In [1]:
```python
import csv
import os
import numpy as np
import random

data = []
with open("data.csv", 'r') as f:
    reader = csv.reader(f)
    for row in reader:
        temp = np.array(row).astype(np.float).tolist()
        data.append(temp)

maxi = [-9999999 for i in range(len(data[0]))]
mini = [9999999 for i in range(len(data[0]))]

for i in range(len(data)):
    for j in range(len(data[i])):
        if data[i][j]>maxi[j]:
            maxi[j] = data[i][j]
        if data[i][j]<mini[j]:
            mini[j] = data[i][j]

for i in range(len(data)):
    for j in range(len(data[i])):
        data[i][j] = (data[i][j]-mini[j])/(maxi[j]-mini[j])

random.shuffle(data)
```

In [2]:
```python
import tensorflow.compat.v1 as tf
tf.disable_eager_execution()

def initialize():
    #placeholder
    x = tf.placeholder(tf.float32, [None, D_in])
    y = tf.placeholder(tf.float32, [None, D_out])
    w = {
        'h1': tf.Variable(tf.random_normal([D_in, D_h1])),
        'h2': tf.Variable(tf.random_normal([D_h1, D_h2])),
        'h3': tf.Variable(tf.random_normal([D_h2, D_h3])),
        'h4': tf.Variable(tf.random_normal([D_h3, D_h4])),
        #'h5': tf.Variable(tf.random_normal([D_h4, D_h5])),
        #'h6': tf.Variable(tf.random_normal([D_h5, D_h6])),
        'out': tf.Variable(tf.random_normal([D_h4, D_out]))
    }
    b = {
        'h1': tf.Variable(tf.random_normal([D_h1])),
        'h2': tf.Variable(tf.random_normal([D_h2])),
        'h3': tf.Variable(tf.random_normal([D_h3])),
        'h4': tf.Variable(tf.random_normal([D_h4])),
        #'h5': tf.Variable(tf.random_normal([D_h5])),
        #'h6': tf.Variable(tf.random_normal([D_h6])),
        'out': tf.Variable(tf.random_normal([D_out]))
    }

    #activation functions #tf.layers.batch_normalization()
    def multilayer_perceptron(x):
        h1_layer = tf.sigmoid(tf.add(tf.matmul(x, w['h1']), b['h1']))
        h2_layer = tf.sigmoid(tf.add(tf.matmul(h1_layer, w['h2']), b['h2']))
        h3_layer = tf.sigmoid(tf.add(tf.matmul(h2_layer, w['h3']), b['h3']))
        h4_layer = tf.sigmoid(tf.add(tf.matmul(h3_layer, w['h4']), b['h4']))
        #h5_layer = tf.sigmoid(tf.add(tf.matmul(h4_layer, w['h5']), b['h5']))
        #h6_layer = tf.sigmoid(tf.add(tf.matmul(h5_layer, w['h6']), b['h6']))
        out_layer = tf.sigmoid(tf.add(tf.matmul(h4_layer, w['out']), b['out']))
        return out_layer
    pred = multilayer_perceptron(x)

    #loss function
    #cost = tf.reduce_sum(tf.where(tf.greater(pred, 0.932), 10*(y-pred)*(y-pred),
    #                     tf.where(tf.greater(0.042, pred), 10*(y-pred)*(y-pred),
    #                     (y-pred)*(y-pred))))
    cost = tf.reduce_sum((y-pred)*(y-pred))

    #optimizer and others
    optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)
    init = tf.global_variables_initializer()
    variables_dict = {
        'b1': b['h1'],
        'b2': b['h2'],
        'b3': b['h3'],
        'b4': b['h4'],
        #'b5': b['h5'],
        #'b6': b['h6'],
        'bout': b['out'],
        'w1': w['h1'],
        'w2': w['h2'],
        'w3': w['h3'],
        'w4': w['h4'],
        #'w5': w['h5'],
        #'w6': w['h6'],
        'wout': w['out']
    }
    saver = tf.train.Saver(variables_dict)
    """
    tf.summary.scalar('loss',cost)
    tf.summary.histogram('b1', b['h1'])
    tf.summary.histogram('b2', b['h2'])
    tf.summary.histogram('b3', b['h3'])
    tf.summary.histogram('b4', b['h4'])
    tf.summary.histogram('bout', b['out'])
    tf.summary.histogram('w1', w['h1'])
    tf.summary.histogram('w2', w['h2'])
    tf.summary.histogram('w3', w['h3'])
    tf.summary.histogram('w4', w['h4'])
    tf.summary.histogram('wout', w['out'])
    """
    saver = tf.train.Saver(max_to_keep=1000, keep_checkpoint_every_n_hours=1)
    return x, y, pred, cost, optimizer, init, saver


def evaluate(filename, dimensionx):
    xdata = []
    with open(filename) as f:
        lines = f.readlines()
    temp = []
    for line in lines:
        temp.append(float(line))
        if len(temp)==dimensionx:
            xdata.append(temp)
            temp = []

    with tf.Session() as sess:
        saver.restore(sess, "./model.ckpt")
        yhat = pred.eval(feed_dict = {x: xdata})


def evaluatewhentrain(xdata, modelfile):
    with tf.Session() as sess:
        saver.restore(sess, modelfile)
        yhat = pred.eval(feed_dict = {x: xdata})
    return yhat


def train(X_test, y_test):
    with tf.Session() as sess:
        sess.run(init)
        merged_all = tf.summary.merge_all()
        writer = tf.summary.FileWriter('logs/', sess.graph)
        for epoch in range(training_epochs):
            avg_cost = 0.
            total_batch = int(num/batch_size)
            if num%batch_size != 0:
                total_batch = total_batch + 1
            for i in range(total_batch):
                batch_xs = xdata[i*batch_size:(i+1)*batch_size]
                batch_ys = ydata[i*batch_size:(i+1)*batch_size]
                _, c = sess.run([optimizer, cost], feed_dict={x: batch_xs, y: batch_ys})
                avg_cost += c / total_batch
            if (epoch+1) % display_step == 0 or epoch == 0:
                print("Epoch:", '%04d' % (epoch+1), " cost=", "{:.9f}".format(avg_cost))
                #merged = sess.run(merged_all, feed_dict={x: batch_xs, y: batch_ys})
                #writer.add_summary(merged, epoch)
            if (epoch+1) % save_step == 0:
                saver.save(sess, "./model"+str(epoch)+".ckpt")
                print("save model done.")
                yhat = np.array(evaluatewhentrain(X_test,"./model"+str(epoch)+".ckpt"))
                accuracy = 100 - np.around(np.median(np.abs(yhat-y_test)/yhat*100,axis=0),de
cimals=1)
                print('5-layer ANN', accuracy, '% Step '+str(epoch))
```

In [3]:
```python
#training settings
training_epochs = 1000
save_step = 100

batch_size = 100
display_step = 10

#network structure
D_in, D_out = 8, 1
D_h1 = 20
D_h2 = 20
D_h3 = 20
D_h4 = 20
#D_h5 = 20
#D_h6 = 20

#initialize
x, y, pred, cost, optimizer, init, saver = initialize()

#load data
xdata = np.array(data)[:500,0:8]
ydata = np.mat(np.array(data)[:500,8]).T
X_test = np.array(data)[500:,0:8]
y_test = np.mat(np.array(data)[500:,8]).T

num = len(xdata)
#print(X_test)
#print(y_test)
```

In [4]:
```python
train(X_test, y_test)
```

```
Epoch: 0001  cost= 7.634255886
Epoch: 0010  cost= 1.160608697
Epoch: 0020  cost= 1.084276414
Epoch: 0030  cost= 1.033107996
Epoch: 0040  cost= 0.986758208
Epoch: 0050  cost= 0.942865157
Epoch: 0060  cost= 0.902528405
Epoch: 0070  cost= 0.868660140
Epoch: 0080  cost= 0.844467199
Epoch: 0090  cost= 0.826716185
Epoch: 0100  cost= 0.810775602
save model done.
INFO:tensorflow:Restoring parameters from ./model99.ckpt
5-layer ANN [[77.9]] % Step 99
Epoch: 0110  cost= 0.794251847
Epoch: 0120  cost= 0.776368058
Epoch: 0130  cost= 0.757749736
Epoch: 0140  cost= 0.740274036
Epoch: 0150  cost= 0.725482833
Epoch: 0160  cost= 0.713436246
Epoch: 0170  cost= 0.703327596
Epoch: 0180  cost= 0.694372809
Epoch: 0190  cost= 0.686074901
Epoch: 0200  cost= 0.678069425
save model done.
INFO:tensorflow:Restoring parameters from ./model199.ckpt
5-layer ANN [[75.2]] % Step 199
Epoch: 0210  cost= 0.670149934
Epoch: 0220  cost= 0.662253439
Epoch: 0230  cost= 0.654404473
Epoch: 0240  cost= 0.646673894
Epoch: 0250  cost= 0.639148051
Epoch: 0260  cost= 0.631910449
Epoch: 0270  cost= 0.625028259
Epoch: 0280  cost= 0.618547177
Epoch: 0290  cost= 0.612489545
Epoch: 0300  cost= 0.606857407
save model done.
INFO:tensorflow:Restoring parameters from ./model299.ckpt
5-layer ANN [[75.3]] % Step 299
Epoch: 0310  cost= 0.601636124
Epoch: 0320  cost= 0.596800685
Epoch: 0330  cost= 0.592323381
Epoch: 0340  cost= 0.588180643
Epoch: 0350  cost= 0.584358054
Epoch: 0360  cost= 0.580849177
Epoch: 0370  cost= 0.577648592
Epoch: 0380  cost= 0.574746019
Epoch: 0390  cost= 0.572122055
Epoch: 0400  cost= 0.569751126
save model done.
INFO:tensorflow:Restoring parameters from ./model399.ckpt
5-layer ANN [[74.4]] % Step 399
Epoch: 0410  cost= 0.567603499
Epoch: 0420  cost= 0.565649980
Epoch: 0430  cost= 0.563863558
Epoch: 0440  cost= 0.562220168
Epoch: 0450  cost= 0.560699856
Epoch: 0460  cost= 0.559285700
Epoch: 0470  cost= 0.557963848
Epoch: 0480  cost= 0.556722593
Epoch: 0490  cost= 0.555552632
Epoch: 0500  cost= 0.554445952
save model done.
INFO:tensorflow:Restoring parameters from ./model499.ckpt
5-layer ANN [[73.3]] % Step 499
Epoch: 0510  cost= 0.553395730
Epoch: 0520  cost= 0.552396345
Epoch: 0530  cost= 0.551442826
Epoch: 0540  cost= 0.550530952
Epoch: 0550  cost= 0.549656731
Epoch: 0560  cost= 0.548816967
Epoch: 0570  cost= 0.548008394
Epoch: 0580  cost= 0.547228312
Epoch: 0590  cost= 0.546474326
Epoch: 0600  cost= 0.545743996
save model done.
INFO:tensorflow:Restoring parameters from ./model599.ckpt
5-layer ANN [[73.8]] % Step 599
Epoch: 0610  cost= 0.545035243
Epoch: 0620  cost= 0.544346052
Epoch: 0630  cost= 0.543674695
Epoch: 0640  cost= 0.543019676
Epoch: 0650  cost= 0.542379278
Epoch: 0660  cost= 0.541752231
Epoch: 0670  cost= 0.541137308
Epoch: 0680  cost= 0.540533239
Epoch: 0690  cost= 0.539938879
Epoch: 0700  cost= 0.539353412
save model done.
INFO:tensorflow:Restoring parameters from ./model699.ckpt
5-layer ANN [[74.3]] % Step 699
Epoch: 0710  cost= 0.538775432
Epoch: 0720  cost= 0.538204646
Epoch: 0730  cost= 0.537639785
Epoch: 0740  cost= 0.537079936
Epoch: 0750  cost= 0.536524606
Epoch: 0760  cost= 0.535972893
Epoch: 0770  cost= 0.535424137
Epoch: 0780  cost= 0.534877580
Epoch: 0790  cost= 0.534332538
Epoch: 0800  cost= 0.533788306
save model done.
INFO:tensorflow:Restoring parameters from ./model799.ckpt
5-layer ANN [[73.4]] % Step 799
Epoch: 0810  cost= 0.533244550
Epoch: 0820  cost= 0.532700402
Epoch: 0830  cost= 0.532155275
Epoch: 0840  cost= 0.531608784
Epoch: 0850  cost= 0.531060290
Epoch: 0860  cost= 0.530509198
Epoch: 0870  cost= 0.529954863
Epoch: 0880  cost= 0.529396987
Epoch: 0890  cost= 0.528834993
Epoch: 0900  cost= 0.528268188
save model done.
INFO:tensorflow:Restoring parameters from ./model899.ckpt
5-layer ANN [[72.4]] % Step 899
Epoch: 0910  cost= 0.527696365
Epoch: 0920  cost= 0.527118677
Epoch: 0930  cost= 0.526534790
Epoch: 0940  cost= 0.525944167
Epoch: 0950  cost= 0.525346196
Epoch: 0960  cost= 0.524740416
Epoch: 0970  cost= 0.524126214
Epoch: 0980  cost= 0.523502970
Epoch: 0990  cost= 0.522870326
Epoch: 1000  cost= 0.522227478
save model done.
INFO:tensorflow:Restoring parameters from ./model999.ckpt
5-layer ANN [[72.7]] % Step 999
```

In [5]:
```python
X_pre = X_test

with tf.Session() as sess:
    saver.restore(sess, "./model999.ckpt")
    yhat = pred.eval(feed_dict = {x: X_pre})

X_pre = X_pre.tolist()

accuracy = 0
for i in range(len(X_pre)):
    for j in range(len(X_pre[i])):
        X_pre[i][j] = X_pre[i][j]*(maxi[j]-mini[j])+mini[j]
    X_pre[i].extend(y_test[i]*(maxi[j+1]-mini[j+1])+mini[j+1])
    X_pre[i].extend(yhat[i]*(maxi[j+1]-mini[j+1])+mini[j+1])
    accuracy += 1-(abs(X_pre[i][len(X_pre[i])-1]-X_pre[i][len(X_pre[i])-2])/X_pre[i][len(X_p
re[i])-2])
accuracy = accuracy/len(X_pre)
print(accuracy)

np.savetxt('pre_results.csv', X_pre, delimiter = ',')
```

```
INFO:tensorflow:Restoring parameters from ./model999.ckpt
[[0.85545222]]
```

In [ ]: