini-batch size is 100 examples. The number of epoch is increased to 20 to have higher accuracy. Every 100 mini-batch over the training set, I print the training and validation accuracies. For training, the dropout parameter is 0.5, which means it will drop half of the neurons in the previous layer. However, when evaluating the model, the dropout parameter is changed to 1. After training, I print out the test accuracy.
    - After training the 9-output model, I create a new output layer with 10 outputs corresponding to 10 digits in the MNIST data set, and connect it to the last hidden layer. Then I perform transfer learning by training only the parameters of the added layer using `var_list` argument of `minimize` method while fixing the other parameters. The training process also uses Adam optimizer with mini-batch approach. The data used for training and validating is now the original MNIST data set with all 10 digits. I also print the training and validation accuracies during training to see the progress of the learning process. After that, I print out the test accuracy of the transfer learning model.
- Interpretation of the results
    - We can see that the during the training process of both models, **the training and validation accuracies of both models increase**. This means the training processes are making good impact on the model by changing the parameters so that the models become more and more accurate.
    - The test accuracy of the pre-trained model is about 99.24%, while the test accuracy for the transfer learning model is a little bit lower, at 98.81%. This makes sense because when we perform transfer learning model, we only train the new layer while fixing the parameters of the other layers. Therefore, the fixed parameters are not optimized for the new label, which leads to **a lower accuracy of the transfer learning model**. However, with the accuracy of 98.81%, the transfer learning model can still be considered as a good model.

Task 3. PRINCIPLE COMPONENT ANALYSIS (PCA)

- Source code (TensorFlow v1.3.0, NumPy v1.13.3, MatPlotLib v1.5.1)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets('./MNIST_data', one_hot=True)
sess = tf.InteractiveSession()
x_ = tf.placeholder(tf.float32, shape=[None, 784])
y_ = tf.placeholder(tf.float32, shape=[None, 10])

# Convolutional layer
x_image = tf.reshape(x_, [-1,28,28,1])
W_conv = tf.Variable(tf.truncated_normal([5, 5, 1, 30], stddev=0.1))
b_conv = tf.Variable(tf.constant(0.1, shape=[30]))
h_conv = tf.nn.conv2d(x_image, W_conv, strides=[1, 1, 1, 1], padding='VALID')
h_relu = tf.nn.relu(h_conv + b_conv)
h_pool = tf.nn.max_pool(h_relu, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')


# Fully-connected layer
W_fc1 = tf.Variable(tf.truncated_normal([12 * 12 * 30, 500], stddev=0.1))
b_fc1 = tf.Variable(tf.constant(0.1, shape=[500]))
h_pool_flat = tf.reshape(h_pool, [-1, 12*12*30])
h_fc1 = tf.nn.relu(tf.matmul(h_pool_flat, W_fc1) + b_fc1)

# Output layer
W_fc2 = tf.Variable(tf.truncated_normal([500, 10], stddev=0.1))
b_fc2 = tf.Variable(tf.constant(0.1, shape=[10]))
y_hat=tf.nn.softmax(tf.matmul(h_fc1, W_fc2) + b_fc2)

# Train and Evaluate the Model
cross_entropy = - tf.reduce_sum(y_*tf.log(y_hat))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_hat,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

sess.run(tf.global_variables_initializer())
print '================================='
print '|Epoch\tBatch\t|Train\t|Val\t|'
print '|===============================|'
for j in range(5):
    for i in range(550):
        batch = mnist.train.next_batch(100)
        train_step.run(feed_dict={x_: batch[0], y_: batch[1]})
        if i%50 == 49:
            train_accuracy = accuracy.eval(feed_dict={x_:batch[0], y_: batch[1]})
            val_accuracy = accuracy.eval(feed_dict=\
                {x_: mnist.validation.images, y_:mnist.validation.labels})
            print '|%d\t|%d\t|%.4f\t|%.4f\t|'%(j+1, i+1, train_accuracy, val_accuracy)
print '|===============================|'
test_accuracy = accuracy.eval(feed_dict=\
    {x_: mnist.test.images, y_:mnist.test.labels})
print 'test accuracy=%.4f'%(test_accuracy)

vectorNum = 10000
mat1 = h_fc1.eval(feed_dict = { x_: mnist.test.images[0:vectorNum, :]})
mat2 = mat1 - np.mean(mat1, axis = 0)
W, s, V = np.linalg.svd(np.dot(mat2.T, mat2))
```

```
Z = np.dot(mat2, W)

# Figure 1: Plot first 2 columns of Z
c = ['blue', 'red', 'green', 'cyan', 'magenta', 'yellow', 'black', 'yellowgreen', 'orange'
, 'brown']
y_val = mnist.test.labels[0:vectorNum, :]
plt.figure(1)
ax = plt.subplot(111)
for currDigit in range(10):
    a = np.zeros((10))
    a[currDigit] = 1
    b = np.where(np.all(y_val == a, axis = 1))
    x0 = Z[b,0]
    x1 = Z[b,1]
    ax.scatter(x0, x1, s = 50, c = c[currDigit], label = currDigit, linewidths = 0.)
box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])
ax.legend(loc = 'center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)
plt.show()

# Get 100 examples from original data set
dataSetImages = []
dataSetLabels = []
dataSetSize = 100
expEachDigit = 10
digitNum = dataSetSize/expEachDigit

for currDigit in range(digitNum):
    i = j = 0
    while ( (i < expEachDigit) and (j < mnist.test.labels.shape[0]) ):
        argMax = np.argmax(mnist.test.labels[j,:])
        if ( argMax == currDigit ):
            dataSetImages.append(mnist.test.images[j,:])
            dataSetLabels.append(mnist.test.labels[j,:])
            i += 1
        j += 1

dataSetImages = np.array(dataSetImages)
dataSetLabels = np.array(dataSetLabels)

omega = h_fc1.eval(feed_dict = {x_: dataSetImages, y_: dataSetLabels})
Q = np.dot(omega, W)

# Figure 2: Plot first two columns of Q
plt.figure(2)
ax = plt.subplot(111)
for i in range(10):
    plt.scatter(Q[(i*10):((i+1)*10),0], Q[(i*10):((i+1)*10),1],\
            s = 50, c = c[i], label = i)
box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])
ax.legend(loc = 'center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)
plt.show()
```

- Plot

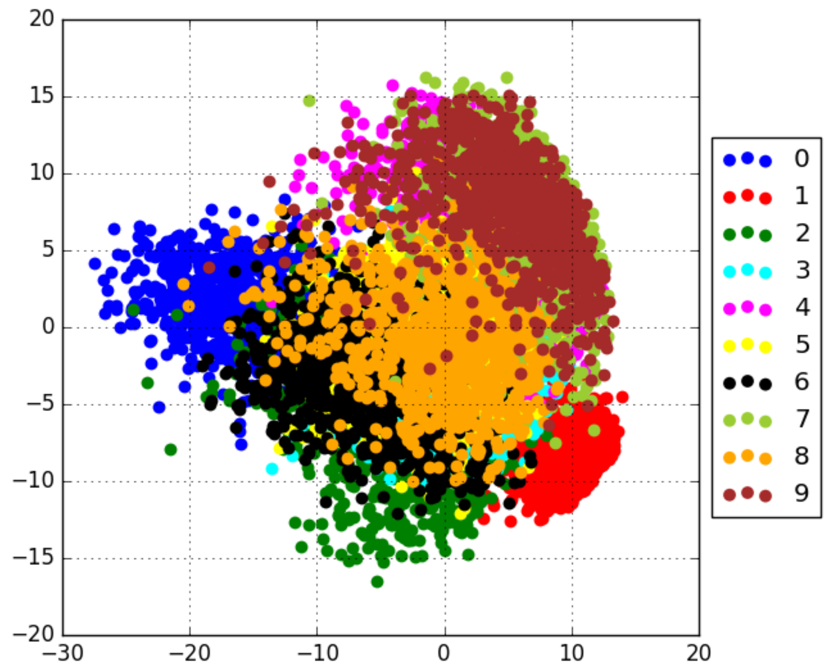

Figure 2. Plot of first two columns of Z

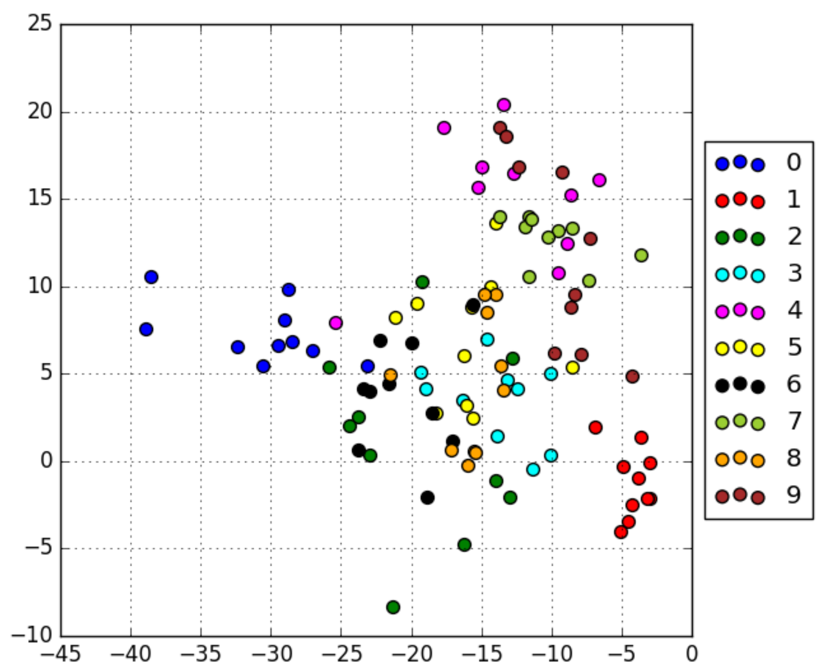

Figure 3. Plot of first two columns of Q

- Explanation of the code design

- In this task, I keep the network structure as well as the other learning parameters from mnist_conv1.py file.
- After training and get the appropriate parameters, I get the Ψ matrix by taking first 10000 examples from the test set. Then, I calculate Φ matrix by subtracting the mean of the Ψ matrix along each column. Applying SVD to $\Phi^T\Phi$, I get the W matrix. Then I calculate $Z = \Phi W$ and plot the first two columns of this matrix, as shown in Figure 2.
- Next, I create a test data set which has the first 10 examples labeled by '0', next 10 examples labeled by '1', and so on. I also apply the same procedure to calculate matrix Q. After that, I plot the first two columns of Q, as shown in Figure 3.

- Interpretation of the results
  - The first principle component (PC) is along the x axis, and the second PC is along the y axis. We can see that the variation of data along the x axis (first PC) is **bigger** than the variation of data along the y axis (second PC). This observation matches the characteristics of PCA, which finds the directions of maximum variance of data. The first two columns of Z and Q contain most of the valuable information of all of the features.
  - In the figure 2, we can also see that the labels are **clustered into different regions**. Figure 3 with less examples provides better vision. This connection is posited as an additional explanation of the success of PCA beyond the idea that it helps keep important parts (signal) and eliminate trivial parts (noise) of data.