SKLearn Tutorial: DNN on Boston Data

This tutorial follows very closely two other good tutorials and merges elements from both:

- https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/skflow/boston.py (https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/skflow/boston.py)
- http://bigdataexaminer.com/uncategorized/how-to-run-linear-regression-in-python-scikit-learn/ (http://bigdataexaminer.com/uncategorized/how-to-run-linear-regression-in-python-scikit-learn/)

D. Thiebaut
August 2016.

Get the Boston Data

This part is basically taken directly from the bigdataexaminer (http://bigdataexaminer.com/uncategorized/how-to-run-linear-regression-in-python-scikit-learn/) tutorial.

Imports, first!

In [170]:
```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from sklearn import cross_validation
from sklearn import metrics
from sklearn import preprocessing
import tensorflow as tf
#from tensorflow.contrib import learn
from tensorflow.contrib import learn
import pandas as pd
```

Get the data...

In [171]:
```
from sklearn.datasets import load_boston
boston = load_boston()
print( "type of boston = ", type(boston))
```

```
type of boston =  <class 'sklearn.datasets.base.Bunch'>
```

In [172]:
```
boston.keys()
```

Out[172]:
```
['data', 'feature_names', 'DESCR', 'target']
```

In [173]:
```
boston.data.shape
```

Out[173]:
```
(506, 13)
```

In [174]:
```
print( boston.feature_names )
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

In [175]:
```
print( boston.DESCR )
```

```
Boston House Prices dataset

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,
000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds rive
r; 0 otherwise)
```

In [176]:
```
print( "target = ",
          ",".join( str(k) for k in boston.target[0:5]  ),
          "...",
          ", ".join( str(k) for k in boston.target[-5:] ) )
```

```
target =  24.0,21.6,34.7,33.4,36.2 ... 22.4, 20.6, 23.9, 22.0, 11.9
```

Convert the boston data into a panda data-frame

In [177]:
```
bostonDF = pd.DataFrame( boston.data )
bostonDF.head()
```

Out[177]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33

Add column names

In [178]:
```
bostonDF.columns = boston.feature_names
bostonDF.head()
```

Out[178]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
0	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33

In [179]:
```
print( "Number of features = ", len( bostonDF.columns ) )
```

```
Number of features =  13
```

In [180]:
```
X = bostonDF
y = boston.target
print( "shape of X = ", X.shape, "  shape of y = ", y.shape )
```

```
shape of X =  (506, 13)   shape of y =  (506,)
```

Split the data into training and test data.

In [181]:
```
 X_train, X_test, y_train, y_test = cross_validation.train_test_split(
      X, y, test_size=0.2, random_state=42)
```

Scale the X data to 0 mean and unit standard deviation

In [182]:
```
scaler = preprocessing.StandardScaler( )
X_train = scaler.fit_transform( X_train )
X_train
```

Out[182]:
```
array([[ 1.29133866, -0.50032012,  1.03323679, ...,  0.84534281,
        -0.07433689,  1.75350503],
       [-0.3338103 , -0.50032012, -0.41315956, ...,  1.20474139,
         0.4301838 , -0.5614742 ],
       [-0.40072291,  1.01327135, -0.71521823, ..., -0.63717631,
         0.06529747, -0.65159505],
       ...,
       [-0.40294118,  2.95931752, -1.30336132, ..., -0.59225149,
         0.37901005, -0.91069248],
       [ 0.85524904, -0.50032012,  1.03323679, ...,  0.84534281,
        -2.69458597,  1.52257036],
       [-0.37881118, -0.50032012, -0.35216694, ...,  1.15981657,
        -3.12158061, -0.25731635]])
```

In [183]:
```
y_train
```

```
array([22. ,  7.2, 20.4, 13.8, 13. , 18.4, 23.1, 21.2, 23.1,
       23.5, 50. , 26.6, 22.2, 50. ,  8.3, 23.3, 21.7, 18.9,
       18.4, 17.4, 13.4, 12.1, 26.6, 21.7, 28.4, 20.5, 22. ,
       13.9, 11.3, 29.9, 26.6, 10.5, 23.2, 24.4, 46. , 21.9,
        7.5, 36.2, 44. , 17.8, 27.5, 37.6, 14.1, 28.1, 10.2,
       19.1, 43.8, 27.9, 25. , 16. , 16.6, 13.2, 50. , 22.2,
       32.9, 15.2, 14.8, 13.8, 24.3, 33.8, 22.3, 50. ,  9.5,
       13.3, 22.2, 18.1, 18. , 25. , 16.5, 23. , 20.1, 33. ,
       24.8, 18.2, 13.1, 34.9, 10.2, 19.9, 27.9, 23.3, 35.1,
       12.8, 22. , 18.5, 25.1, 22.5, 22.4, 28.6, 19.5, 24.8,
       24.5, 21.4, 33.1, 22.9, 20.7, 24.1, 50. , 24.7, 28.7,
        7.2, 37. , 20.3, 30.1, 19.5, 23.4, 11.5, 21.6, 14.9,
       15.2, 19.4,  8.4, 28. , 22.6, 13.5, 14.5, 31. , 10.9,
       21.9, 22. , 19. , 21.4, 25. , 17.5, 36.5, 20.1, 20.4,
       16.2, 23.6,  7.4, 35.2, 50. , 19.3, 21.2, 15.6, 33.4,
       19.1, 21. , 23.7, 18.9, 16.8, 19.7, 17.7, 22.6, 11.8,
       34.9, 20.6, 20.2, 32. , 22.3, 23.3, 14.4, 31.2, 24. ,
       29.6, 19.6, 21.6, 20. , 27. , 33.2, 15.4, 30.5,  7.2,
       23.9, 16.3, 23.9, 50. , 22.8, 15.4, 19.2, 19.6, 22.6,
       33.2, 50. , 22.2, 14.9, 19.8, 23.7, 19. , 20.3, 11.9,
```

Building a [10,10] Neural Net

In [184]:
```
# Build a 2 layer fully connected DNN
feature_columns = boston.feature_names
regressor = learn.DNNRegressor( feature_columns=None,
                                hidden_units=[10, 10] )#,
                                #model_dir = '/tmp/tf')
regressor.fit( X_train, y_train, steps=5000, batch_size=1 )
```

Out[184]:
```
DNNRegressor()
```

Predict and score

In [185]:
```
y_predicted = regressor.predict( scaler.transform( X_test ) )
score = metrics.mean_squared_error( y_predicted, y_test )
print('MSE: {0:f}'.format(score) )
```

```
MSE: 14.098925
```

In [186]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.scatter( y_predicted, y_test, s=5 )
plt.xlabel( "Predicted Prices")
plt.ylabel( "Real Prices")
plt.title( "Real vs Predicted Housing Prices")
```

Out[186]: <matplotlib.text.Text at 0x124891e10>

Generating the graph of the NN with tensorboard

In order to generate the graph of the network, we simply add "model_dir = '/tmp/tf' " to the DNNRegressor() instantiation:

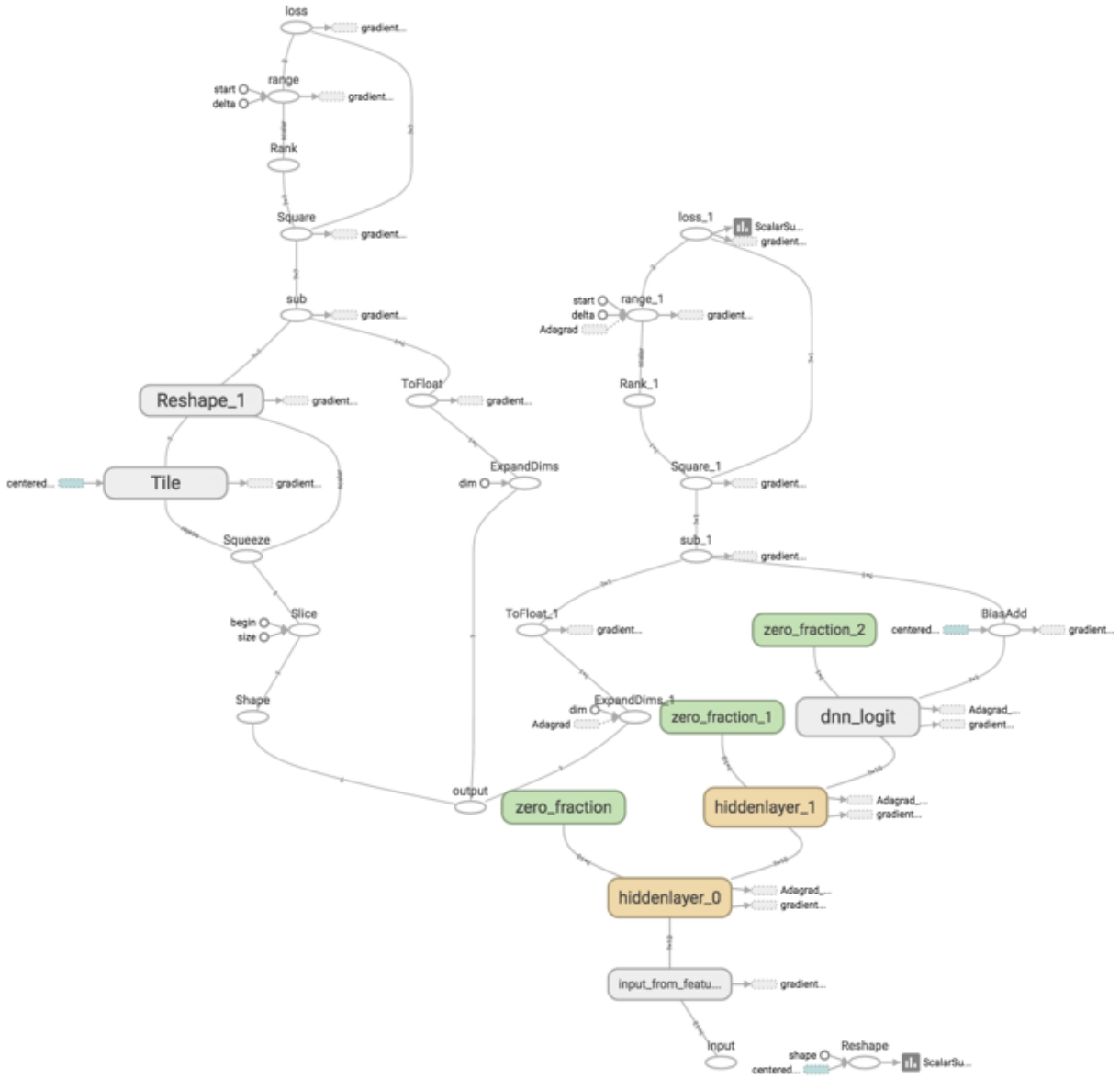
In [187]:
```
'''
regressor = learn.DNNRegressor( feature_columns=None,
                                hidden_units=[10, 10] ),
                                model_dir = '/tmp/tf')
'''
```

Out[187]:
```
"\nregressor = learn.DNNRegressor( feature_columns=None,\n
                  hidden_units=[10, 10] ),\n
  model_dir = '/tmp/tf')\n"
```

This might generate an error when we try to fit, but that will happen after the useful information is written in /tmp/tf, and we can safely run tensorboard as follows, from the command line:

```
tensorboard --logdir /tmp/tf
```

And here's the graph we get for the 10x10 NN:



Building a [13,13] Neural Net

In [188]:
```
# Build a 2 layer fully connected DNN
regressor = learn.DNNRegressor( feature_columns=None,
                                hidden_units=[13, 13])
regressor.fit(X_train, y_train, steps=5000, batch_size=10)
```

Out[188]:
```
DNNRegressor()
```

Predict and score

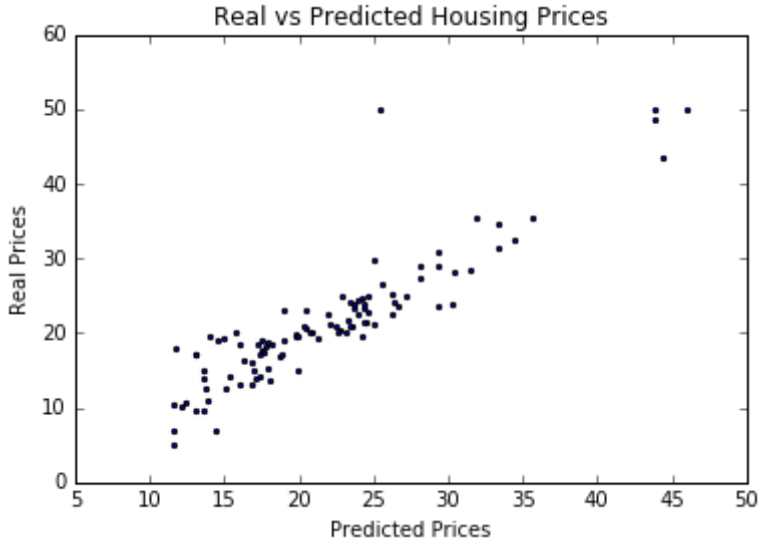
In [189]:
```
y_test_predicted = regressor.predict( scaler.transform( X_test ) )
scoreTest = metrics.mean_squared_error( y_test_predicted, y_test )
print('MSE Test Data: {0:f}'.format(scoreTest) )

y_train_predicted = regressor.predict( X_train  )
scoreTrain = metrics.mean_squared_error( y_train_predicted, y_train )
print('MSE Train Data: {0:f}'.format(scoreTrain ) )
```

```
MSE Test Data: 10.162460
MSE Train Data: 5.946833
```

In [190]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.scatter( y_predicted, y_test, s=5 )
plt.xlabel( "Predicted Prices")
plt.ylabel( "Real Prices")
plt.title( "Real vs Predicted Housing Prices")
```

Out[190]:
```
<matplotlib.text.Text at 0x1281e5a10>
```



Building a [13,13,10] Neural Net

In [191]:
```
# Build a 2 layer fully connected DNN
feature_columns = boston.feature_names
regressor = learn.DNNRegressor( feature_columns=None,
                                hidden_units=[13, 13, 10],
                               model_dir = '/tmp/tf/')
regressor.fit(X_train, y_train, steps=5000, batch_size=1)
```

Out[191]:
```
DNNRegressor()
```

Predict and score

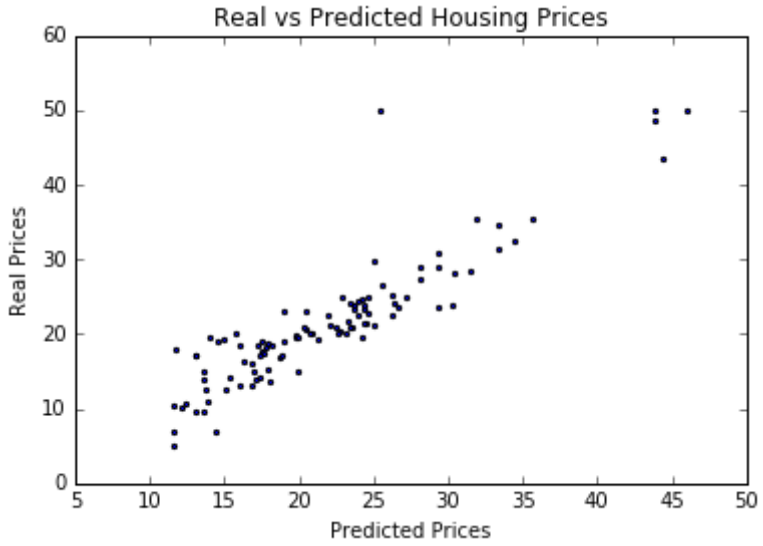
In [192]:
```
y_test_predicted = regressor.predict( scaler.transform( X_test ) )
scoreTest = metrics.mean_squared_error( y_test_predicted, y_test )
print('MSE Test Data: {0:f}'.format(scoreTest) )

y_train_predicted = regressor.predict( X_train  )
scoreTrain = metrics.mean_squared_error( y_train_predicted, y_train )
print('MSE Train Data: {0:f}'.format(scoreTrain ) )
```

```
MSE Test Data: 11.432168
MSE Train Data: 10.082671
```

In [193]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.scatter( y_predicted, y_test, s=5 )
plt.xlabel( "Predicted Prices")
plt.ylabel( "Real Prices")
plt.title( "Real vs Predicted Housing Prices")
```

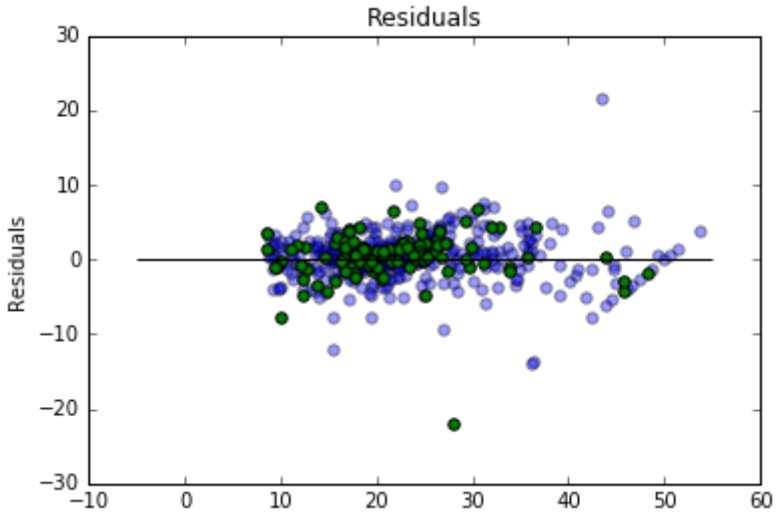
Out[193]:
```
<matplotlib.text.Text at 0x128ec5bd0>
```



Plotting Residuals

In [194]:
```
plt.scatter( y_train_predicted, y_train_predicted - y_train,
           c ='b', s=30, alpha=0.4 )
plt.scatter(y_test_predicted, y_test_predicted - y_test,
           c ='g', s=30 )
plt.hlines( y=0, xmin=-5, xmax=55)
plt.title( "Residuals" )
plt.ylabel( "Residuals" )
```

Out[194]: <matplotlib.text.Text at 0x128f50110>



In [ ]: