Why Bitcoin prices can't be predicted using Deep Learning.

April 27, 2018

In [103]:
```
import json
import requests

from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout, LSTM
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.metrics import mean_absolute_error

sns.set_palette('Set2')
%matplotlib inline
```

In [104]:
```
endpoint = 'https://min-api.cryptocompare.com/data/histoday'
res = requests.get(endpoint + '?fsym=BTC&tsym=USD&limit=2000')
hist = pd.DataFrame(json.loads(res.content)['Data'])
hist = hist.set_index('time')
hist.index = pd.to_datetime(hist.index, unit='s')
```

In [188]:
```
hist.head()
```

Out[188]:

time	close	high	low	open	volumefrom	volumeto
2012-11-02	10.47	10.80	10.33	10.57	24485.26	258957.79
2012-11-03	10.64	10.65	10.40	10.47	16732.94	176345.08
2012-11-04	10.80	10.90	10.51	10.64	16750.34	178761.66
2012-11-05	10.75	10.88	10.61	10.80	21776.74	233650.36
2012-11-06	10.90	10.90	10.67	10.75	26995.30	291515.94

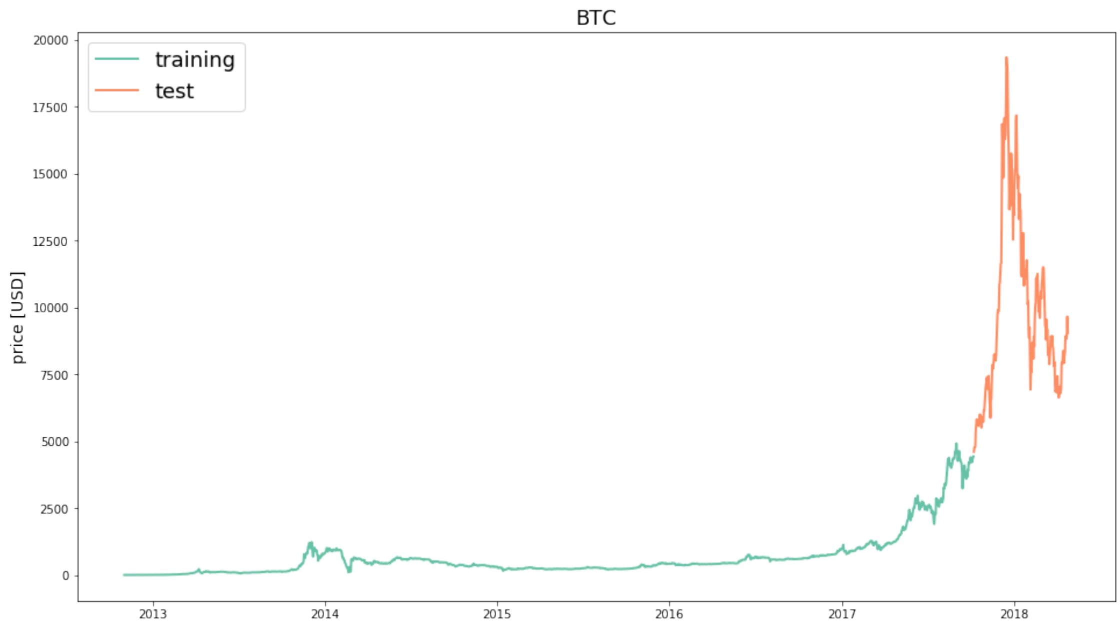
In [107]:
```
target_col = 'close'
```

In [186]:
```
def train_test_split(df, test_size=0.1):
    split_row = int(len(df) - (test_size * len(df)))
    train_data = df.iloc[:split_row]
    test_data = df.iloc[split_row:]
    return train_data, test_data
```

In [108]:
```
train, test = train_test_split(hist, test_size=0.1)
```

```
In [109]: def line_plot(line1, line2, label1=None, label2=None, title='', lw=2):
              fig, ax = plt.subplots(1, figsize=(16, 9))
              ax.plot(line1, label=label1, linewidth=lw)
              ax.plot(line2, label=label2, linewidth=lw)
              ax.set_ylabel('price [USD]', fontsize=14)
              ax.set_title(title, fontsize=18)
              ax.legend(loc='best', fontsize=18);
```

```
In [110]: line_plot(train[target_col], test[target_col], 'training', 'test', title='BTC')
```




```
In [111]: def normalise_zero_base(df):
              """ Normalise dataframe column-wise to reflect changes with respect to first ent
              return df / df.iloc[0] - 1

          def normalise_min_max(df):
              """ Normalise dataframe column-wise min/max. """
              return (df - df.min()) / (data.max() - df.min())
```

```
In [112]: def extract_window_data(df, window_len=10, zero_base=True):
              """ Convert dataframe to overlapping sequences/windows of len `window_data`.

                  :param window_len: Size of window
                  :param zero_base: If True, the data in each window is normalised to reflect
                      with respect to the first entry in the window (which is then always 0)
              """
              window_data = []
              for idx in range(len(df) - window_len):
```

```
        tmp = df[idx: (idx + window_len)].copy()
        if zero_base:
            tmp = normalise_zero_base(tmp)
        window_data.append(tmp.values)
    return np.array(window_data)
```

```
In [113]: def prepare_data(df, target_col, window_len=10, zero_base=True, test_size=0.2):
              """ Prepare data for LSTM. """
              # train test split
              train_data, test_data = train_test_split(df, test_size=test_size)

              # extract window data
              X_train = extract_window_data(train_data, window_len, zero_base)
              X_test = extract_window_data(test_data, window_len, zero_base)

              # extract targets
              y_train = train_data[target_col][window_len:].values
              y_test = test_data[target_col][window_len:].values
              if zero_base:
                  y_train = y_train / train_data[target_col][:-window_len].values - 1
                  y_test = y_test / test_data[target_col][:-window_len].values - 1

              return train_data, test_data, X_train, X_test, y_train, y_test
```

```
In [57]: def prepare_data(df, target_col, window_len=10, zero_base=True, test_size=0.2):
             """ Prepare data for LSTM. """
             # train test split
             train_data, test_data = train_test_split(df, test_size=test_size)

             # extract window data
             X_train = extract_window_data(train_data, window_len, zero_base)
             X_test = extract_window_data(test_data, window_len, zero_base)

             # extract targets
             y_train = train_data[target_col][window_len:].values
             y_test = test_data[target_col][window_len:].values
             if zero_base:
                 y_train = y_train / train_data[target_col][:-window_len].values - 1
                 y_test = y_test / test_data[target_col][:-window_len].values - 1

             return train_data, test_data, X_train, X_test, y_train, y_test
```

0.0.1 A multi layer LSTM model is designed with activation function Tanh and Optimizer Adam

```
In [171]: def build_lstm_model(input_data, output_size, neurons=20, activ_func='tanh',
                               dropout=0.25, loss='mae', optimizer='adam'):
              model = Sequential()
```

```
        model.add(LSTM(neurons, input_shape=(input_data.shape[1], input_data.shape[2]), r
        model.add(LSTM((1),return_sequences = False))
        model.add(Dropout(dropout))
        model.add(Dense(units=output_size))
        model.add(Activation(activ_func))

        model.compile(loss=loss, optimizer=optimizer)
        return model
```

In [162]:
```
np.random.seed(42)

# data params
window_len = 7
test_size = 0.1
zero_base = True

# model params
lstm_neurons = 20
epochs = 50
batch_size = 4
loss = 'mae'
dropout = 0.25
optimizer = 'adam'
```

In [163]:
```
train, test, X_train, X_test, y_train, y_test = prepare_data(
    hist, target_col, window_len=window_len, zero_base=zero_base, test_size=test_size
```

In [126]:
```
X_train.shape
```

Out[126]:
```
(1794, 7, 6)
```

In [172]:
```
model = build_lstm_model(
    X_train, output_size=1, neurons=lstm_neurons, dropout=dropout, loss=loss,
    optimizer=optimizer)
history = model.fit(
    X_train, y_train, epochs=epochs, batch_size=batch_size, verbose=1, shuffle=True)
```

```
Epoch 1/50
1794/1794 [==============================] - 8s 4ms/step - loss: 0.0834
Epoch 2/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0693
Epoch 3/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0632
Epoch 4/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0590
Epoch 5/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0580
Epoch 6/50
```

```
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0563
Epoch 7/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0579
Epoch 8/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0574
Epoch 9/50
1794/1794 [==============================] - 6s 3ms/step - loss: 0.0556
Epoch 10/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0561
Epoch 11/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0558
Epoch 12/50
1794/1794 [==============================] - 6s 3ms/step - loss: 0.0540
Epoch 13/50
1794/1794 [==============================] - 6s 3ms/step - loss: 0.0579
Epoch 14/50
1794/1794 [==============================] - 6s 3ms/step - loss: 0.0562
Epoch 15/50
1794/1794 [==============================] - 6s 3ms/step - loss: 0.0558
Epoch 16/50
1794/1794 [==============================] - 6s 3ms/step - loss: 0.0549
Epoch 17/50
1794/1794 [==============================] - 6s 3ms/step - loss: 0.0558
Epoch 18/50
1794/1794 [==============================] - 6s 3ms/step - loss: 0.0540
Epoch 19/50
1794/1794 [==============================] - 6s 3ms/step - loss: 0.0521
Epoch 20/50
1794/1794 [==============================] - 6s 3ms/step - loss: 0.0521
Epoch 21/50
1794/1794 [==============================] - 6s 3ms/step - loss: 0.0560
Epoch 22/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0562
Epoch 23/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0537
Epoch 24/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0525
Epoch 25/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0533
Epoch 26/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0548
Epoch 27/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0545
Epoch 28/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0521
Epoch 29/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0544
Epoch 30/50
```

```
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0527
Epoch 31/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0535
Epoch 32/50
1794/1794 [==============================] - 6s 3ms/step - loss: 0.0539
Epoch 33/50
1794/1794 [==============================] - 6s 3ms/step - loss: 0.0533
Epoch 34/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0535
Epoch 35/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0548
Epoch 36/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0516
Epoch 37/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0502
Epoch 38/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0519
Epoch 39/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0531
Epoch 40/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0534
Epoch 41/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0517
Epoch 42/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0566
Epoch 43/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0525
Epoch 44/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0527
Epoch 45/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0531
Epoch 46/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0529
Epoch 47/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0521
Epoch 48/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0534
Epoch 49/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0516
Epoch 50/50
1794/1794 [==============================] - 5s 3ms/step - loss: 0.0555
```

```
In [177]: targets = test[target_col][window_len:]
          preds = model.predict(X_test).squeeze()
```

```
In [178]: mean_absolute_error(preds, y_test)
```

```
Out[178]: 0.057658119546962219
```

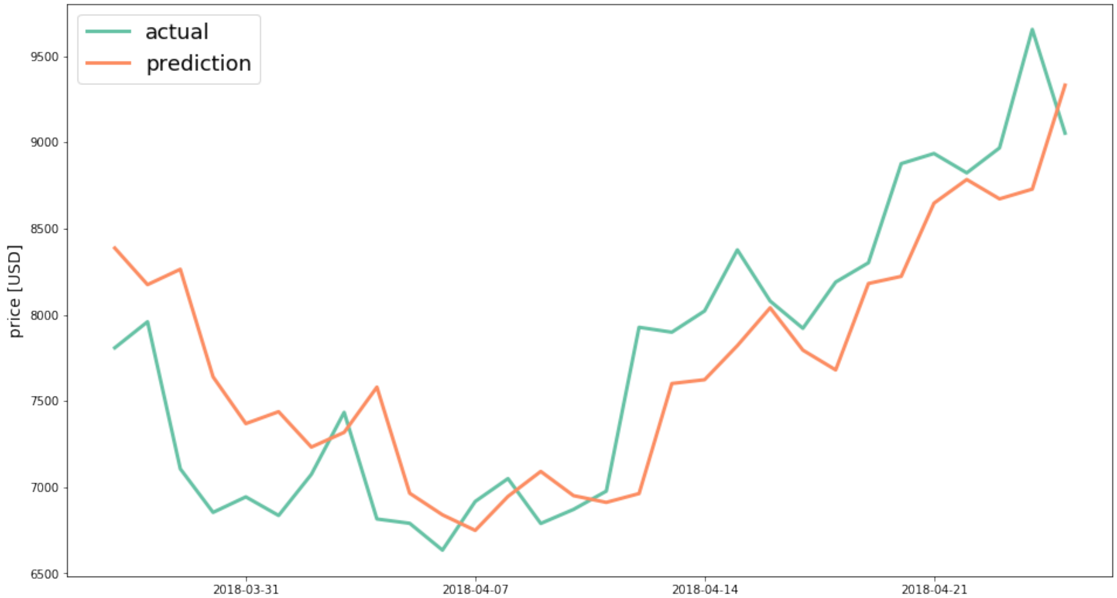
In [179]:
```
preds = test[target_col].values[:-window_len] * (preds + 1)
preds = pd.Series(index=targets.index, data=preds)

line_plot(targets, preds, 'actual', 'prediction', lw=3)
```

0.0.2 To have a better understanding of these results, let us have a closer look at the actual and predicted values of Bitcoin. For that, let us consider taking last 30 days values.

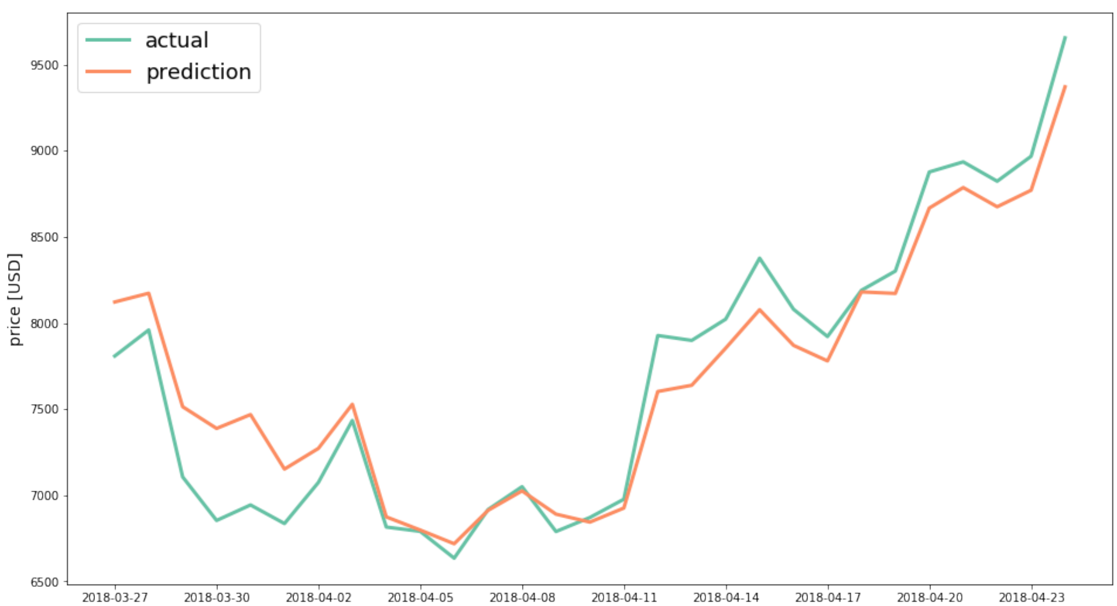
In [180]:
```
n_points = 30

line_plot(targets[-n_points:], preds[-n_points:], 'actual', 'prediction', lw=3)
```



0.0.3 If you have seen it carefully, the predicted values are just the predictions that are made by considering the previous day's value. The value may not be fitting exactly the same, but our LSTM model is performing well in replicating the trend. Shifting the predicted values 1 day earlier would help us understand the inference better.

```
In [160]: line_plot(targets[-n_points:][:-1], preds[-n_points:].shift(-1), 'actual', 'predicti
```



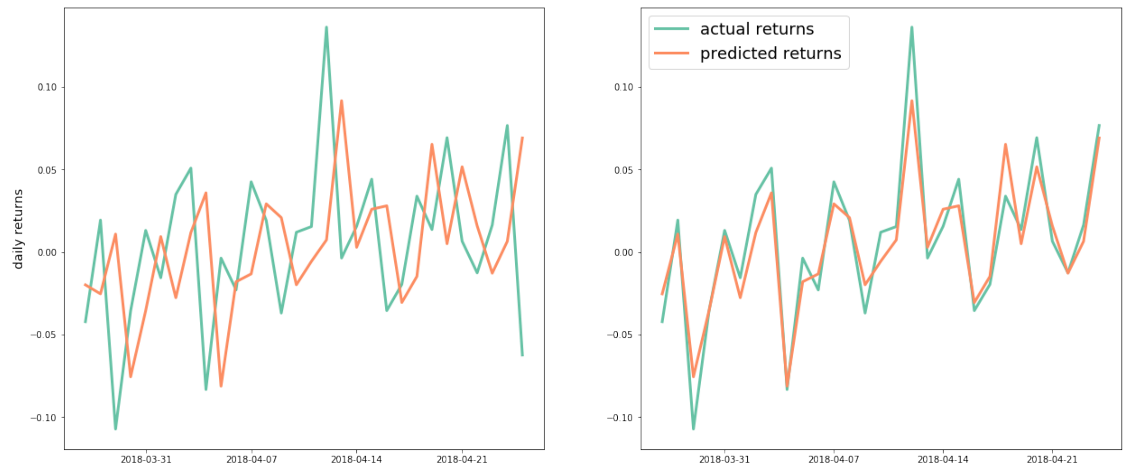
0.0.4 This is happening because whenever a new value is predicted by the model, it basically takes the bucket of previous 10 values for training. This means when we go to predict the next value, we have the previous 10 values in our training, i.e., more importantly, the previous day's value is there and the model is majorly referencing from that value to predict the current price. That's why the predicted value of today is so close to the previous day's actual value.

0.0.5 More the accuracy of the above plot it, we can infer that more is the model capability of replicating the previous trend. But we cannot rely on this for actual trading as the actual correlation between the predicted and actual value would be very less.

```
In [181]: actual_returns = targets.pct_change()[1:]
          predicted_returns = preds.pct_change()[1:]
```

```
In [182]: def dual_line_plot(line1, line2, line3, line4, label1=None, label2=None, title='', l
              import matplotlib.dates as mdates
              fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(21, 9))
              ax1.plot(line1, label=label1, linewidth=lw)
              ax1.plot(line2, label=label2, linewidth=lw)
              ax2.plot(line3, label=label1, linewidth=lw)
              ax2.plot(line4, label=label2, linewidth=lw)
              ax2.set_xticks(ax1.get_xticks())
              ax2.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
              ax1.set_ylabel('daily returns', fontsize=14)
              ax2.legend(loc='best', fontsize=18);
```

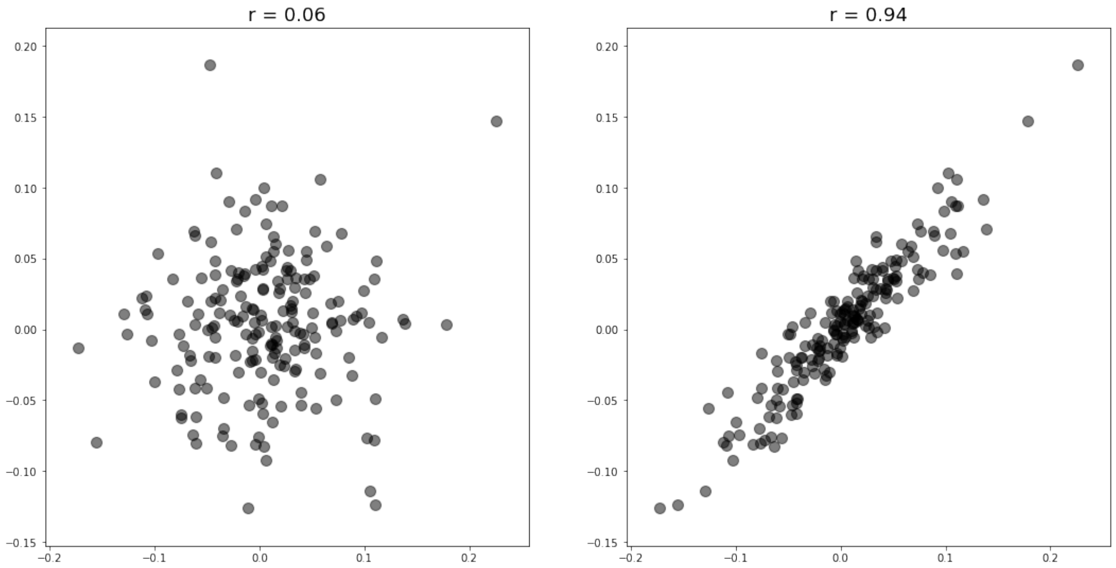
```
In [183]: dual_line_plot(actual_returns[-n_points:],
                         predicted_returns[-n_points:],
                         actual_returns[-n_points:][:-1],
                         predicted_returns[-n_points:].shift(-1),
                         'actual returns', 'predicted returns', lw=3)
```



In [185]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 9))

# actual correlation
corr = np.corrcoef(actual_returns, predicted_returns)[0][1]
ax1.scatter(actual_returns, predicted_returns, color='k', marker='o', alpha=0.5, s=1(
ax1.set_title('r = {:.2f}'.format(corr), fontsize=18)

# shifted correlation
shifted_actual = actual_returns[:-1]
shifted_predicted = predicted_returns.shift(-1).dropna()
corr = np.corrcoef(shifted_actual, shifted_predicted)[0][1]
ax2.scatter(shifted_actual, shifted_predicted, color='k', marker='o', alpha=0.5, s=1(
ax2.set_title('r = {:.2f}'.format(corr), fontsize=18);
```



0.0.6 Look at the correlation difference between the actual and predicted value to that of predicted value and previous day's actual value. The left plot displays the inefficiency of model to predict new price where as the right plot displays the model capacity to replicate the previous day's trend.