TF Mutiple 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

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.

In [62]:
```
boston = learn.datasets.load_dataset('boston')
#print( "boston = ", boston )
x, y = boston.data, boston.target
y.resize( y.size, 1 ) #make y = [[x], [x], [x], ... ]

train_x, test_x, train_y, test_y = cross_validation.train_test_split(
                                    x, y, test_size=0.2, random_state=42)

print( "Dimension of Boston test_x = ", test_x.shape )
print( "Dimension of test_y = ", test_y.shape )

print( "Dimension of Boston train_x = ", train_x.shape )
print( "Dimension of train_y = ", train_y.shape )
```

```
Dimension of Boston test_x =  (102, 13)
Dimension of test_y =  (102, 1)
Dimension of Boston train_x =  (404, 13)
Dimension of train_y =  (404, 1)
```

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 x 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.

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, 
        b.append( tf.Variable(tf.random_normal([Layers[i]], 0, 0.1, dtype=tf

    dropout = 0.990                              # Dropout, probability to kee
    keep_prob = tf.placeholder(tf.float32)   # dropout (keep probability)
```

Define the Layer operations as a Python funtion

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

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

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

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.

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(co
    #train_op = tf.train.AdamOptimizer( learning_rate=learning_rate ).minimi
    train_op = tf.train.AdagradOptimizer( learning_rate=learning_rate ).mini
```

Train the Model

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 trai
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
            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, outpu

    # 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\r
    for (y, yHat ) in zip( test_y, pred_y )[0:10]:
        print( "%1.1f\t%1.1f" % (y, yHat ) )
```

```
batch size =  50
test length=  404
number batches =  8
--- Beginning Training ---
Epoch: 0 - Error diff: 118.00605869
Epoch: 1000 - Error diff: 1.73801895
Epoch: 2000 - Error diff: 0.71169004
Epoch: 3000 - Error diff: 0.69569829
Epoch: 4000 - Error diff: 0.55786825
Epoch: 5000 - Error diff: 0.42155909
Epoch: 6000 - Error diff: 0.33274260
Epoch: 7000 - Error diff: 0.37239324
Epoch: 8000 - Error diff: 0.35726526
Epoch: 9000 - Error diff: 0.39215576
Epoch: 10000 - Error diff: 0.32150087
Epoch: 11000 - Error diff: 0.35009851
Epoch: 12000 - Error diff: 0.28756304
Epoch: 13000 - Error diff: 0.30331052
Epoch: 14000 - Error diff: 0.31456433
Epoch: 15000 - Error diff: 0.22728056
Epoch: 16000 - Error diff: 0.27345408
Epoch: 17000 - Error diff: 0.24991406
Epoch: 18000 - Error diff: 0.23854101
Epoch: 19000 - Error diff: 0.26857675
Epoch: 20000 - Error diff: 0.22771443
Epoch: 21000 - Error diff: 0.30227334
--- STOPPING ---
Test Cost = 15.7541

A few predictions versus real data from test set
Prediction
real    predicted
23.0    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
```

R2 score

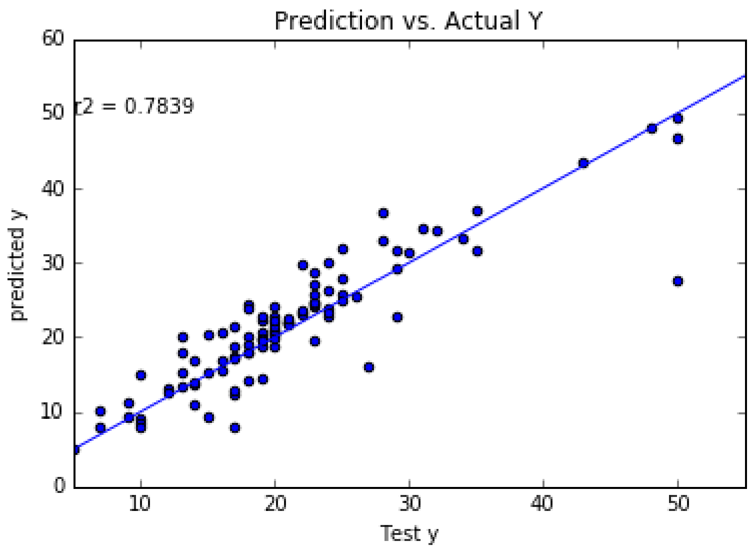
```
In [70]: r2 =  metrics.r2_score(test_y, pred_y)
         print( "mean squared error = ", metrics.mean_squared_error(test_y, pred_y))
         print( "r2 score (coef determination) = ", metrics.r2_score(test_y, pred_y))
```

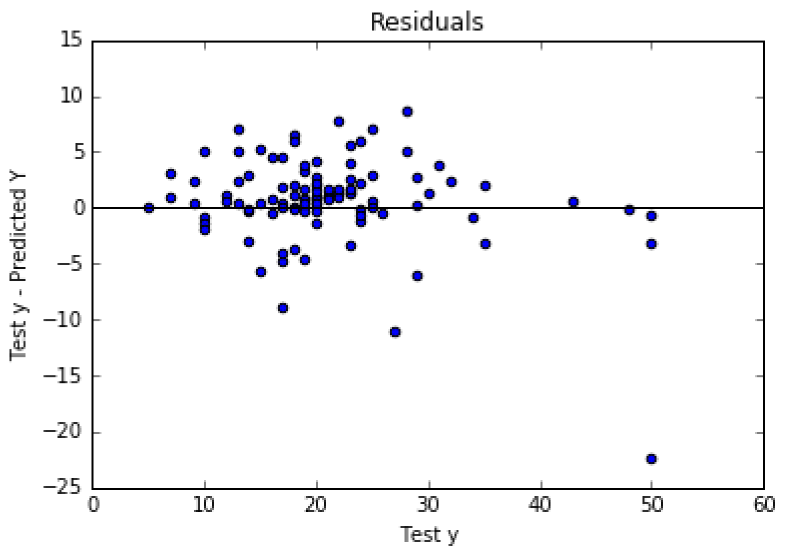
```
mean squared error =  15.9532791154
r2 score (coef determination) =  0.783881796368
```

Plot Prediction vs. Real Housing Price

In [71]:

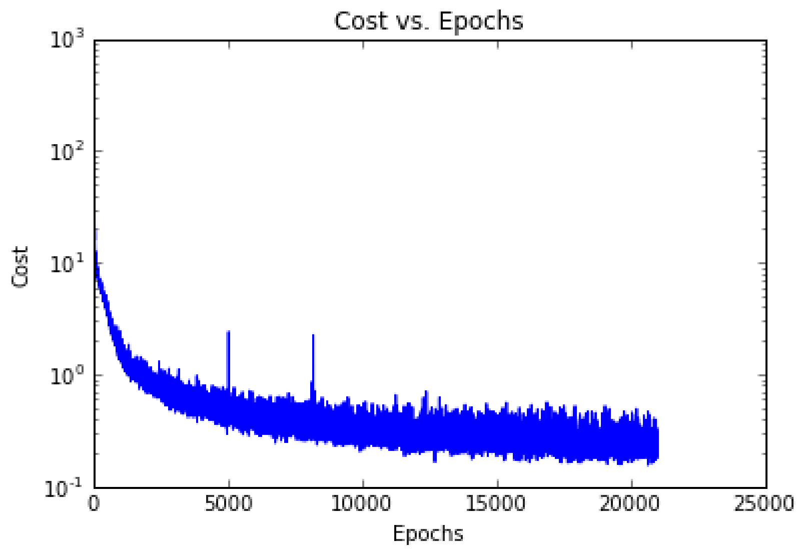
```
fig = plt.figure()
xmin = min(test_y)
xmax = max(test_y) + 5
plt.xlim(xmin, xmax)

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

plt.text(5, 50, r'r2 = %1.4f' % r2)
plt.xlabel( "Test y" )
plt.ylabel( "predicted y" )
plt.title( "Prediction vs. Actual Y" )
#plt.save( "images/sigmoid_adagrad_52_39_26_13_1.png")
plt.show()
fig.savefig('files/PredVsRealBoston.png', bbox_inches='tight')

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')
```



In [ ]:
```
```