Austin Apt
Master's Project

A Survey of Trading Bot Methods and Experimental Analysis of a Reinforcement Learning Architecture

Summary:

Trading bots, also known as automated trading systems (ATS), are automated agents which react to changes in the market. This reaction consists of creating buy and sell orders, and then submitting them to some market exchange. When this reaction occurs is decided by that agent's trading strategy, which typically consists of technical analysis-related indicators or other inputs such as sentiment analysis statistics sourced from social media e.g. a web scrape of all posts on Twitter and the Wall Street Journal related to stock "X".

There are many advantages to using an ATS over human agents in exchange for the risks of using a theoretical algorithm to handle your real money. Bots eliminate emotion in the decision-making process, i.e. once a set of rules for a trading strategy is defined, certain price and volume movements will always trigger the same agent reaction even when the market is volatile. Automated agents are also able to execute trades magnitudes faster than human agents once a relevant event or indicator in the strategy is signaled.

This paper presents a survey of existing classical methods, and implements two novel state-of-the-art architectures in order to measure their merits. An example of a simple classical trading bot is one that uses a relative strength index (RSI) to determine when a stock is overbought or oversold. The RSI is a measure of how well a stock is currently performing compared to moving averages (MA) of its gains and losses over some period of time (typically 14 days). The equation is given here:

$$RSI = 100 - \frac{100}{1 + RS}$$

$$\text{Average Gain} = \frac{\text{(Total Gains/n)}}{\text{(Total Losses/n)}} = \text{Average Loss}$$

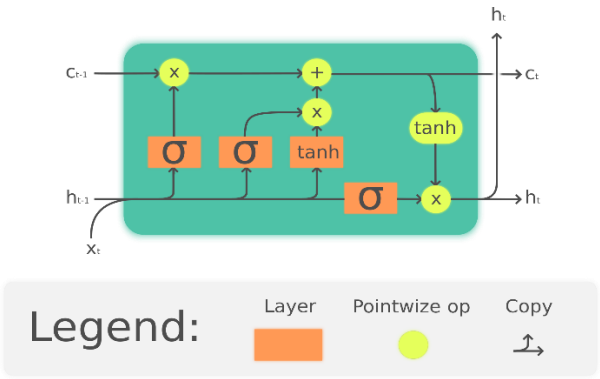
$$\text{First RS} = \text{(Average Gain/Average Loss)}$$

$$\text{Smoothed RS} = \frac{\text{[(previous Average Gain) x 13 + Current Gain]/14}}{\text{[(previous Average Loss) x 13 + Current Loss]/14}}$$

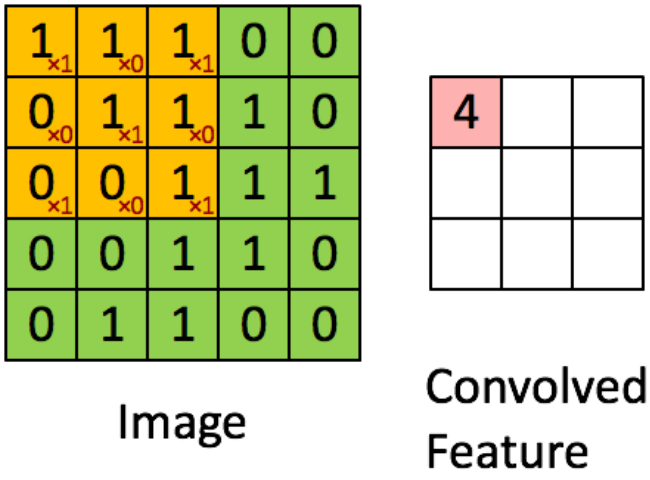
$$n = \text{number of RSI periods}$$

Rules such as the 70/30 are used with RSI in order to create a buy/sell signal (acc. to [1], 66.6 and 33.3 are truer indicators of bull and bear markets than 70/30 or 80/20 rules). More specifically, when the RSI reaches a value of 70, this indicates that the stock is overbought by the market at large and should be sold by reasoning that a downward correction will shortly follow. Conversely, when the RSI reaches 30, this designates that the stock is oversold and may shortly have a positive correction. The RSI is only one of many man-made indicators which can be used as a basis for an agent's buy and sell signals. Another is the Ichimoku Cloud which uses four different moving averages, and a lagged "closing price" feature line. When the Leading Span A (average of 9-day and 26-day MA's) is above Leading Span B derived from a 52-day MA, it signifies an uptrend and that recent movements suggest a higher price support and bull market conditions. The Ichimoku Cloud is often combined with RSI in order to maximize risk-adjusted returns. A RSI is useful for getting immediate results and current momentum, while Ichimoku adds more context to price movements by plotting various levels of support and resistance that can be used to see bigger trends in the data as shown below:

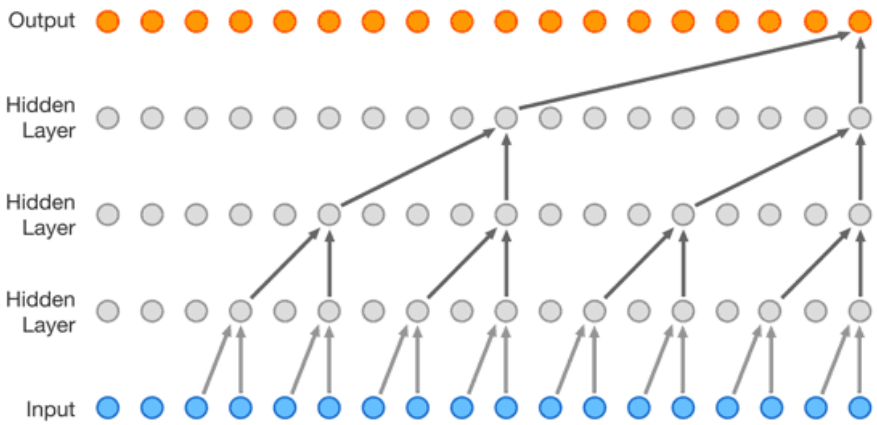
Naturally, as with all time-series data, neural nets began being applied to stock and cryptocurrency information. Deep learning models such as the recurrent neural network (RNN), and its further development into long-short-term memory (LSTM, pictured below) networks achieved effective prediction results, often without any features besides open and closing prices. LSTM networks are theoretically able to capture both short and longer-ranging dependencies between data points in the time series through its use of numerous gated functions intended to mimic human concepts of working memory, i.e. what to discard, and what to keep in order to have the most relevant information at each step in time. This project is an extension of my earlier 2019 implementation of an LSTM network which was used to predict/forecast Bitcoin (BTC) price. At the time of that writing BTC was around $4,000 and my model showed it should double to $8,000 in only a few months; today in December 2020 it is routinely greater than $20,000. This paper improves on those early methods by implementing two higher-complexity, state of the art prediction models. Another difference is that here actionable agents were created after the model to see how an ATS built using these models actually performs as opposed to just predicting BTC close price as in 2019.



While increasingly complicated versions of RNN's continue to be used on time series, there has also been recent use of convolutional neural nets (CNN) to generate features from time series data and predict future outputs. Typically, CNN's are used to apply 2-d or 3-d convolutions to images and videos respectively in order to capture robust information by the convolutional kernels which are moved along groups of pixels to create feature maps of the original image. E.g. the 3x3 kernel being applied to the below 5x5 image creates a 3x3 feature map.

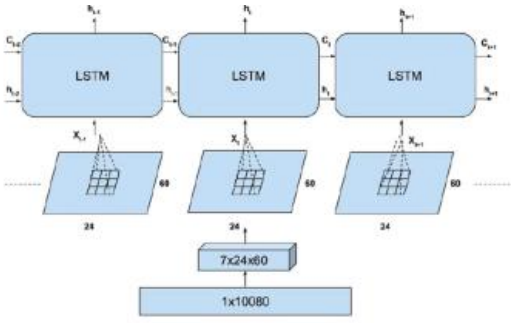


In 2016, Google's Deepmind presented WaveNet, a causal dilated 1-d CNN which was used to generate and predict audio waveforms. Instead of being applied to images and videos, this model presented a CNN which applies 1-d convolutions on sequenced or windowed time series input data. Causal denotes that only past observed information is considered for future predictions and dilated refers to the way in which the receptive field grows exponentially for each convolutional layer as shown below. Similarly to the LSTM, this architecture is an attempt to capture clear present information while also maintaining some effective representation of data that may be far in the past.



Another approach which the paper explores is a model which combines CNN and LSTM architectures. In this case, the feature maps extracted by the 1-d convolutions on the time series data are used as sequenced input to an LSTM network. This is defined in the keras file by the following four lines of code. Below that is an illustration of the architecture “ConvLSTM”.

```
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=7, activation='re
lu'), input_shape=(None, n_steps, n_features)))
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(50, activation='relu'))
```



(b) Structure ConvLSTM*

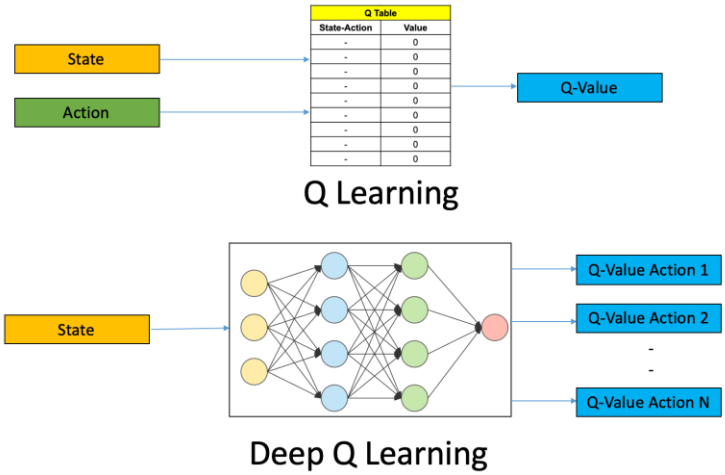
Tensortrade: https://www.tensortrade.org/en/latest/index.html

The approaches mentioned thus far are part of the supervised learning paradigm of machine learning; this section of the paper introduces a reinforcement-learning (RL) based ATS. In short, supervised learning models are trained using labelled datasets where the desired outcome or prediction (usually expressed as y or the target value) is given. In RL models, an agent is able to teach itself through its own repeated interactions within its observed environment. Instead of using an error function to compare results to actual values (this is

more desired for regression or classification problems), the agent has some notion of a reward function which it will continue to improve on through being in the environment. At a given time step, usually known as a state *s*, the agent can take some action *a*, and then receive a reward *R* and be in a new state *s'*.

TensorTrade is a specialized implementation of the general deep RL framework Tensorforce, which is itself branched from TensorFlow. The package consists of a variety of modular components which allows the user to quickly test a variety of different trading bots and schemes. The implementation discussed here uses a Deep-Q Network (DQN) as the basis for agent actions. DQN's are an extension of Q-Learning (QL) where actions are chosen using a cost function that incorporates current and subsequent future rewards due to that action. At each step or state s, QL creates a Q-table (expressed Q[S,A]) where all possible actions, and the subsequent states s' they bring the agent to are considered. Q' represents the ideal policy to take at that step.

Depending on the state and action space, this can become prohibitively expensive to update at each iteration of the agent's exploration e.g. 10,000 states and 1,000 possible actions creates a table of size 10,000,000. DQN's attempt to solve this issue by using a neural network to approximate the ideal Q-value function at each state.



Evaluation and Results:

Tensortrade models were created using TensorFlow, and the ConvLSTM with keras using a TensorFlow backend. Training and tests were completed on a 8-core local machine with Intel i7-8550U CPU and 16GB RAM. ConvLSTM were built using mean squared error (MSE) as its loss function, and Adam as the optimizer. Datasets are drawn and updated from

cryptodatadownload.com, and the yfinance (Yahoo Finance) Python library. Feature sets were created with the Python ta (technical analysis) library and its “add_all_ta_features” function.

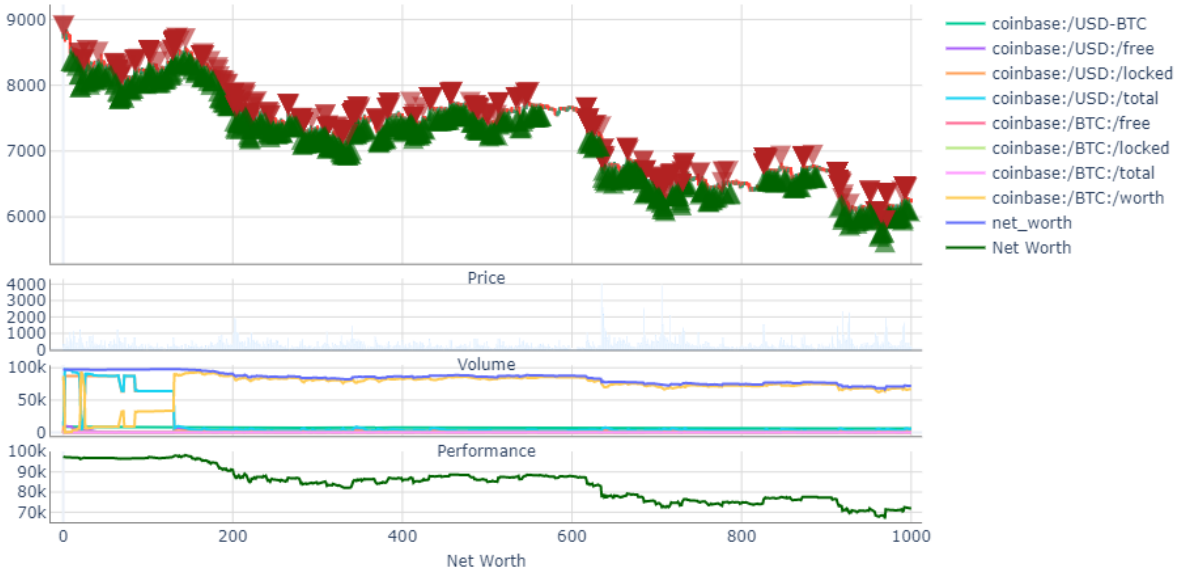
The graphs shown below display actions and rewards of the DQN agent from Tensortrade, as well as the Conv-LSTM model test and validation results. These charts were drawn using Plotly and contain interactive information when a cursor is dragged over. Green arrows show the agent’s buy orders, while red is for sell; the interactive chart displays the volume and $ amount of BTC exchanged in that step when the cursor is drawn over an arrow.

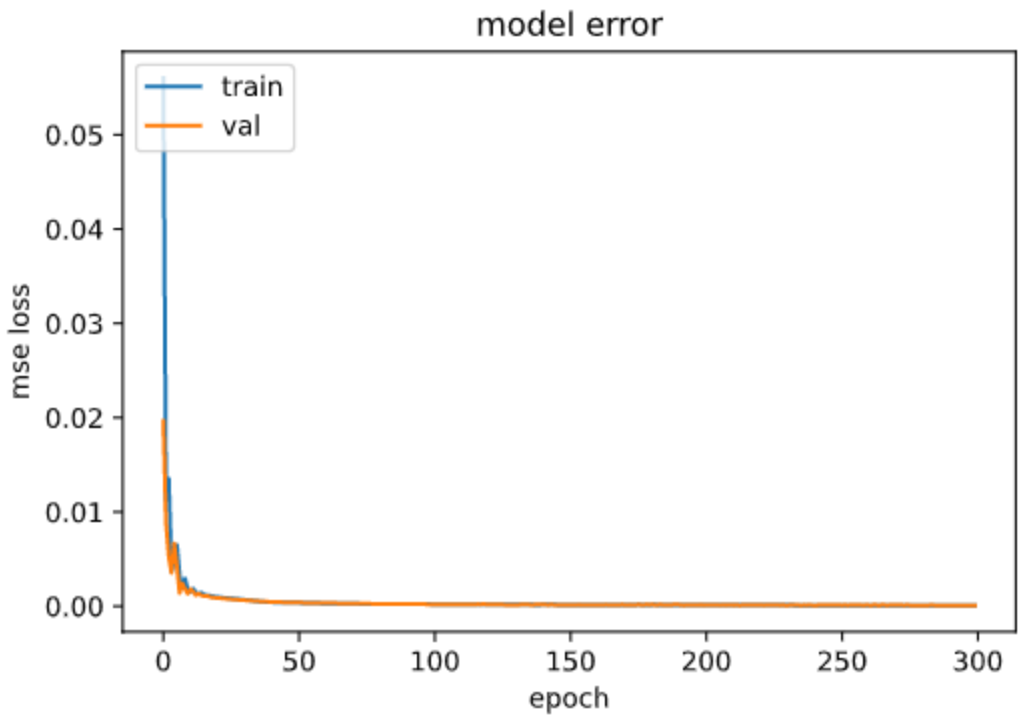
[2020-12-17 23:19:03 PM] Episode: 100/100 Step: 101/200

[2020-12-17 23:19:08 PM] Episode: 100/100 Step: 151/200

[2020-12-17 23:19:12 PM] Episode: 100/100 Step: 201/200

[2020-12-19 23:54:38 PM] Episode: 50/50 Step: 1001/1000





Conclusion:

The two models here are capable of performing well on the data of both cryptocurrencies and traditional stocks such as Apple. Future work involves calculating returns over a longer period of time and allowing the bots to run concurrently throughout the day. Further development of this project could be with the use of ETF's and portfolios instead of the single currency and stock performance covered in this paper.

The difference between the DQN bot's performance depending on training length shows the need for further hyperparameter optimization. This applies for the ConvLSTM as well where a kernel size of seven was chosen after a small grid search [1-12], and may not represent the optimal sequence length to convolute.

While the DQN rarely made significant amounts of profit, the charts show that this was due to the bot ignoring longer-term trends in the data and oftentimes buying or selling at the slightest hint of price movement. Even though RSI and other indicators showing levels of support are created as features, it is possible that further feature selection is required for more effective models. Baseline results of a simple rule-based bot using, e.g. RSI and Ichimoku would provide further evidence to the deep learning approach's effectiveness.

The inspiration for this project comes from my earlier work with BTC prediction, and having a lot more time for commission-free trading with apps such as Alpaca and Robinhood during the time of the Covid-19 shutdown. The ConvLSTM network is setup to take advantage of Alpaca's API and is capable of executing actual trades on the market which is in testing now.

Code for this Project here:

1. https://github.com/ama66843/ConvolutionalTensorTradeBot
2. https://github.com/ama66843/tensortrade

References:

1. "Feature extraction using convolution," *Feature extraction using convolution - Ufldl*. [Online]. Available: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution. [Accessed: 01-Dec-2020].
2. J. H. Hayden, *RSI: the complete guide*. Greenville, SC: Traders Press, 2004.
3. "How to Utilize the Ichimoku Cloud Trading Strategy," *Timothy Sykes*, 22-Oct-2020. [Online]. Available: https://www.timothysykes.com/blog/ichimoku-cloud/. [Accessed: 02-Dec-2020].
4. V. Sim, "Using Deep Learning to Create a Stock Trading Bot," *Medium*, 27-Aug-2020. [Online]. Available: https://medium.com/analytics-vidhya/using-deep-learning-to-create-a-stock-trading-bot-a96e6351d31c. [Accessed: 01-Dec-2020].
5. A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," arXiv.org, 19-Sep-2016. [Online]. Available: https://arxiv.org/abs/1609.03499v2. [Accessed: 08-Dec-2020].
6. Sepp Hochreiter and J¨urgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8), 1997.
7. Matiisen, Tambet (December 19, 2015). "Demystifying Deep Reinforcement Learning". *neuro.cs.ut.ee*. Computational Neuroscience Lab. Retrieved 2018-04-06.
8. Ankit ChoudharyIIT Bombay Graduate with a Masters and Bachelors in Electrical Engineering.I have previously worked as a lead decision scientist for Indian National Congress deploying statistical models (Segmentation, "Deep Q-Learning: An Introduction To Deep Reinforcement Learning," *Analytics Vidhya*, 27-Apr-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/. [Accessed: 01-Dec-2020].