TF Multiple Hidden Layers: Regression on Boston Data
Batched, Parameterized, with Dropout

This is adapted from Frossard's tutorial (http://www.cs.toronto.edu/~frossard/post/tensorflow/).
This approach is not batched, and the number of layers is fixed.

D. Thiebaut
August 2016

**Import the Libraries and Tools**

```python
In [61]:
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.contrib import learn
from sklearn import cross_validation
from sklearn import preprocessing
from sklearn import metrics
from __future__ import print_function

%matplotlib inline
```

**Import the Boston Data**

We don't worry about adding column names to the data.
We scale the inputs to have mean 0 and standard variation 1.

In [63]:
scaler = preprocessing.StandardScaler( )
train_x = scaler.fit_transform( train_x )
test_x = scaler.fit_transform( test_x )

We verify that we have 13 features...

In [64]:
numFeatures = train_x.shape[1]

print( "number of features = ", numFeatures )

number of features = 13

**Input & Output Place-Holders**

Define 2 place holders to the graph, one for the inputs one for the outputs...

In [65]:
with tf.name_scope("IO"):
    inputs = tf.placeholder(tf.float32, [None, numFeatures], name="X")
    outputs = tf.placeholder(tf.float32, [None, 1], name="Yhat")

**Define the Coeffs for the Layers**

For each layer the input vector will be multiplied by a matrix $h$ of dim $n \times m$, where $n$ is the dimension of the input vector and $m$ the dimention of the output vector. Then a bias vector of dimension $m$ is added to the product.
Define the Layer operations as a Python function

```python
In [66]:
    with tf.name_scope("LAYER"):
        # network architecture
        Layers = [numFeatures, 52, 104, 52, 52, 52, 1]
        h = []
        b = []
        for i in range(1, len(Layers)):
            h.append( tf.Variable(tf.random_normal([Layers[i-1], Layers[i]], 0, 0.1), dtype=tf.float32)
            b.append( tf.Variable(tf.random_normal([Layers[i]], 0, 0.1), dtype=tf.float32)
        dropout = 0.990
        keep_prob = tf.placeholder(tf.float32)
```

Define the operations that are performed

We define what happens to the inputs (x), when they are provided, and what we do with the outputs of the layers (compare them to the y values), and the type of minimization that must be done.

```python
In [67]:
    def model( inputs, h, b ):
        lastY = inputs
        for i, (hi, bi) in enumerate( zip( h, b ) ):
            y = tf.add( tf.matmul( lastY, hi), bi )

        if i==len(h)-1:
            return y

        lastY = tf.nn.sigmoid( y )
        lastY = tf.nn.dropout( lastY, dropout )
```

Train the Model

```python
In [68]:
    with tf.name_scope("train"):

        learning_rate = 0.250
        #yout = model2( inputs, [h1, b1, h2, b2, h3, b3, hout, bout] )
        yout = model( inputs, h, b )

        cost_op = tf.reduce_mean( tf.pow( yout - outputs, 2 )
        #cost_op = tf.reduce_sum( tf.pow( yout - outputs, 2 )
        #cost_op = tf.reduce_mean(-tf.reduce_sum( yout * tf.log( outputs ) )

        #train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_op)
        #train_op = tf.train.AdamOptimizer( learning_rate=learning_rate ).minimize(cost_op)
        train_op = tf.train.AdamOptimizer( learning_rate=learning_rate ).minimize(cost_op)
```
We are now ready to go through many sessions, and in each one train the model. Here we train on the whole x-train and y-train data, rather than batching into smaller groups.
In [69]:
# define variables/constants that control the training
epoch = 0  # counter for number of rounds training network
last_cost = 0  # keep track of last cost to measure difference
max_epochs = 20000  # total number of training sessions
tolerance = 1e-6  # we stop when diff in costs less than that
batch_size = 50  # we batch the data in groups of this size
num_samples = train_y.shape[0]  # number of samples in train
num_batches = int( num_samples / batch_size )  # compute number of batches, # batch size

print( "batch size = ", batch_size )
print( "test length = ", num_samples )
print( "number batches = ", num_batches )
print( "--- Beginning Training ---" )

sess = tf.Session()  # Create TensorFlow session
with sess.as_default():

    # initialize the variables
    init = tf.initialize_all_variables()
    sess.run(init)

    # start training until we stop, either because we’ve reached the max
    # number of epochs, or successive errors are close enough to each other
    # (less than tolerance)
    costs = [
    epochs= [
    while True:
        # Do the training
        cost = 0
        for n in range( num_batches ):
            batch_x = train_x[ n*batch_size : (n+1)*batch_size ]
            batch_y = train_y[ n*batch_size : (n+1)*batch_size ]
            sess.run( train_op, feed_dict={inputs: batch_x, outputs: batch_y} )
            c = sess.run(cost_op, feed_dict={inputs: batch_x, outputs: batch_y})
            cost += c
        cost /= num_batches

        costs.append( cost )
        epochs.append( epoch )

        # Update the user every 1000 epochs
        if epoch % 1000==0:
            print( "Epoch: %d - Error diff: %1.8f" %(epoch, cost) )

        # time to stop?
        if epoch > max_epochs or abs(last_cost - cost) < tolerance:
            print( "--- STOPPING ---" )
            break
        last_cost = cost
        epoch += 1

    # we’re done...
    # print some statistics...
print( "Test Cost =", sess.run(cost_op, feed_dict={inputs: test_x, output: test_y})

# compute the predicted output for test_x
pred_y = sess.run( yout, feed_dict={inputs: test_x, outputs: test_y} )

print( "\nA few predictions versus real data from test set\nPrediction\nreal predicted\n23.0  28.6  
32.0  34.3
13.0  18.0
22.0  23.3
16.0  15.5
20.0  22.0
17.0  21.5
14.0  17.0
19.0  22.3
16.0  20.6\nR2 score
In [70]:
    r2 = metrics.r2_score(test_y, pred_y)
    print( "mean squared error = ", metrics.mean_squared_error(test_y, pred_y))
    print( "r2 score (coef determination) = ", metrics.r2_score(test_y, pred_y))

    mean squared error =  15.9532791154
    r2 score (coef determination) =  0.783881796368

Plot Prediction vs. Real Housing Price
In [71]:

```python
fig = plt.figure()
xmin = min(test_y)
xmax = max(test_y) + 5
plt.xlim(xmin, xmax)

x = np.linspace(xmin, xmax)
plt.scatter(test_y, pred_y)
plt.plot(x, x)

plt.text(5, 50, r'$r^2 = %.4f$' % r2)
plt.xlabel("Test y")
plt.ylabel("predicted y")
plt.title("Prediction vs. Actual Y")
# plt.savefig("images/sigmoid_adagrad_52_39_26_13_1.png")
plt.show()
fig.savefig('files/PredVsRealBoston.png', bbox_inches='tight')
```

```python
fig = plt.figure()
plt.scatter(test_y, -test_y + pred_y)
plt.axhline(0, color='black')
plt.xlabel("Test y")
plt.ylabel("Test y - Predicted Y")
plt.title("Residuals")
plt.show()
fig.savefig('files/ResidualsBoston.png', bbox_inches='tight')
```
Plot Cost vs Epochs

In [72]:
    fig = plt.figure()
    plt.semilogy(epochs, costs)
    plt.xlabel("Epochs")
    plt.ylabel("Cost")
    plt.title("Cost vs. Epochs")
    plt.show()
    fig.savefig('files/CostVsEpochs.png', bbox_inches='tight')