Trade Advisor: Artificial Intelligence based cryptocurrency trade forecasting

Whitepaper version 0.9
First published: August 7, 2018

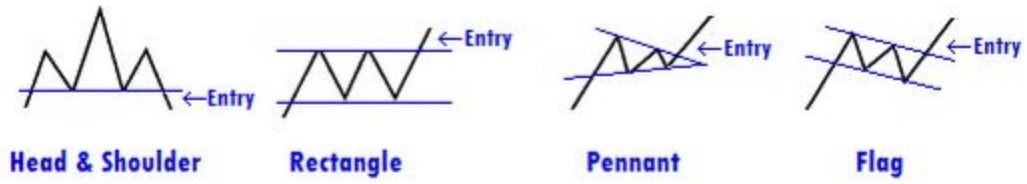
Minche Chang, Justin Woods, Paul Baker, Matthew Fletcher

www.advisor.trade

Contents

Abstract

Since the very beginning of cryptocurrency trading there was a lot of discussions about various patterns that emerge while viewing cryptocurrency trade charts on the exchanges. “Head and shoulders”, “Flag”, “Rectangle” to name a few.



The idea is that previous trading activity of an asset should influence its future price. While the idea is sensible the execution of it is usually limited by humans imagination. Human brain is limited by the images it operates with and is able to process patterns that are no larger than a dozens of points. On the other hand Artificial Neural Networks have proven to be an excellent tool in pattern recognition and can be used to recognise patterns of data that human can not comprehend. Patterns that consist of hundreds and thousands of data points offer no challenge for AI.
Trade Advisor project is introduced to address the problem of trading patterns recognition by the use of elaborate Artificial Neural Networks and computer vision. An algorithm can easily learn and recognize not only basic trading patterns but also discover much more complicated and implicit patterns that human eye can not recognize at all.

Introduction

Trade Advisor (TA) is a computer vision, self learning algorithm based on Keras (tensorflow) library aimed to assist traders in decision-making. The general idea of TA is to digest existing trade history on multiple most populated Cryptocurrency exchanges. Trade Advisor will be parsing exchange tick data (that will also be available for download in case you may want to tinker with your own AI). This data then will be processed in order to represent it as a number of simplified digital images. Pairs of those images will be then used as a datasets for machine learning. The final goal of TA algorithm is to create a computer program that receives last N minutes of market data and outputs it's prediction of price change within next M minutes, that can further be used as a basis to open or close certain market positions.

Solution

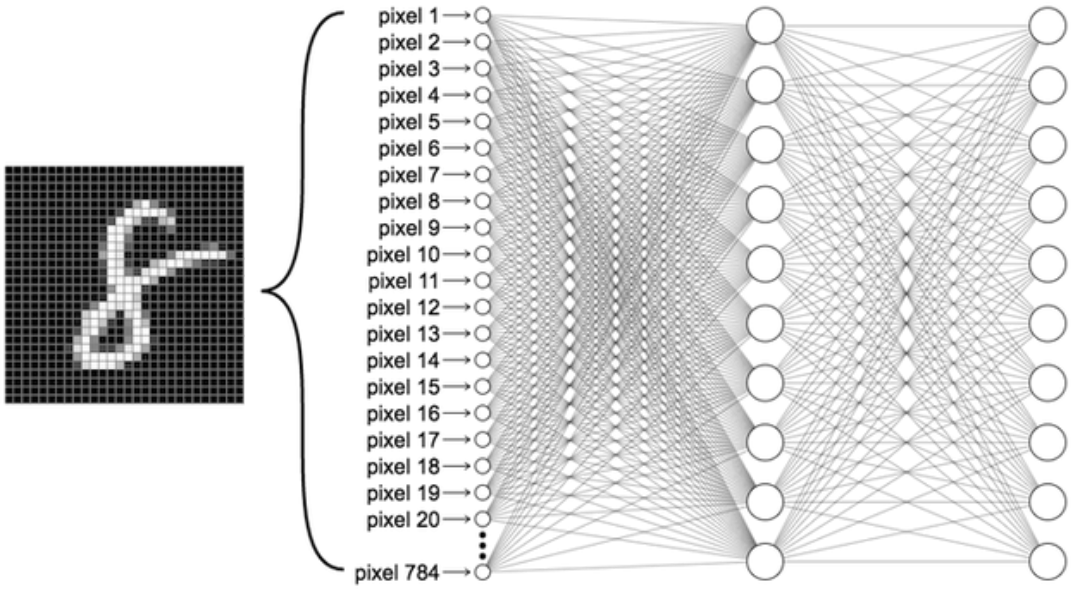
At this stage of development TA uses an algorithm similar to famous ‘MNIST’ in it’s Python implementation. Current prototype is yet to show significant prediction accuracy, but even with relatively small datasets it produces 60%-80% accuracy of market predictions

Understanding TA using classic ‘MNIST’ algorithm

Just like programming has Hello World, machine learning has MNIST. MNIST is a simple computer vision dataset. It consists of images of handwritten digits like these:

Besides the digits it contains labels that describe each digit (5, 0, 4, 1 in this case). A neural network is then trained to recognize which label should be attached to arbitrary picture. Neural network is a computer program that receives an input on each of its input points (so-called neurons). In MNIST example, input picture will be split to a number of individual “pixels”, described by a single digit, that represent its color. In black and white 24x24 pixel implementation, every picture will look something like that after every pixel will be coded to binary in following manner:

Human-readable representation	Raw binary representation:
000000000000000000000000000000 000000000000111110000000000000 000000000010000010000000000 000000000011000011000000000 000000000010000000110000000 000000000011000000000000000 000000000010000000000000000 000000001100000000000000000 000000001100000000000000000 000000001100000000010000000 000000000011000000011100000 000000000001100000100000000 000000000000100001000000000 000000000000001110000000000 000000000000001100000000000 000000000000001001000000000 000000000000011000100000000 000000000110000000010000000 000000001100000000001000000 000000001100000000001000000 000000000011000000000100000 000000000011000000110000000 000000000000001111000000000 000000000000000000000000000	000000000000000000000000000000000000000011111000000000000000000000010000010000000 000000000000001100000110000000000000000000010000000110000000000000000011000000000 000000000000000001000000000000000000000000011000000000000000000000000000011000000 000000000000000000000011000000001000000000000000000110000000111000000000000000011 000001000000000000000000000010000100000000000000000000000011100000000000000000000 000000110000000000000000000000001001000000000000000000000000110001000000000000000 000011000000001000000000000000011000000000010000000000000001000000000010000000000 000000000110000000010000000000000000110000011000000000000000000000011110000000000 000000000000000000000000000000000



Network then will receive each byte of data to an appropriate input, multiply it by the number of weighted coefficients and then summed before entering second layer of neurons. This whole matrix of calculations then produce an output as an array of 10 digits that will represent probability of input image being one of 10 digits. Highest probable digit will be selected as a final answer.

Initially - algorithm does not know how to produce relevant answer so before use it have to undergo learning process, where weighted coefficients will be determined. To do so an algorithm will select each coefficient at random, receive an input with known outcome and compare its own output with what it should be. Then, after thousands and thousands of repeated coefficient adjustments network being tested on another known dataset to determine accuracy of its predictions. If accuracy is satisfactory - network can be used to ‘read’ arbitrary 28x28 picture and tell what number is written on it.

Following simple python code can be easily implemented to try and play with ‘MNIST’ dataset:

```
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28
```

```
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
   x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
   x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
   input_shape = (1, img_rows, img_cols)
else:
   x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
   x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
   input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Computer vision and Market Analysis

As with a MNIST example simple candlestick chart can be easily represented as picture. However instead of ‘normal’ black and white picture, data producing candlestick chart contains way less ‘pixels’. For example standard json response from Binance contain following information:

```
{
  "timezone": "UTC",
  "serverTime": 1508631584636,
{
 "e": "kline",     // Event type
 "E": 123456789,   // Event time
 "s": "BNBBTC",    // Symbol
 "k": {
   "t": 123400000, // Kline start time
   "T": 123460000, // Kline close time
   "s": "BNBBTC",  // Symbol
   "i": "1m",      // Interval
   "f": 100,       // First trade ID
```

```
  "L": 200,       // Last trade ID
  "o": "0.0010",  // Open price
  "c": "0.0020",  // Close price
  "h": "0.0025",  // High price
  "l": "0.0015",  // Low price
  "v": "1000",    // Base asset volume
  "n": 100,       // Number of trades
  "x": false,     // Is this kline closed?
  "q": "1.0000",  // Quote asset volume
  "V": "500",     // Taker buy base asset volume
  "Q": "0.500",   // Taker buy quote asset volume
  "B": "123456"   // Ignore
}
```

Ignoring the metadata this information contains enough information to draw standard candlestick market graph we all used to see. Every minute, this information being collected from Binance API (and APIs of other exchanges like Binance, Coinbase Pro etc) and written to csv file. After dataset grows large enough, all relevant data have to be normalized. In our first examples we used this simple normalization function to digest information and represent each candle as future row of pixels, colored from 0 - white to 1 - black:

```
def normalize(list_of_lists):
    dim = len(list_of_lists[0])
    mins = list()
    maxs = list()
    for i in range(dim):
        column = [row[i] for row in list_of_lists]
        mins.append(min(column))
        maxs.append(max(column))
    for i, row in enumerate(list_of_lists):
        for j, item in enumerate(row):
            list_of_lists[i][j] = (item - mins[j])/(maxs[j] - mins[j])
    col = list()
    for row in list_of_lists:
        col.append(row[0])
    return list_of_lists
```

Those rows of pixels was added on ‘top’ of each other to form a ‘picture’, that represent 10 minutes candle chart - each pixel contains condensed information about one significant parameter of the initial candle and each row of the picture is a visual representation of 10 minutes chart. But how can we know what pattern we are looking for by looking at those seemingly meaningless pictures? In fact it is easiest problem to solve. Since we have weeks and months of tick data collected from exchanges we are already aware of what will happen soon after certain patterns occur simply by analyzing following patterns and price. I.e. if we see that in following 10 minutes we see price increase we classify pattern as “Bullish” and assign coefficient to reflect steepness of market growth. Otherwise pattern is marked as “Bearish” in case of steep decline or “Sideways” if price change was insignificant. Patterns are ranked on a scale from 1 to 9 where 1 indicate strong price decline and 9 - price growth. Each indicator then can be used as a signal to buy or sell cryptocurrency.
Preliminary testing with dataset collected from Binance tick data (raw dataset available for download at a later date) shows that even simplest Keras

implementation of MNIST-like computer vision algorithm can yield significant results. After a sufficient model training on Binance dataset our team was able to achieve 72% accuracy on the real-time data, capable of doing profitable trades roughly 7 times out of 10. However current implementation is in very early stage of development and currently unable to account for sudden price spikes. In order to account for such changes additional control layer have to be implemented on top of the neural network.

Trade Advisor Token [TAT]

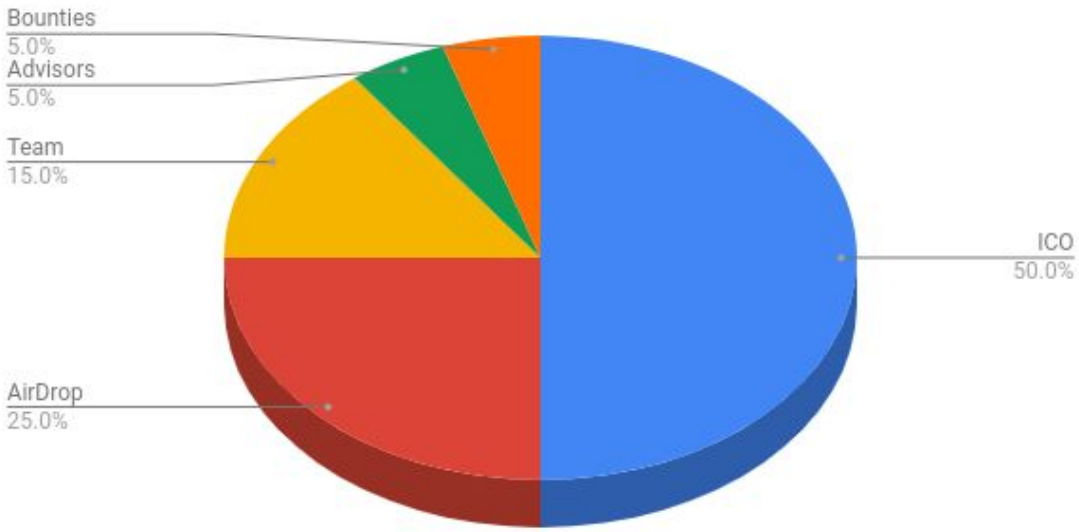
Trade Advisor Token [TAT] is an Ethereum based ERC20 utility token used to grant public access to Trade Advisor service. Trade Advisor will redeem TATs in exchange for requests quota, allowing users to access TA web service anonimpously. However we understand that at this very early stage of development, significant part TAT tokens should be distributed for free in order to facilitate fluent and easy to access testing environment for the community.

We believe that in order for Trade Advisor become a successful and widely used service it needs to have an optimal token distribution strategy.

Total supply: 100,000,000 TAT

A large portion (25%) of total TAT supply is going to be distributed through an airdrop for free to ensure a large base of supporters. We also plan to launch an ICO where we will offer to buy any amount of TAT at $0.01 price. An ICO stage will only begin as soon as working prototype is available for testing.

Chart below represents Trade Advisor Token distribution.



Trade Advisor Tokens will be used as a currency to purchase cryptocurrency trade forecasting services on www.advisor.trade. Users will be able to purchase forecasts for any major cryptocurrency market paying with TAT. Later on we plan to release the source code of the service behind www.advisor.trade so that a broader market of trading forecasts may emerge.

Team behind the Trade Adviser project

	**Minche Chang** CEO + Co-founder
	**Justin Woods** CTO + Co-founder
	**Paul Baker** COO

	**Brian Bjorn** Business Development
	**Namrata Ahuja** Chief of Staff
	**Matthew Fletcher** Data Scientist
	**Jake Tudge** Economics and Finance consultant

References:

Tensorflow - An open source machine learning framework for everyone	https://www.tensorflow.org/
Keras - Deep learning for Humans	https://github.com/keras-team
MNIST - Machine Learning for beginners	https://www.tensorflow.org/versions/r1.0/get_started/mnist/beginners
François Chollet - Deep learning with Python	https://www.manning.com/books/deep-learning-with-python
Tensorforce - Bitcoin Trading Bot based on reinforced training	https://github.com/lefnire/tforce_btc_trader

Contacts

If you would like to contribute to the project, feel free to contact our team via email: support@advisor.trade
For the most up-to-date info about Trade Advisor Project please visit our website: www.advisor.trade