In [1]:
```
# import required packages

import numpy as np
import pandas as pd
import math
import sklearn
import sklearn.preprocessing
import datetime
import os
import matplotlib.pyplot as plt
import tensorflow as tf
```

In [2]:
```
# display parent directory and working directory

print(os.path.dirname(os.getcwd())+':', os.listdir(os.path.dirname(os.getcwd
())));
print(os.getcwd()+':', os.listdir(os.getcwd()));
```

```
C:\Users\sanch\OneDrive\Documents\Graduate School\UT-Austin - MSIS\INF 385T -
Intro to Machine Learning: ['Class Lectures', 'Homework', 'Labs', 'Project',
'Syllabus.pdf', 'Textbooks']
C:\Users\sanch\OneDrive\Documents\Graduate School\UT-Austin - MSIS\INF 385T -
Intro to Machine Learning\Project: ['.ipynb_checkpoints', 'fundamentals.csv',
'INF385T - Prioject Pre-prosal - Sanchit Singhal Gaurav Lalwani.pdf', 'INF385
T - Project Outline - Sanchit Singhal Gaurav Lalwani.pdf', 'INF385T - Project
Proposal - Sanchit Singhal Gaurav Lalwani.pdf', 'INF385T_ML_Project_Models_v1
0.1.ipynb', 'INF385T_ML_Project_Prototype.pdf', 'Pre-Proposal.docx', 'prices-
split-adjusted.csv', 'prices.csv', 'Project Prototype.ipynb', 'securities.cs
v']
```

In [3]:
```
# split data in 80%/10%/10% train/validation/test sets

valid_set_size_percentage = 10
test_set_size_percentage = 10
```

In [4]:
```
# import stock prices data

df = pd.read_csv("prices-split-adjusted.csv", index_col = 0)
df.info()

# number of different stocks

print('\nnumber of different stocks: ', len(list(set(df.symbol))))
print(list(set(df.symbol))[:10])
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 851264 entries, 2016-01-05 to 2016-12-30
Data columns (total 6 columns):
symbol    851264 non-null object
open      851264 non-null float64
close     851264 non-null float64
low       851264 non-null float64
high      851264 non-null float64
volume    851264 non-null float64
dtypes: float64(5), object(1)
memory usage: 45.5+ MB

number of different stocks:  501
['BCR', 'GWW', 'MMC', 'PX', 'NWL', 'TDG', 'FLR', 'IFF', 'DTE', 'DLTR']
```

In [5]:
```
df.head()
```

Out[5]:

	symbol	open	close	low	high	volume
date						
2016-01-05	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0
2016-01-06	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0
2016-01-07	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0
2016-01-08	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0
2016-01-11	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0

In [6]:
```
df.tail()
```

Out[6]:

	symbol	open	close	low	high	volume
date						
2016-12-30	ZBH	103.309998	103.199997	102.849998	103.930000	973800.0
2016-12-30	ZION	43.070000	43.040001	42.689999	43.310001	1938100.0
2016-12-30	ZTS	53.639999	53.529999	53.270000	53.740002	1701200.0
2016-12-30	AIV	44.730000	45.450001	44.410000	45.590000	1380900.0
2016-12-30	FTV	54.200001	53.630001	53.389999	54.480000	705100.0

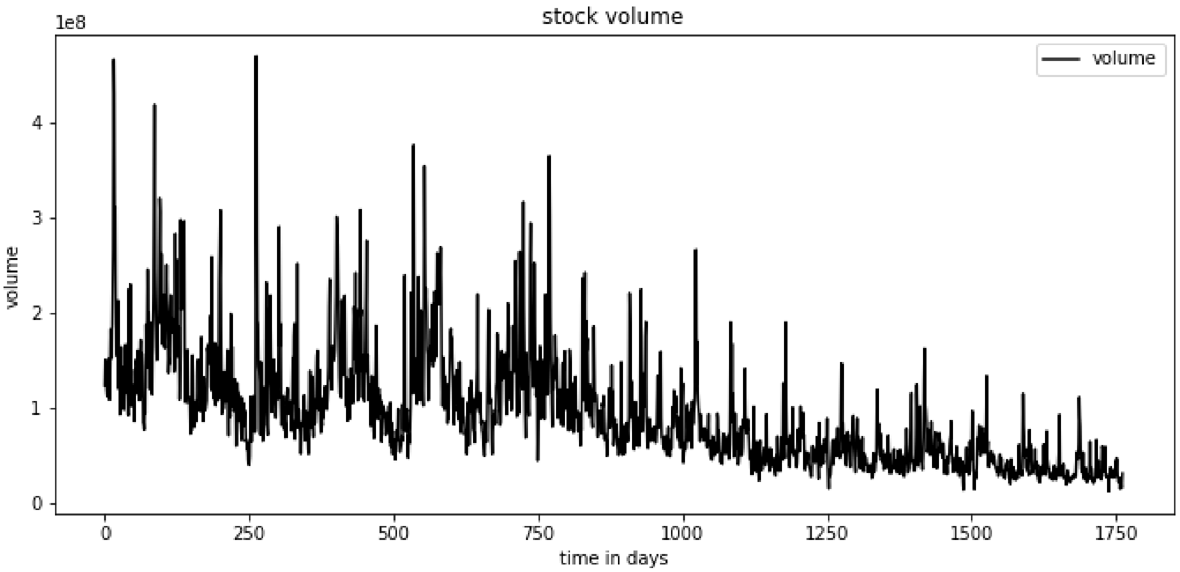
In [7]:
```
df.describe()
```

Out[7]:

	open	close	low	high	volume
count	851264.000000	851264.000000	851264.000000	851264.000000	8.512640e+05
mean	64.993618	65.011913	64.336541	65.639748	5.415113e+06
std	75.203893	75.201216	74.459518	75.906861	1.249468e+07
min	1.660000	1.590000	1.500000	1.810000	0.000000e+00
25%	31.270000	31.292776	30.940001	31.620001	1.221500e+06
50%	48.459999	48.480000	47.970001	48.959999	2.476250e+06
75%	75.120003	75.139999	74.400002	75.849998	5.222500e+06
max	1584.439941	1578.130005	1549.939941	1600.930054	8.596434e+08

In [8]:
```
#Plotting volume and price of a specific stock versus time

plt.figure(figsize=(25, 5));
plt.subplot(1,2,1);
plt.plot(df[df.symbol == 'AAPL'].open.values, color='orange', label='open')
plt.plot(df[df.symbol == 'AAPL'].close.values, color='black', label='close')
plt.plot(df[df.symbol == 'AAPL'].low.values, color='red', label='low')
plt.plot(df[df.symbol == 'AAPL'].high.values, color='green', label='high')
plt.title('stock price')
plt.xlabel('time in days')
plt.ylabel('price')
plt.legend(loc='best')
# plt.show()

plt.figure(figsize=(25, 5));
plt.subplot(1,2,2);
plt.plot(df[df.symbol == 'AAPL'].volume.values, color='black', label='volume')
plt.title('stock volume')
plt.xlabel('time in days')
plt.ylabel('volume')
plt.legend(loc='best');
plt.show()
```


stock price
price
time in days
open
close
low
high
140
120
100
80
60
40
0
250
500
750
1000
1250
1500
1750

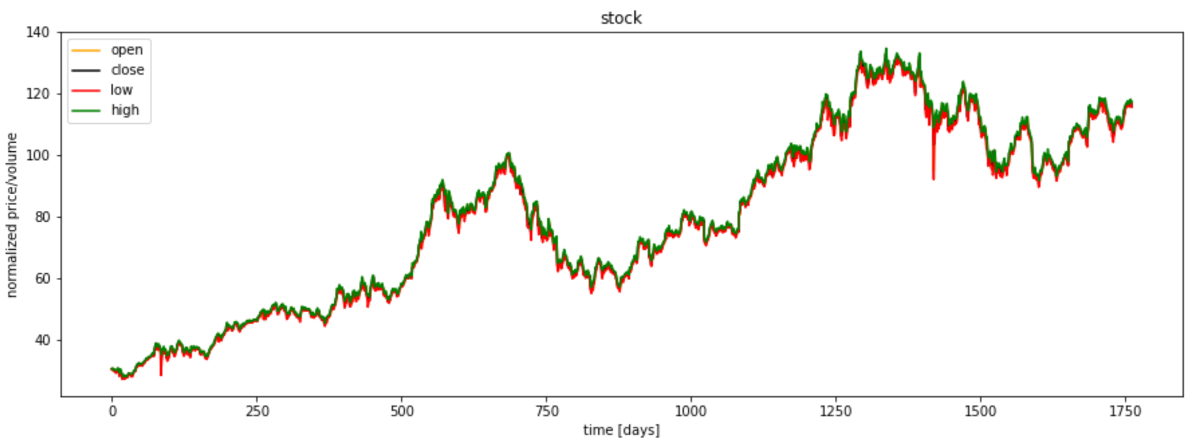

stock volume
1e8
volume
time in days
volume
4
3
2
1
0
0
250
500
750
1000
1250
1500
1750

In [9]:
```
# function for min-max normalization of stock
def normalize_data(df):
    min_max_scaler = sklearn.preprocessing.MinMaxScaler()
    df['open'] = min_max_scaler.fit_transform(df.open.values.reshape(-1,1))
    df['high'] = min_max_scaler.fit_transform(df.high.values.reshape(-1,1))
    df['low'] = min_max_scaler.fit_transform(df.low.values.reshape(-1,1))
    df['close'] = min_max_scaler.fit_transform(df['close'].values.reshape(-
1,1))
    return df

# function to create train, validation, test data given stock data and sequ
ence length
def load_data(stock, seq_len):
    raw_data = stock.as_matrix() # convert to numpy array
    data = []

    # create all possible sequences of length seq_len
    for index in range(len(raw_data) - seq_len):
        data.append(raw_data[index: index + seq_len])

    data = np.array(data);
    valid_set_size = int(np.round(valid_set_size_percentage/100*data.shape[
0]));
    test_set_size = int(np.round(test_set_size_percentage/100*data.shape[0
]));
    train_set_size = data.shape[0] - (valid_set_size + test_set_size);

    x_train = data[:train_set_size,:-1,:]
    y_train = data[:train_set_size,-1,:]

    x_valid = data[train_set_size:train_set_size+valid_set_size,:-1,:]
    y_valid = data[train_set_size:train_set_size+valid_set_size,-1,:]

    x_test = data[train_set_size+valid_set_size:,:-1,:]
    y_test = data[train_set_size+valid_set_size:,-1,:]

    return [x_train, y_train, x_valid, y_valid, x_test, y_test]

# choose one stock
df_stock = df[df.symbol == 'AAPL'].copy()
df_stock.drop(['symbol'],1,inplace=True)
df_stock.drop(['volume'],1,inplace=True)

cols = list(df_stock.columns.values)
print('df_stock.columns.values = ', cols)

# normalize stock
df_stock_norm = df_stock.copy()
df_stock_norm = normalize_data(df_stock_norm)

# create train, test data
seq_len = 20 # choose sequence length
x_train, y_train, x_valid, y_valid, x_test, y_test = load_data(df_stock_nor
m, seq_len)
print('x_train.shape = ',x_train.shape)
print('y_train.shape = ', y_train.shape)
```

```
print('x_valid.shape = ',x_valid.shape)
print('y_valid.shape = ', y_valid.shape)
print('x_test.shape = ', x_test.shape)
print('y_test.shape = ',y_test.shape)
```

```
df_stock.columns.values =  ['open', 'close', 'low', 'high']
x_train.shape =  (1394, 19, 4)
y_train.shape =  (1394, 4)
x_valid.shape =  (174, 19, 4)
y_valid.shape =  (174, 4)
x_test.shape =  (174, 19, 4)
y_test.shape =  (174, 4)
```

In [10]:
```
plt.figure(figsize=(15, 5));
plt.plot(df[df.symbol == 'AAPL'].open.values, color='orange', label='open')
plt.plot(df[df.symbol == 'AAPL'].close.values, color='black', label='close')
plt.plot(df[df.symbol == 'AAPL'].low.values, color='red', label='low')
plt.plot(df[df.symbol == 'AAPL'].high.values, color='green', label='high')

#plt.plot(df_stock_norm.volume.values, color='gray', label='volume')

plt.title('stock')
plt.xlabel('time [days]')
plt.ylabel('normalized price/volume')
plt.legend(loc='best')
plt.show()
```



In [11]:
```
# prepare batches

index_in_epoch = 0;
permutation_array  = np.arange(x_train.shape[0])
np.random.shuffle(permutation_array)

# function to get the next batch
def get_next_batch(batch_size):
    global index_in_epoch, x_train, perm_array
    start = index_in_epoch
    index_in_epoch += batch_size

    if index_in_epoch > x_train.shape[0]:
        np.random.shuffle(permutation_array) # shuffle permutation array
        start = 0 # start next epoch
        index_in_epoch = batch_size

    end = index_in_epoch
    return x_train[permutation_array[start:end]], y_train[permutation_array
[start:end]]
```

In [12]:
```
## Model Run w/ Basic RNN

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = 200
n_outputs = 4
n_layers = 2
learning_rate = 0.001
batch_size = 50
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use Basic RNN Cell
layers = [tf.contrib.rnn.BasicRNNCell(num_units=n_neurons, activation=tf.nn
.elu)
          for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})
            print('%.2f epochs: MSE train/valid = %.6f/%.6f'%(
                iteration*batch_size/train_set_size, mse_train, mse_valid))

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})
```

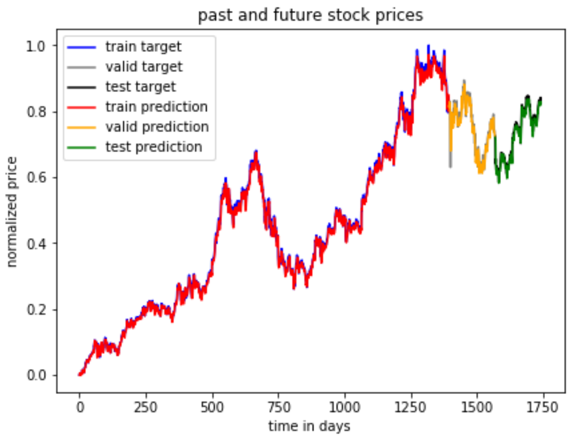
```
0.00 epochs: MSE train/valid = 0.310532/0.496241
4.99 epochs: MSE train/valid = 0.000292/0.000643
9.97 epochs: MSE train/valid = 0.000235/0.000591
14.96 epochs: MSE train/valid = 0.000214/0.000518
19.94 epochs: MSE train/valid = 0.000188/0.000484
24.93 epochs: MSE train/valid = 0.000171/0.000437
29.91 epochs: MSE train/valid = 0.000148/0.000394
34.90 epochs: MSE train/valid = 0.000145/0.000379
39.89 epochs: MSE train/valid = 0.000138/0.000359
44.87 epochs: MSE train/valid = 0.000219/0.000510
49.86 epochs: MSE train/valid = 0.000141/0.000359
54.84 epochs: MSE train/valid = 0.000136/0.000398
59.83 epochs: MSE train/valid = 0.000143/0.000400
64.81 epochs: MSE train/valid = 0.000215/0.000502
69.80 epochs: MSE train/valid = 0.000108/0.000302
74.78 epochs: MSE train/valid = 0.000207/0.000483
79.77 epochs: MSE train/valid = 0.000106/0.000294
84.76 epochs: MSE train/valid = 0.000133/0.000339
89.74 epochs: MSE train/valid = 0.000112/0.000330
94.73 epochs: MSE train/valid = 0.000100/0.000288
99.71 epochs: MSE train/valid = 0.000099/0.000273
```

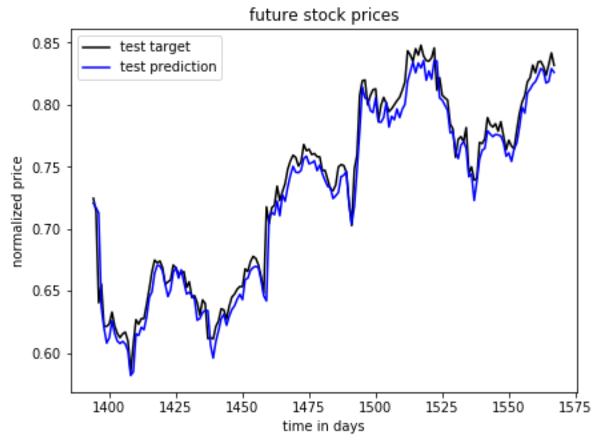
In [13]:

```
print(y_train.shape[0])
print(y_test.shape[0])
```

```
1394
174
```

In [14]:
```
var = 0 # 0 = open, 1 = close, 2 = highest, 3 = lowest

## show predictions
plt.figure(figsize=(15, 5));
plt.subplot(1,2,1);

plt.plot(np.arange(y_train.shape[0]), y_train[:,var], color='blue', label='tra
in target')

plt.plot(np.arange(y_train.shape[0], y_train.shape[0]+y_valid.shape[0]), y_val
id[:,var],
         color='gray', label='valid target')

plt.plot(np.arange(y_train.shape[0]+y_valid.shape[0],
                   y_train.shape[0]+y_test.shape[0]+y_test.shape[0]),
         y_test[:,var], color='black', label='test target')

plt.plot(np.arange(y_train_pred.shape[0]),y_train_pred[:,var], color='red',
         label='train prediction')

plt.plot(np.arange(y_train_pred.shape[0], y_train_pred.shape[0]+y_valid_pred.s
hape[0]),
         y_valid_pred[:,var], color='orange', label='valid prediction')

plt.plot(np.arange(y_train_pred.shape[0]+y_valid_pred.shape[0],
                   y_train_pred.shape[0]+y_valid_pred.shape[0]+y_test_pred.sha
pe[0]),
         y_test_pred[:,var], color='green', label='test prediction')

plt.title('past and future stock prices')
plt.xlabel('time in days')
plt.ylabel('normalized price')
plt.legend(loc='best');

plt.subplot(1,2,2);

plt.plot(np.arange(y_train.shape[0], y_train.shape[0]+y_test.shape[0]),
         y_test[:,var], color='black', label='test target')

plt.plot(np.arange(y_train_pred.shape[0], y_train_pred.shape[0]+y_test_pred.sh
ape[0]),
         y_test_pred[:,var], color='blue', label='test prediction')

plt.title('future stock prices')
plt.xlabel('time in days')
plt.ylabel('normalized price')
plt.legend(loc='best');

plt.show()
```





In [15]:

```
# classify into binary classes based on Close-Open predictions (gainers= +, lo
ssers= -)

corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train[:,
0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_tra
in.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid[:,
0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_val
id.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,0
]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_test.
shape[0]


print('correct classifiation of close - open price for train/valid/test: %.2f/
%.2f/%.2f'%(
    corr_price_development_train, corr_price_development_valid, corr_price_dev
elopment_test))
```

```
correct classifiation of close - open price for train/valid/test: 0.74/0.78/
0.89
```

In [16]:
```
## Model Run w/ Gated RNN, LSTM

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = 200
n_outputs = 4
n_layers = 2
learning_rate = 0.001
batch_size = 50
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})
            print('%.2f epochs: MSE train/valid = %.6f/%.6f'%(
                iteration*batch_size/train_set_size, mse_train, mse_valid))

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
```

```
 lossers= -)

corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]


print('correct classifiation of close - open price for train/valid/test: %.
2f/%.2f/%.2f'%(
    corr_price_development_train, corr_price_development_valid, corr_price_
development_test))
```

```
0.00 epochs: MSE train/valid = 0.140582/0.334605
4.99 epochs: MSE train/valid = 0.000600/0.001285
9.97 epochs: MSE train/valid = 0.000542/0.001170
14.96 epochs: MSE train/valid = 0.000733/0.001340
19.94 epochs: MSE train/valid = 0.000740/0.001439
24.93 epochs: MSE train/valid = 0.000494/0.001082
29.91 epochs: MSE train/valid = 0.000350/0.000693
34.90 epochs: MSE train/valid = 0.000302/0.000630
39.89 epochs: MSE train/valid = 0.000361/0.000787
44.87 epochs: MSE train/valid = 0.000282/0.000616
49.86 epochs: MSE train/valid = 0.000282/0.000699
54.84 epochs: MSE train/valid = 0.000358/0.000937
59.83 epochs: MSE train/valid = 0.000310/0.000777
64.81 epochs: MSE train/valid = 0.000180/0.000419
69.80 epochs: MSE train/valid = 0.000216/0.000461
74.78 epochs: MSE train/valid = 0.000226/0.000474
79.77 epochs: MSE train/valid = 0.000150/0.000325
84.76 epochs: MSE train/valid = 0.000146/0.000322
89.74 epochs: MSE train/valid = 0.000168/0.000341
94.73 epochs: MSE train/valid = 0.000120/0.000292
99.71 epochs: MSE train/valid = 0.000118/0.000291
correct classifiation of close - open price for train/valid/test: 0.75/0.78/
0.89
```

In [17]:
```
# Optimize Hyper-paramters of Model
```

In [18]:
```
number_neurons=[]
corr_price_development_train_set=[]
corr_price_development_valid_set=[]
corr_price_development_test_set=[]
```

In [19]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Neurons (1)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = 1
n_outputs = 4
n_layers = 2
learning_rate = 0.001
batch_size = 50
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

number_neurons.append(n_neurons)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [20]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Neurons (10)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = 10
n_outputs = 4
n_layers = 2
learning_rate = 0.001
batch_size = 50
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.nn.e
lu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean squared e
rror
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trainin
g batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +, lo
ssers= -)

corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train[:,
```

```
0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_tra
in.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid[:,
0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_val
id.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,0
]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_test.
shape[0]

# append results

number_neurons.append(n_neurons)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [21]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Neurons (100)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = 100
n_outputs = 4
n_layers = 2
learning_rate = 0.001
batch_size = 50
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

number_neurons.append(n_neurons)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [22]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Neurons (200)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = 200
n_outputs = 4
n_layers = 2
learning_rate = 0.001
batch_size = 50
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.nn.e
lu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean squared e
rror
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trainin
g batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +, lo
ssers= -)

corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train[:,
```

```
0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_tra
in.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid[:,
0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_val
id.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,0
]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_test.
shape[0]

# append results

number_neurons.append(n_neurons)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [23]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Neurons (500)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = 500
n_outputs = 4
n_layers = 2
learning_rate = 0.001
batch_size = 50
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

number_neurons.append(n_neurons)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

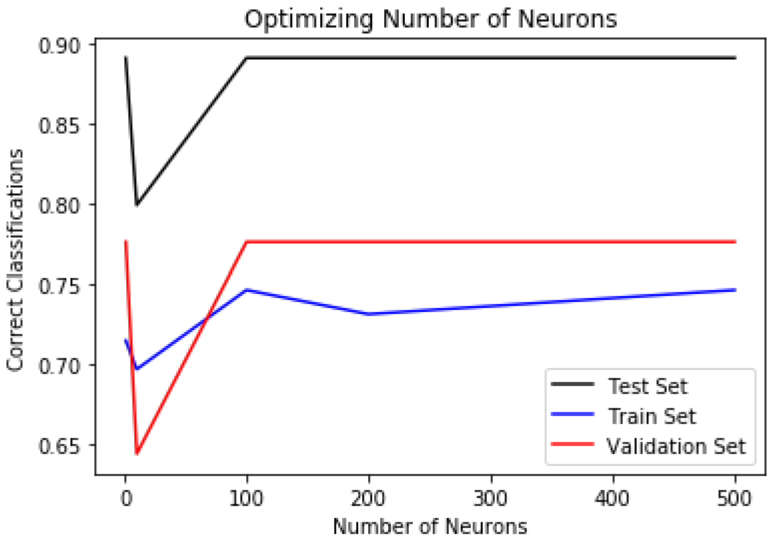
In [24]:
```
print(corr_price_development_test_set)
```

```
[0.89080459770114939, 0.79885057471264365, 0.89080459770114939, 0.89080459770
114939, 0.89080459770114939]
```

In [25]:
```
print(number_neurons)
```

```
[1, 10, 100, 200, 500]
```

In [26]:
```
# evaluate results

plt.plot(number_neurons,corr_price_development_test_set,color='black', label=
'Test Set')
plt.plot(number_neurons,corr_price_development_train_set,color='blue', label=
'Train Set')
plt.plot(number_neurons,corr_price_development_valid_set,color='red', label='V
alidation Set')
plt.legend()
plt.title('Optimizing Number of Neurons')
plt.xlabel('Number of Neurons')
plt.ylabel('Correct Classifications')

plt.show()

selected_value_neurons = max(corr_price_development_test_set)
selected_index = corr_price_development_test_set.index(selected_value_neurons)
selected_neurons = number_neurons[selected_index]

print('Optimal Number of Neurons: %1f'%(selected_neurons))
print('Highest correct classifiation of close - open price for test set: %.15f
'%(selected_value_neurons))
```




```
Optimal Number of Neurons: 1.000000
Highest correct classifiation of close - open price for test set: 0.890804597
701149
```

In [27]:
```
number_layers=[]
corr_price_development_train_set=[]
corr_price_development_valid_set=[]
corr_price_development_test_set=[]
```

In [28]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Layers (1)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = 1
learning_rate = 0.001
batch_size = 50
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0]))).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0]))).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0]))).astype(int)) / y_te
st.shape[0]

# append results

number_layers.append(n_layers)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [29]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Layers (2)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = 2
learning_rate = 0.001
batch_size = 50
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

number_layers.append(n_layers)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [30]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Layers (3)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = 3
learning_rate = 0.001
batch_size = 50
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0]))).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0]))).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0]))).astype(int)) / y_te
st.shape[0]

# append results

number_layers.append(n_layers)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [31]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Layers (4)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = 4
learning_rate = 0.001
batch_size = 50
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

number_layers.append(n_layers)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [32]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Layers (5)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = 5
learning_rate = 0.001
batch_size = 50
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

number_layers.append(n_layers)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

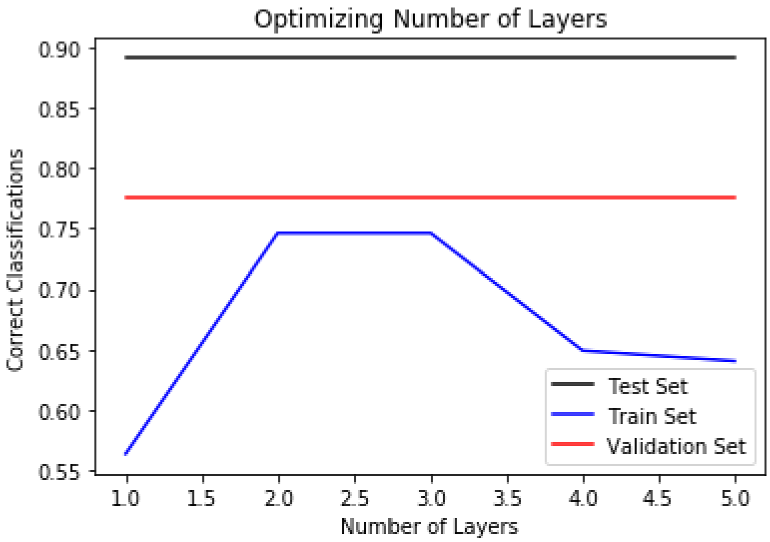
In [33]:
```
print(corr_price_development_test_set)
```

```
[0.89080459770114939, 0.89080459770114939, 0.89080459770114939, 0.89080459770
114939, 0.89080459770114939]
```

In [34]:
```
print(number_layers)
```

```
[1, 2, 3, 4, 5]
```

In [35]:
```
# evaluate results

plt.plot(number_layers,corr_price_development_test_set,color='black', label='T
est Set')
plt.plot(number_layers,corr_price_development_train_set,color='blue', label='T
rain Set')
plt.plot(number_layers,corr_price_development_valid_set,color='red', label='Va
lidation Set')
plt.legend()
plt.title('Optimizing Number of Layers')
plt.xlabel('Number of Layers')
plt.ylabel('Correct Classifications')

plt.show()

selected_value_layers = max(corr_price_development_test_set)
selected_index = corr_price_development_test_set.index(selected_value_layers)
selected_layers = number_layers[selected_index]

print('Optimal Number of Layers: %1f'%(selected_layers))
print('Highest correct classifiation of close - open price for test set: %.15f
'%(selected_value_layers))
```




```
Optimal Number of Layers: 1.000000
Highest correct classifiation of close - open price for test set: 0.890804597
701149
```

In [36]:
```
batch=[]
corr_price_development_train_set=[]
corr_price_development_valid_set=[]
corr_price_development_test_set=[]
```

In [37]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Batch Sizes (20)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.001
batch_size = 20
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

batch.append(batch_size)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [38]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Batch Sizes (30)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.001
batch_size = 30
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

batch.append(batch_size)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [39]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Batch Sizes (40)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.001
batch_size = 40
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

batch.append(batch_size)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [40]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Batch Sizes (50)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.001
batch_size = 50
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0]))).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0]))).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0]))).astype(int)) / y_te
st.shape[0]

# append results

batch.append(batch_size)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [41]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Batch Sizes (60)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.001
batch_size = 60
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

batch.append(batch_size)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

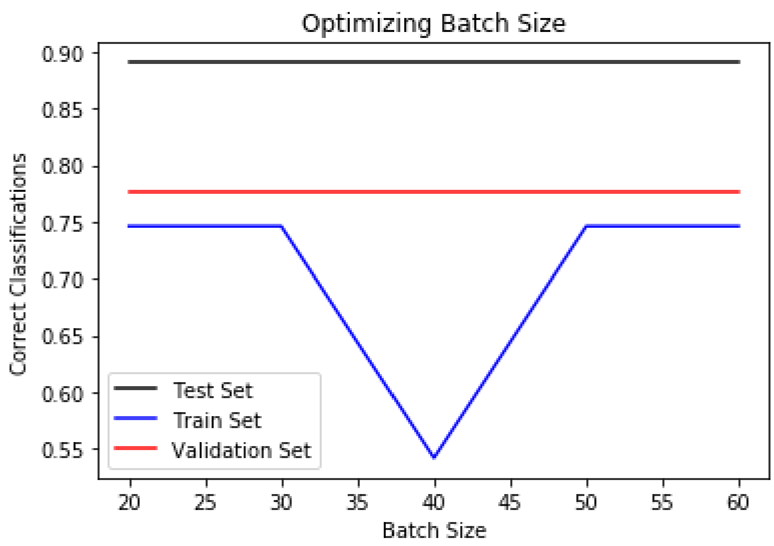
In [42]:
```
print(corr_price_development_test_set)
```

```
[0.89080459770114939, 0.89080459770114939, 0.89080459770114939, 0.89080459770
114939, 0.89080459770114939]
```

In [43]:
```
print(batch)
```

```
[20, 30, 40, 50, 60]
```

In [44]:
```
# evaluate results

plt.plot(batch,corr_price_development_test_set,color='black', label='Test Set'
)
plt.plot(batch,corr_price_development_train_set,color='blue', label='Train Se
t')
plt.plot(batch,corr_price_development_valid_set,color='red', label='Validation
 Set')
plt.legend()
plt.title('Optimizing Batch Size')
plt.xlabel('Batch Size')
plt.ylabel('Correct Classifications')

plt.show()

selected_value_batch = max(corr_price_development_test_set)
selected_index = corr_price_development_test_set.index(selected_value_batch)
selected_batch = batch[selected_index]

print('Optimal Number of Layers: %1f'%(selected_batch))
print('Highest correct classifiation of close - open price for test set: %.15f
'%(selected_value_batch))
```




```
Optimal Number of Layers: 20.000000
Highest correct classifiation of close - open price for test set: 0.890804597
701149
```

In [45]:
```
epochs=[]
corr_price_development_train_set=[]
corr_price_development_valid_set=[]
corr_price_development_test_set=[]
```

In [46]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Epochs (20)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.001
batch_size = selected_batch
n_epochs = 20
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

epochs.append(n_epochs)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [47]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Epochs (30)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.001
batch_size = selected_batch
n_epochs = 30
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0]))).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0]))).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0]))).astype(int)) / y_te
st.shape[0]

# append results

epochs.append(n_epochs)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [48]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Epochs (70)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.001
batch_size = selected_batch
n_epochs = 70
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0]))).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0]))).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0]))).astype(int)) / y_te
st.shape[0]

# append results

epochs.append(n_epochs)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [49]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Epochs (100)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.001
batch_size = selected_batch
n_epochs = 100
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0]))).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0]))).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0]))).astype(int)) / y_te
st.shape[0]

# append results

epochs.append(n_epochs)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [50]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Number of Epochs (200)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.001
batch_size = selected_batch
n_epochs = 200
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

epochs.append(n_epochs)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

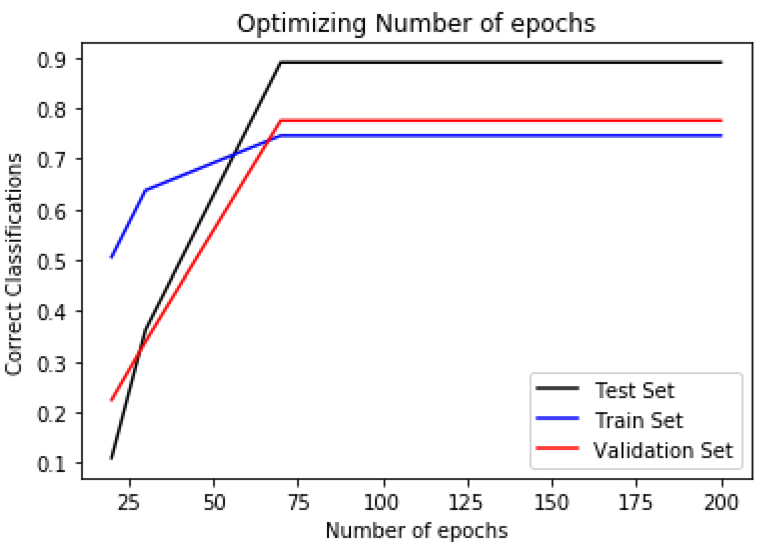
In [51]:
```
print(corr_price_development_test_set)
```

```
[0.10919540229885058, 0.36206896551724138, 0.89080459770114939, 0.89080459770
114939, 0.89080459770114939]
```

In [52]:
```
print(epochs)
```

```
[20, 30, 70, 100, 200]
```

In [53]:
```
# evaluate results

plt.plot(epochs,corr_price_development_test_set,color='black', label='Test Se
t')
plt.plot(epochs,corr_price_development_train_set,color='blue', label='Train Se
t')
plt.plot(epochs,corr_price_development_valid_set,color='red', label='Validatio
n Set')
plt.legend()
plt.title('Optimizing Number of epochs')
plt.xlabel('Number of epochs')
plt.ylabel('Correct Classifications')

plt.show()

selected_value_epoch = max(corr_price_development_test_set)
selected_index = corr_price_development_test_set.index(selected_value_epoch)
selected_epoch = epochs[selected_index]

print('Optimal Number of Epochs: %1f'%(selected_epoch))
print('Highest correct classifiation of close - open price for test set: %.15f
'%(selected_value_epoch))
```




```
Optimal Number of Epochs: 70.000000
Highest correct classifiation of close - open price for test set: 0.890804597
701149
```

In [54]:
```
learn_rate=[]
corr_price_development_train_set=[]
corr_price_development_valid_set=[]
corr_price_development_test_set=[]
```

In [55]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Learning Rates (0.001)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.001
batch_size = selected_batch
n_epochs = selected_epoch
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0]))).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0]))).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0]))).astype(int)) / y_te
st.shape[0]

# append results

learn_rate.append(learning_rate)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [56]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Learning Rates (0.01)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.01
batch_size = selected_batch
n_epochs = selected_epoch
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

learn_rate.append(learning_rate)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [57]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Learning Rates (0.05)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.05
batch_size = selected_batch
n_epochs = selected_epoch
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

learn_rate.append(learning_rate)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [58]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Learning Rates (0.1)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.1
batch_size = selected_batch
n_epochs = selected_epoch
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

learn_rate.append(learning_rate)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [59]:
```
## Model Run w/ Gated RNN, LSTM w/ Varying Learning Rates (0.5)

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = 0.5
batch_size = selected_batch
n_epochs = selected_epoch
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

# append results

learn_rate.append(learning_rate)
corr_price_development_train_set.append(corr_price_development_train)
corr_price_development_valid_set.append(corr_price_development_valid)
corr_price_development_test_set.append(corr_price_development_test)
```

In [60]:
```
print(corr_price_development_test_set)
```

```
[0.89080459770114939, 0.89080459770114939, 0.89080459770114939, 0.89080459770
114939, 0.10919540229885058]
```

In [61]:
```
print(learn_rate)
```

```
[0.001, 0.01, 0.05, 0.1, 0.5]
```

In [66]:
```
# evaluate results

plt.plot(learn_rate,corr_price_development_test_set,color='black', label='Test
 Set')
plt.plot(learn_rate,corr_price_development_train_set,color='blue', label='Trai
n Set')
plt.plot(learn_rate,corr_price_development_valid_set,color='red', label='Valid
ation Set')
plt.legend()
plt.title('Optimizing Learning Rate')
plt.xlabel('Learning Rate')
plt.ylabel('Correct Classifications')

plt.show()

selected_value_learn_rate = max(corr_price_development_test_set)
selected_index = corr_price_development_test_set.index(selected_value_learn_ra
te)
selected_learn_rate = learn_rate[selected_index]

print('Optimal Learning Rate: %1f'%(selected_learn_rate))
print('Highest correct classifiation of close - open price for test set: %.15f
'%(selected_value_learn_rate))
```


```
Optimal Learning Rate: 0.001000
Highest correct classifiation of close - open price for test set: 0.890804597
701149
```

In [63]:
```
# Optimized Model for Given Stock
```

In [64]:
```
## Model Run w/ Gated RNN, LSTM  - Optimized Hyper Parameters

# parameters
n_steps = seq_len-1
n_inputs = 4
n_neurons = selected_neurons
n_outputs = 4
n_layers = selected_layers
learning_rate = selected_learn_rate
batch_size = selected_batch
n_epochs = selected_epoch
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]

tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# use LSTM Cell
layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.elu)
         for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float
32)

stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:,n_steps-1,:] # keep only last output of sequence

loss = tf.reduce_mean(tf.square(outputs - y)) # loss function = mean square
d error
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

# run model
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(int(n_epochs*train_set_size/batch_size)):
        x_batch, y_batch = get_next_batch(batch_size) # fetch the next trai
ning batch
        sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
        if iteration % int(5*train_set_size/batch_size) == 0:
            mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
            mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})

    y_train_pred = sess.run(outputs, feed_dict={X: x_train})
    y_valid_pred = sess.run(outputs, feed_dict={X: x_valid})
    y_test_pred = sess.run(outputs, feed_dict={X: x_test})

# classify into binary classes based on Close-Open predictions (gainers= +,
 lossers= -)
```

```
corr_price_development_train = np.sum(np.equal(np.sign(y_train[:,1]-y_train
[:,0]),
            np.sign(y_train_pred[:,1]-y_train_pred[:,0])).astype(int)) / y_
train.shape[0]

corr_price_development_valid = np.sum(np.equal(np.sign(y_valid[:,1]-y_valid
[:,0]),
            np.sign(y_valid_pred[:,1]-y_valid_pred[:,0])).astype(int)) / y_
valid.shape[0]

corr_price_development_test = np.sum(np.equal(np.sign(y_test[:,1]-y_test[:,
0]),
            np.sign(y_test_pred[:,1]-y_test_pred[:,0])).astype(int)) / y_te
st.shape[0]

print('Final Model Classification Accuracy: %.15f'%(corr_price_development_
test))
```

```
Final Model Classification Accuracy: 0.890804597701149
```