Machine learning for market trend prediction in Bitcoin

Anique Akhtar

Abstract

Bitcoin is the world's leading cryptocurrency, allowing users to make transactions securely and anonymously over the internet. In this paper, I tried to predict the future price of bitcoin in a shorter period. I implemented a lot of trading indicators and technical analysis techniques used in financial stocks followed by machine learning techniques to learn from these indicators and predict the future price of bitcoin. I used convolutional Neural Network (CNN) with technical indicators to obtain unto 90% accuracy.

Introduction

Bitcoin is a decentralized digital currency that uses cryptographic protocol. It is not bound or backed by any government and works on a peer-to-peer system. Bitcoin is the worlds most valuable cryptocurrency introduced following the release of a white-paper published in 2008 under the pseudo name Satoshi Nakamoto [1]. Bitcoin is very different from traditional financial markets. It operates on a decentralized, peer-to-peer and trustless system in which all transactions are posted to an open ledger called the Blockchain. This type of transparency is unheard of in other financial markets. Recently, there has been a lot of other crypto-currency (alt-coins) introduced to compliment bitcoin and what it's trying to achieve. The crypto-currency has a market cap of 305 billion USD with Bitcoin dominating the market with 54% of the market cap [2].

Bitcoin is highly volatile and unlike traditional markets, its price is tougher to predict. Bitcoin also faces a lot of market manipulation issues and the social trading and wisdom of the crowd plays a big role. However, Bitcoin still follows the general financial trend analysis just on a very fast rate since it is seeing an exponential growth. In this paper we implement a short term future price prediction for Bitcoin. We gather the price data of Bitcoin and apply a lot of financial technical indicators to the price data to get a lot of information. We use this information to train a Convolutional Neural Network (CNN) to predict the short-term future price of Bitcoin. Every Bitcoin trader prefers their own technical indicators when trading in Bitcoin. We thought it would be interesting to train a CNN using these indicators so that the CNN can extract the relevant information from

each of the technical indicators in order to perform an accurate estimate of the future price pattern.

Dataset

Bitcoin is being traded in a lot of online exchanges at the moment with the biggest being BitFinex, Bitthumb, GDAX, Poloniex, and Bitstamp. I collected data from Bitstamp since it's one of the older exchanges and haven't had any hacking or market manipulation complaints. I collected the data from 2012-01-01_to_2017-10-20 for Bitstamp available via their API [3]. For my training and analysis, I only used the last 1 year of that data. Since I wanted to make a short term prediction, I used the last one hour of data and performed technical analysis for the last one hour to train the convolutional neural network. The CNN would then predict the 20 minutes future price of bitcoin.

I collected the following per minute data from this datasets:

- Opening Price.
- Closing Price.
- High.
- Low.
- Volume.

Feature Engineering and Technical Analysis

Technical analysis of stocks is in some ways very similar to feature engineering. Technical analysis and technical indicators are used by traders and stock market experts to predict the future price. Every trader prefers their own technical indicator with each indicator working differently in different environment. I used these technical indicators as features for my convolutional network. Rather than letting my CNN learn the feature on its own, I extracted a lot of features to make it easier for the CNN to learn through these features. From the collected data described in previous section, I used the most commonly used indicators by traders to engineer the following features:

Weighted Price

Rather than using just the closing price, it is also preferable to know the volume weighted price. A price change due to a movement of a lot of volume usually has a bigger effect on the market rather than a price change due to a smaller volume movement. So a Volume-Weighted price is a good technical indicator.

Simple Moving Average

Simple Moving Average (SMA) is an arithmetic moving average calculated by taking a mean of the closing price of the previous time periods. In my opinion, the most commonly used SMA within the Bitcoin traders are for time periods = 7, 21, 77, 231. I used these to create four different SMA values. The cross over between these moving averages is very important in predicting the market sentiment and the future price.

Relative Strength Index

Relative Strength Index (RSI) is the most commonly used momentum oscillator that predicts whether the market is in oversold or overbought. The RSI oscillates between zero and 100. Traditionally the RSI is considered overbought when above 70 and oversold when below 30. Signals can be generated by looking for divergences and failure swings. RSI can also be used to identify the general trend. I implemented multiple RSI with different time periods to get a better understanding.

MACD

Moving Average Convergence Divergence (MACD) is again an oscillator. MACD is a trend-following momentum indicator that shows the relationship between two moving averages of prices. The MACD is calculated by subtracting the 26-day exponential moving average (EMA) from the 12-day EMA. A nine-day EMA of the MACD, called

the "signal line", is then plotted on top of the MACD, functioning as a trigger for buy and sell signals.

Image Creation and Data Preparation

Rather than converting this into an exact price prediction, I changed it into a classification problem since that is what the traders are interested in. Since the price of Bitcoin has also changed from a single digit to now five digits, it is also difficult for the neural network to learn this behavior if the price is not standardized or normalized. We created five classes to classify the future value:

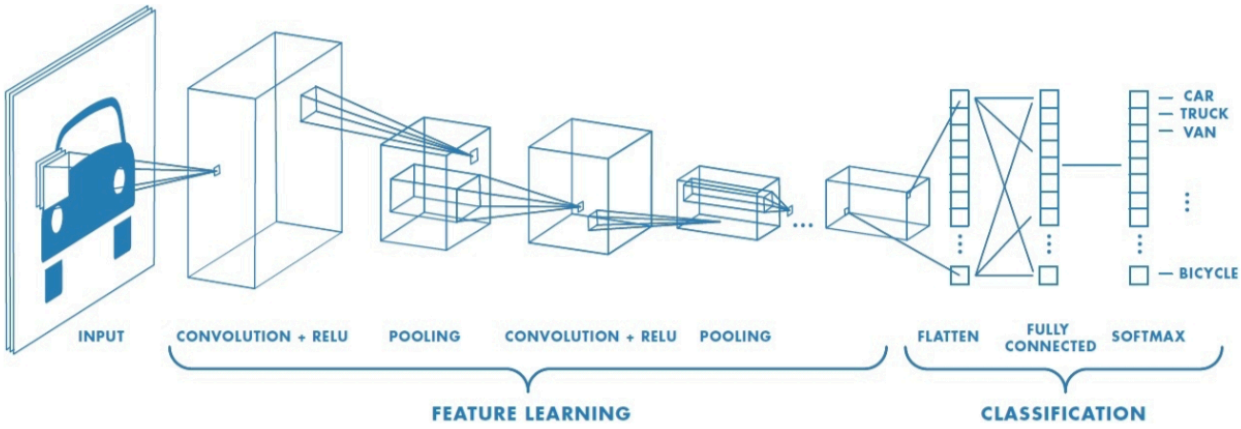
- Within 0.4% of the closing price.
- Between 0.4% to 0.8% above the closing price.
- Between 0.4% to 0.8% lower the closing price.
- More than 0.8% higher than the closing price.
- More than 0.8% lower than the closing price.

When the Bitcoin price is 10,000 USD, 0.8% constitutes an $80 change. Which is quite a reasonable assumption given the volatility of Bitcoin this much change can happen in a time period of 20 minutes.

I used the per minute data and collected it for one hour. After implementing all the indicators, I had multiple time series data of length 60. I combined all these time series data and information to form an Image of size 24x60. This Image would form the input to the convolutional neural network. I also collected in a separate variable a list of all the ground truth or the actual answers. This variable “y” is a one-hot vector of the true classification.

I took 20% as my test data and the remaining 80% is used as a training data.

Convolutional Neural Network



I implemented a CNN containing two convolutional layers followed by a fully connected layer and a softmax for classification.

I input my Data Image into the first layer of a 5x5x32 convolutional neural networks. The output of this network is pooled by a 2x2 maxpool layer. The output of this neural network is passed to another 5x5x2 convolutional neural network followed by another 2x2 maxpool layer.

I take the outputs from the feature learning stage and flattened them out and inserted it into a fully connected layer of 1024 neurons (multi-layer perceptron). I took the output of the fully connected layer and passed it through a softmax classifier with 5 classes.

To prevent the Convolutional Neural Network from overfitting, I implemented a dropout mechanism [4] after the fully connected layer and before the softmax layer. Overfitting is a serious problem in Neural Network and Dropout is a commonly used technique to prevent that.

Simulation Results

I first implemented the CNN in Knet library in Julia. However, due to some errors and a lack of resources in this library I wasn't able to finish that work. Therefore, I shifted to TensorFlow in Python [5]. I re-implemented everything in TensorFlow and it gave accurate predictions.

The results were never able to converge the accuracy to a single value, the more the neural network learned the better the results I got. I ran the Neural network for three hours to get the following range of classification accuracy:

Train dataset accuracy : 75%-95%

Test dataset accuracy : 70%-90%

I believe that for trading, these accuracies are very good predictions. Given the volatile nature of Bitcoin, it is understandable that the prediction accuracy changes over time.

Analysis and Conclusion

I believe that for deep learning, it would require a much longer time to train the CNN. I also believe it would be better to train the CNN with more data. I currently obtained the data from just one exchange, maybe combining multiple exchange data and increasing the volume in our data would help the results.

The more indicators I used the better result I got. I also believe we can add more technical analysis and indicators in our data. The financial markets and traders use tons of very useful indicators and we can maybe employ more of such indicators.

I also believe the predictor will not work when the market is being manipulated. In the past, the market manipulation of Bitcoin was much easier but now since the cryptocurrency market cap is above 300 billion, it would be much difficult to manipulate the market. The price of bitcoin has also fluctuated a lot during this time and it would be preferable to use a standardized or normalized price of Bitcoin.

Furthermore, the user sentiment and the wisdom of the crowd plays a huge role in Bitcoin price. There has been a lot of work done in trying to convert the social media sentiments and news from around the web into quantifiable terms in order to find the social influence on Bitcoin price. Adding such indicators in our data would greatly enhance the predicted price of Bitcoin.

References:

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[2] https://coinmarketcap.com/

[3] https://www.kaggle.com/mczielinski/bitcoin-historical-data/data

[4] https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

[5] https://www.tensorflow.org

Source Code:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
@author: aniqueakhtar
"""

import tensorflow as tf
import numpy as np
import pandas as pd

bitcoin = pd.read_csv("bitstampUSD_1-min_data_2012-01-01_to_2017-10-20.csv")

# Moving average 7, 21, 77, 231

bitcoin['MA_7'] = bitcoin['Weighted_Price'].rolling(window=7).mean()
bitcoin['MA_21'] = bitcoin['Weighted_Price'].rolling(window=21).mean()
bitcoin['MA_77'] = bitcoin['Weighted_Price'].rolling(window=77).mean()
bitcoin['MA_231'] = bitcoin['Weighted_Price'].rolling(window=231).mean()
bitcoin.MA_7.fillna(bitcoin.Weighted_Price, inplace=True)
bitcoin.MA_21.fillna(bitcoin.Weighted_Price, inplace=True)
bitcoin.MA_77.fillna(bitcoin.Weighted_Price, inplace=True)
bitcoin.MA_231.fillna(bitcoin.Weighted_Price, inplace=True)

delta = bitcoin['Weighted_Price'].diff().fillna(0)
dUp, dDown = delta.copy(), delta.copy()
dUp[dUp < 0] = 0
dDown[dDown > 0] = 0
dDown = dDown.abs()

# Window size of 14= 1 mi, 3 min, 5 min, ()added a random) min, 15 min, 30 min, 1 hour, 4 hour.

for n in [14, 42, 70, 150, 210, 420, 840, 3360]:
    RolUp = dUp.rolling(window=n).mean()
    RolDown = dDown.rolling(window=n).mean()
```

```
    RS  = RolUp / RolDown
    RS = RS.fillna(1)
    RSI = 100.0 - (100.0 / (1.0 + RS))
    RSI = RSI.fillna(50)
    bitcoin['RSI_'+str(n)] = RSI


bitcoin = bitcoin.iloc[-525600:,:]
y2 = []
#len(bitcoin)
for i in xrange(60,len(bitcoin),60):
    a = bitcoin['Close'].iloc[i-1]
    b = bitcoin['Close'].iloc[i-1+20]
    percent = ((b-a)/b)*100
    if percent<=0.4 and percent>=-0.4:
        y2.append(2)
    elif percent>0.4 and percent<=0.8:
        y2.append(3)
    elif percent>0.8:
        y2.append(4)
    elif percent<-0.4 and percent>=-0.8:
        y2.append(1)
    elif percent<-0.8:
        y2.append(0)

y1 = np.zeros((len(y2), 5))
y1[np.arange(len(y2)), np.array(y2)] = 1

x1 = bitcoin.as_matrix()
x2 = x1.reshape(8760,60,20)
x2 = x2[:8759,:,:]

x_train = x2[:7059,:,:]
x_test = x2[-1700:,:,:]
y_train = y1[:7059,:]
y_test = y1[-1700:,:]

# CODE FOR CONVOLUTIONAL NEURAL NETWORK
```

```
sess = tf.InteractiveSession()

x = tf.placeholder(tf.float32, shape=[None, 60, 20])
y_ = tf.placeholder(tf.float32, shape=[None, 5])

def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)

def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)

def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')

# CONV 1
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

x_image = tf.reshape(x, [-1, 60, 20, 1])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

# CONV 2
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

# MLP
W_fc1 = weight_variable([15 * 5 * 64, 1024])
```

```
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 15*5*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

# DROPOUT
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# SOFTMAX
W_fc2 = weight_variable([1024, 5])
b_fc2 = bias_variable([5])

y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2



# TRAINING
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

batch_size = 100
N = x_train.shape[0]
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  for i in range(20000):
    batch_ind = np.random.choice(N,batch_size,replace=False)
    if i % 50 == 0:
          train_accuracy = accuracy.eval(feed_dict={x: x_train[batch_ind], y_: y_train[batch_ind], keep_prob: 1.0})
      print('step %d, training accuracy %g' % (i, train_accuracy))
    train_step.run(feed_dict={x: x_train[batch_ind], y_: y_train[batch_ind], keep_prob: 0.5})

# EVALUATING
  print('test accuracy %g' % accuracy.eval(feed_dict={
      x: x_test, y_: y_test, keep_prob: 1.0}))
```