Novel Deep Learning Trading Bot Strategies

Austin Apt
Department of Computer Science
University of Georgia
Athens, USA
ama66843@uga.edu

***Abstract*—:**

Automated trading systems have seen increasingly wide use since the 1980's. These are agents capable of automatically executing buy and sell orders on a commodity exchange ordering to some set of rules. Recent developments of commission-free trading and the 24/7 cryptocurrency market present two new incentives for the development of always-online trading bots. This paper presents state-of-the-art automated trading systems built using supervised and reinforcement learning-based algorithms to measure their performance on the market.

***Keywords*-Automated Trading Systems, Deep Q Networks, Convolutional Time Series, Long-Short Term Memory, Cryptocurrency**

The code for this paper here: https://github.com/ama66843/tensortrade

and here

https://github.com/ama66843/ConvLSTMTradingBot

I. Introduction

There are more opportunities than ever for individuals to invest with the advent of commission-free trading providers such as Robinhood and Alpaca. These companies typically offer relevant application programming interfaces (API) which allow individuals to access the market and automatically execute buy and sell orders using a scripting language like Python. This has driven the continued development of trading bots, which are examples of automated trading systems (ATS). An ATS can be setup to check the market for any given time interval and execute an arbitrarily large number of orders faster than any human, while also saving the account holder from having to constantly check the market by hand.

Classical ATS systems typically consist of expert-curated features and indicators such as a stock's relative strength index (RSI), moving average convergence divergence (MACD) and Ichimoku Cloud structure. 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.

More recently, ATS's have been created using machine learning (ML) principles, as seen across most fields with appropriate time series data. This includes the application of long-short term memory (LSTM) [2], which are examples of recurrent neural networks (RNN) with the addition of multiple non-linear activation gates to mimic a human's ability to remember and forget. In 2016, Google 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 to creating an ATS system is to format the stock market as a playing field for a reinforcement-learning (RL) based agent. 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'. In the case of this paper, the state or environment is current incoming cryptocurrency or stock price data, and the agent's current portfolio split between liquid dollars and commodities. The available action space consists of, e.g. making (1) buy and (2) sell orders, or holding (3) a commodity at each given time step or state given.

TensorTrade [14] 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

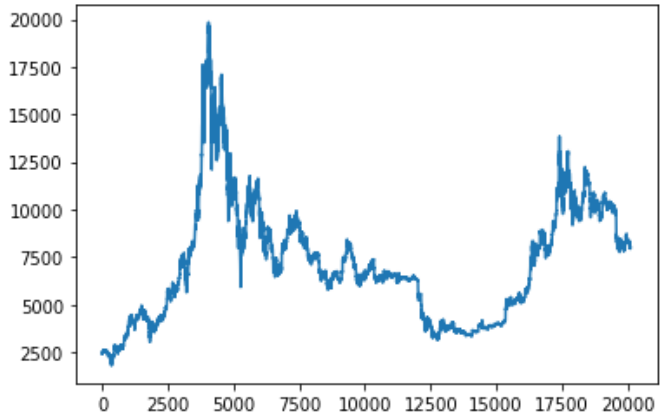

Fig. 1. Bitcoin/USD Closing Price (7.1.17 to 10.17.19)

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. Instead of a Q table being calculated for every state and action, only the current state is used as input to a DQN and Q-values are output for all possible actions.

II. Related Work

There has been extensive research in using time series models to capture and predict commodity price movement. Classical methods like linear regression and ARMA models have been used, as well as classifier-based methods using SVM or Random Forests which use additional features to predict when stock movements occur. The highest performing models use some form of LSTM in order to learn and predict future price values.

This paper presents an indicator-based agent, as well as a reinforcement-learning based model implementation. Existing models have used data survey intervals ranging from an hour to a whole day, with longer intervals generally allowing for smoother data and fewer demand spikes. Depending on the desired frequency of trading, there are different levels of time granularity most effective for the relevant prediction task.

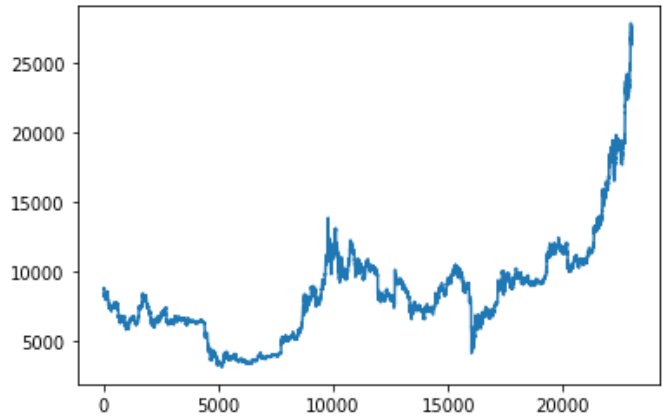

Fig. 2. Bitcoin/USD Closing Price (5.15.18 to 12.28.20)

Cryptocurrency and most stock information is available at a resolution of minutes or even seconds, and this paper uses hourly data when training and making predictions with the models. Effective stock trading is a task which requires small resolution or large resolution depending on the agent. A rapid day-trader would like to make orders frequently throughout the day, requiring them to have information on movements throughout a given day. Others more interested in slower-growth retirement or exchange-traded fund (ETF) investments still profit from having future stock information a week, month or even years ahead. In either case, market makers can make limit buy and sell orders much more profitably knowing the future direction of a stock's price movement.

III. Approach

A. *Problem Definition*

A typical trader would like to know commodity price at some point in the future, for example tomorrow or next month, in order to make a buy or sell decision today. This can be predicted given some past sequence of daily stock open, close, high, low and volume information in the most basic sense.

The data used here is hourly to take advantage of the fine granularity available for cryptocurrency data. The objective is to find a time series model, which when given some past sequence of data, can accurately predict overall commodity price at least one timestep ahead. With this knowledge in hand, a trading agent is able to choose whether to buy, sell or hold a commodity for maximum reward.

B. *Design*

The remaining models were trained by sequencing the hourly data into sliding windows of width 40, representing nearly two days' worth of data. This is represented by the equation (1) below with n=40, with the model predicting stock price one timestep ahead.

$$P(t) = f(P(t-1), P(t-2),...,P(t-n+1), P(t-n)) \quad (1)$$

This value must be weighed and balanced against both hyperparameter and practical considerations. While there is some optimal model value for this sequence length, the model becomes computationally more expensive the longer it is; it then also requires that much more data to create predictions. Window widths were tested between 20 and 40, representing lengths of information from a little under one day to almost two days in length. It was found that models built using a window of 40 were able to turn profit depending on overall market conditions.

C. *Long Short Term Memory Model*

The LSTM [2] cell was created in order to capture both long and short-term dependencies when trained on time series data. It is a variant of a recurrent neural network (RNN) with extra gated functions meant to mimic the human brain's decision to store or forget information up to that point. Generally speaking, these additional non-linear activation gates avoid the vanishing gradient and missing long-term dependency problems of traditional RNN's.

Neural networks built with LSTM cells have been used extensively in recent history, commonly outperforming the most complex existing linear models e.g. SARIMA. This has occurred across a wide variety of fields where time series data is involved: natural language processing, stock price and traffic flow prediction, etc.

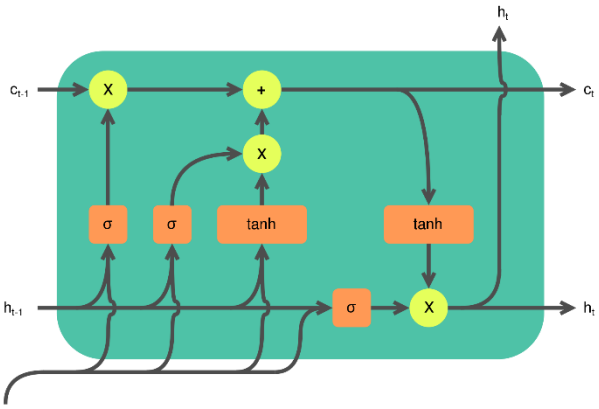

Fig. 3. LSTM Cell with input data at bottom left

Hyperparameter optimization is required when building a large neural network if the model is expected to perform best on that one specific dataset. This paper uses a small grid search based off values commonly used for basic LSTM architectures. Due to the relative insensitivity to gap length when training as compared to traditional RNN, most LSTM models still perform well if built with a sufficiently complex network architecture. Because these models will be exposed to unseen datasets and time series data is non-stationary, it is not vital that they be completely optimized on any one price history sequence yet.

D. *Causal Dilated Convolutional Net Model*

Convolutional neural nets work by applying kernels, defined by some width and height, to input data. The variety of kernels aim to extract a feature map which is representative of the input data. Similarly to LSTM models, CNN are inspired by biological processes. It is based off the human eye, which functions as one organ but contains millions of individual cortical neurons which individually only respond to a restricted region known as the "receptive field" [10] which is analogous to the kernel passing over small portions of input data. Historically they have been used to great success on image and video classification tasks. Since Google Deepmind's WaveNet [6] paper 1-d CNN have been applied to more traditional time series tasks such as stock price and traffic prediction with good results.

WaveNet utilizes a causal dilated convolution net. Causal denotes that the filters are only applied to past sequences of data relative to the current timestep and dilated refers to the way in which the model grows its receptive field. The convolutional filter grows in width exponentially as it covers increasingly larger causal past sequences, while avoiding being computationally prohibitive. (2) shows how a list containing dilations with a base of 2 are created in pseudocode.

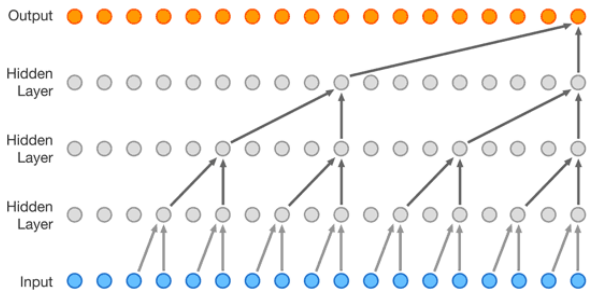

Fig 4. Wavenet CNN with dilation rate of 2

$$dilation\_rates = [2^{**}i \text{ for } i=0 \text{ to } 7] \qquad (2)$$

This paper implements an architecture similar to that pictured in Fig. 4., but with the feature map resulting from 1-d convolutions on the sequenced data being used as input to an LSTM instead of the raw data. This creates is what is known as a ConvLSTM model. The Deep-Q Network also makes use of a 1d CNN for its neural net architecture.

Hyperparameter tuning is more important for CNN than most LSTM when building models which approach LSTM baselines. Such values are the aforementioned dilation rate, the number of layers in the network, learning rate, number and size of the kernels in each layer, batch size, activation function, the addition of dense and/or dropout layers, etc.

E. Deep-Q Network

Reinforcement learning has seen a lot of recent use and success on tasks which have no clear, labelled ideal output set on which to train on. These cases include 2d games such as Chess or Super Mario, to more recent 3d games such as League of Legends and Doom. The DQN model used here is based off another Google Deepmind paper [11] where a Q-Learning infrastructure was augmented with a CNN to replace the Q-table function calculations, and performed better than humans on a variety of Atari 2600 tasks using the same model hyperparameters across all games.

An RL model is trained by repeated interactions of some actor A, with its state environment S (or E); at the end of each state interaction the actor receives a notion of positive or negative reward due to its choice of action for the given S. The sequence of actions an agent takes is typically expressed as a Markov decision process (MDP), where the optimal policy function is one that maps the different states to specific actions which maximize reward in that situation. This paper uses Tensortrade's implementation of a DQN with the available action space given here:

Action Space

The **action space** is a discrete set of N options for the model to take. The total number of discrete actions for the "ManagedRisk" action scheme is determined by taking a product of:

- stop percents (i.e. [0.02, 0.04, 0.06] percent changes to trigger a stop loss)
- take percents (i.e. [0.02, 0.03] value percent changes to take profit at)
- trade sizes (e.g. 1/4, 1/2, 1/3)
- trade durations (e.g. order open for 30 seconds or 60 seconds)
- trade sides (i.e. Buy or Sell)
- the number of tradable pairs (i.e. BTC/USDT, ETH/BTC, etc)

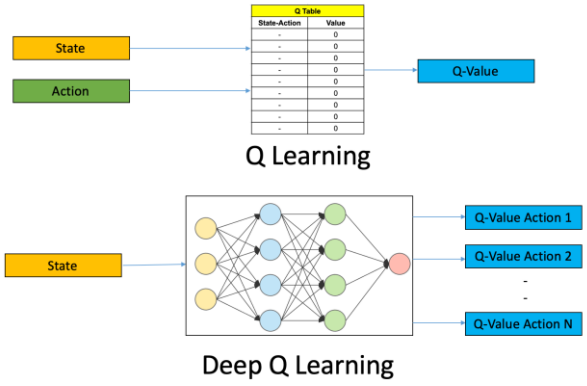

Fig. 5. (Deep) Q Learning

In traditional Q-learning, agents calculate a Q-table which is initialized as an m x n matrix of zeroes where m is given by the number of possible states the agent can be in, and n is the number of actions available to the agent. Each entry of the Q-table holds a Q-value function value, defined as Q(s,a) where s is a state and a is an action. The Q-value function for Q(s,a) is the sum of the immediate reward gained from taking action a in state s, and the expected value of future rewards due to taking action a' in future state s' which is reached due to taking earlier action a in state s. As the agent selects an action a at each time step t, it observes a rewards, is returned a new state environment and updates the Q-table for that value. Q-values are updated using a Bellman equation given by (3). Iterative updates of the Bellman equation guarantee convergence to the ideal Q-value function Q*, given any tuple of state s and action a.

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)}^{\text{temporal difference}}_{\text{new value (temporal difference target)}} \quad (3)$$

Deep-Q Learning replaces the recalculation of each cell in the Q-table by using a neural network to approximate all Q-values for the available action space given only the current state information as input. The original Deepmind DQN paper used a CNN with 2d convolutions due to image data as input, but the architecture in this paper has been repurposed to 1d time series convolutions as demonstrated in WaveNet. Input is the current state environment, along with a past sequence of states which are convoluted to lower dimensionality feature maps. The CNN used for this paper has two hidden convolutional layers, with max pooling placed after each one; after this is a flatten layer followed by two dense layers.

IV. Experiments

This section presents the experimental portion of the project. Models were tested on the two different periods of stock data information presented in Figs. 1. and 2.

A. Experimental Setup

Time Periods: Two windows of data were created when testing the DQN. The first ranged from 2017-07-01 to 2019-10-17 when Bitcoin closed at $2500 to $8000. The second was from 2018-05-15 to 2020-12-15, when Bitcoin first closed at $8740, to where it currently sits around $28000. This allowed for observing how models performed in bear or bull market conditions, and shows the clear non-stationarity of cryptocurrency price data.

Metrics: Supervised models were trained using mean squared error (MSE) as a loss function, which is formulated as the sum of squared (prediction) errors (SSE) divided by the number of samples. Loss values were collected along with R2 and mean absolute error (MAE) statistics for each model. RL models were trained using a Risk Adjusted Return reward metric at each state. This is based off a Sortino Ratio rating of the current portfolio. The Sortino Ratio as shown in Fig. 7. is a version of the Sharpe Ratio where only negative commodity variance (also known as risk) is penalized as opposed to volatility.

$$S = \frac{(R - MAR)}{\text{downside deviation}}$$

.

Fig. 6. Sortino Ratio

Baselines: A trading bot utilizing a combination of traditional RSI and MACD indicators was implemented on the Bitcoin Dataset #2 using the python Backtesting package. Reinforcement learning agents using the "Simple" Action Scheme which does not have action sets for creating stop-loss or take orders.

Implementation: The models were built using sklearn, pandas, tensorflow and keras. Training and tests were completed on a 8-core local machine with Intel i7-8550U CPU and 16GB RAM. Models were built using both MSE and MAE as loss functions during initial testing before deciding to use MSE for the models displayed in this paper from better performance.

B. Evaluation of Model Accuracy

Tensortrade models were created using TensorFlow and trained for a variable amount of training episodes ranging from 2 to 250 which completed in about 16 hours. Training and tests were completed on a 8-core local machine with Intel i7-8550U CPU and 16GB RAM.

Fig. 7. Episode 50 1000-Steps D2 "Simple" Action Scheme

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. An Augmented Dickey-Fuller test was run on Dataset #2, outputting a value of .99 which greatly suggests the data is non-stationary.

A trading bot using moving average and RSI indicators was implemented using the python Backtesting library. The buy signal is sent if the weekly RSI(30) > daily RSI(30) > 70 and the closing price > MA(10) > MA(20) > MA(50) > MA(100); a stop-loss order at 92% of the buy price is also set. A sell signal is generated when the closing price is more than 2% below the MA(10) or the 8% stop loss is hit. In the 1000-period plot represented in Fig. 10., not a single trade was made. For the given dataset, this was a near optimal strategy as there is very little upside to Bitcoin price at any point. When given the complete dataset of 968 days instead of the 41 days represented in the first 1000 timesteps, the indicator bot completed only three trades with a win rate of 33.3% shown in Fig. 8.

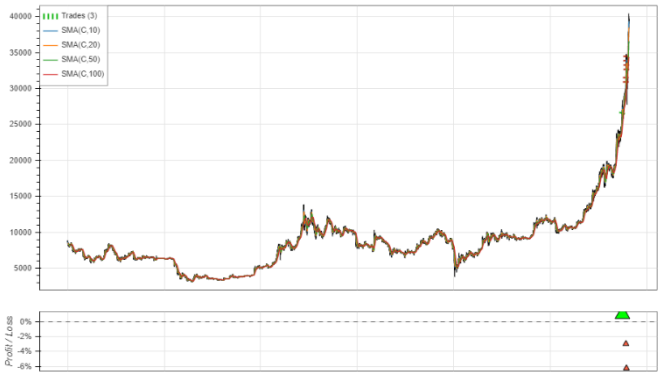

Fig. 8. Indicator Bot on Full Dataset #2 with Profit and Loss Trade %

The graphs shown below display actions and rewards of the DQN agent from Tensortrade alongside the Indicator-based model. 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.

Fig. 9. Episode 4 1000-Steps Dataset #1

The market conditions have a huge effect on the bots' performance. Fig. 9. displays the first 1000 hours following the start of Dataset #1 (D1) on 7/1/2017, while Fig.

Fig. 10. Episode 250 1000-Steps Dataset #2

10. shows 1000 hours of predictions starting from 5/15/18 aka the beginning of Dataset #2 (D2). The overall increase in

Fig. 11. Moving averages [10,20,50,100] drawn for D2

Bitcoin price is around $1200 for D1 (up 50%), while it decreases over $2500 (down 29%) in D2. DQN Agents always made significant profit on D1, even early in the training process as seen with Figs. 9., 13. One clear disadvantage to the DQN agents trained with the Sortino ratio is that they do not convert Bitcoin to USD readily enough given sharp downward pressure. In both models, the price of Bitcoin clearly correlates positively with net worth as the agents' portfolios have very little USD. At the same time, this allowed agents to make about $10,000 or 28.5% profit in about 42 days. This suggests that there is not enough penalty in the model training for losses due to holding a commodity in one form, verses liquidating it to USD and avoiding the losses due to keeping a falling commodity.

DQN models using the "MangedRisk" action scheme never performed as well as the "Simple" model, despite being trained on five times as many episodes. It is possible that more epochs would have allowed the agent to learn how to perform well in a downward market, but little improvement is seen even when comparing Figs. 10 and 12 which are separated by 247 training episodes. The agent using a "Simple" action scheme displayed in Fig. 7. performed much better and more readily converted its BTC to USD when price trended downwards. Overall loss was limited to 5.2% compared to 29.1% loss in value for BTC during the same period.

Models were built using one of two starting portfolios which represented a mixture of U.S. Dollar and Bitcoin commodities. All except for that displayed in Fig. 9. were created using a starting wallet with $10,000 USD and 10 Bitcoins; 9 used an experimental wallet of $100,000 and 5 Bitcoins to measure the effect of starting portfolio on overall profitability. However even in this case, the majority of USD was converted to Bitcoin near completely in timesteps one and two, showing the starting portfolio admixture has little effect on future profitability. Even after sharp downward spikes shown in the first third of each dataset, the agent held most of its value in Bitcoin which is a reason for the aforementioned correlation between BTC price and net worth.

The indicator-based model should be expected to perform well on Dataset #2 as it is a bear market and an optimal agent would make little to no trades. The agent built using a "Simple" action scheme was competitive and had better upside during bull markets.

C. Figures and Tables

Fig. 12. Episode 3 1000-Steps D2

Fig. 13. Episode 3 1000-Steps D1

```
arima_model = pmdarima.auto_arima(train, start_p=2, d=0, start_q=0,
                      max_p=5, max_d=1, max_q=3, start_P=2,
                      D=1, start_Q=1, max_P=5, max_D=1,
                      max_Q=5, m=8, seasonal=True,
                      error_action='warn', trace = True,
                      suppress_warnings=True, stepwise=True,
                      random_state=20, n_fits=10)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,0)(2,1,1)[8] intercept   : AIC=inf, Time=135.26 sec
 ARIMA(0,0,0)(0,1,0)[8] intercept   : AIC=242363.673, Time=0.71 sec
 ARIMA(1,0,0)(1,1,0)[8] intercept   : AIC=212470.694, Time=17.55 sec
 ARIMA(0,0,1)(0,1,1)[8] intercept   : AIC=227939.010, Time=15.93 sec
 ARIMA(0,0,0)(0,1,0)[8]             : AIC=242361.695, Time=0.36 sec
 ARIMA(1,0,0)(0,1,0)[8] intercept   : AIC=217189.499, Time=4.16 sec
 ARIMA(1,0,0)(2,1,0)[8] intercept   : AIC=210764.622, Time=48.40 sec
 ARIMA(1,0,0)(3,1,0)[8] intercept   : AIC=209221.182, Time=209.67 sec
 ARIMA(1,0,0)(4,1,0)[8] intercept   : AIC=208406.755, Time=379.70 sec
 ARIMA(1,0,0)(5,1,0)[8] intercept   : AIC=208278.783, Time=844.65 sec
 ARIMA(1,0,0)(5,1,1)[8] intercept   : AIC=inf, Time=800.54 sec
 ARIMA(1,0,0)(4,1,1)[8] intercept   : AIC=inf, Time=522.68 sec
 ARIMA(0,0,0)(5,1,0)[8] intercept   : AIC=242094.924, Time=248.17 sec
 ARIMA(2,0,0)(5,1,0)[8] intercept   : AIC=207896.792, Time=650.15 sec
 ARIMA(2,0,0)(4,1,0)[8] intercept   : AIC=208408.554, Time=402.74 sec
 ARIMA(2,0,0)(5,1,1)[8] intercept   : AIC=inf, Time=660.84 sec
```

Fig. 14. ARIMA Grid Search with AIC D2

DISCUSSION AND FUTURE WORK

Model Robustness: DQN models often performed well in bull markets as compared to bear markets where the commodity's moving average continues to drop. This reflects a tendency to move slowly on rebuilding portfolio constitution even as an asset's value spikes downward. Perhaps a hybrid model consisting of hard-coded indicators for downward movement and sell signals along with a NN for increasing or bull markets can be implemented.

Further Model Comparison: Future research may look at how a high performing RL-based bot compares to one using supervised learning methods such as an LSTM. It is possible that the output of a time-series prediction model could be added as an input feature to current state information to give the agent better inference of future states.

Real-time API Deployment: This feature is currently being worked on, and will deploy an indicator-based bot to make commission-free trades using the free Python API's provided by financial services companies Robinhood and Alpaca. Alpaca also allows for the use of a "paper-money" account whereas Robinhood only accepts real fund orders.

Smoothing Data: All the datasets used in this paper were at a 1-hour timescale, whereas data sampled per day would likely smooth the data and allow for higher performing models. A daily script would also avoid day-trading fees or extra commission fees.

CONCLUSION

This paper presents a study into the advantages of using recently presented deep models as automated trading agents exchanging cryptocurrencies and USD. A state-of-the-art trading bot using a Deep-Q Network was implemented and compared to traditional indicator based agents. The DQN was inspired by Google Deepmind's recent results of using such a network to play Atari 2600 games at a super-human level using reinforcement learning.

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. 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.

REFERENCES

[1] J. H. Hayden, *RSI: the complete guide*. Greenville, SC: Traders Press, 2004.

[2] Sepp Hochreiter and J¨urgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8), 1997.

[3] "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].

[4] "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].

[5] 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].

[6] 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].

[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].

[9] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph WaveNet for Deep Spatial-Temporal Graph Modeling," *arXiv.org*, 31-May-2019. [Online]. Available: https://arxiv.org/abs/1906.00121. [Accessed: 09-Dec-2020].

[10] Hubel, D. H.; Wiesel, T. N. (1968-03-01). "Receptive fields and functional architecture of monkey striate cortex". The Journal of Physiology. 195 (1): 215–243. doi:10.1113/jphysiol.1968.sp008455. ISSN 0022-3751. PMC 1557912. PMID 4966457.

[11] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys, "Deep Q-learning from Demonstrations," *arXiv.org*, 22-Nov-2017. [Online]. Available: https://arxiv.org/abs/1704.03732. [Accessed: 20-Dec-2020].

[12] S. Pulagam, "Time Series forecasting using Auto ARIMA in python," *Medium*, 27-Jun-2020. [Online]. Available: https://towardsdatascience.com/time-series-forecasting-using-auto-arima-in-python-bb83e49210cd. [Accessed: 01-Jan-2021].

[13] SharmaVidhiHaresh, "SharmaVidhiHaresh/Backtesting-Trading-Strategies-with-Python," *GitHub*. [Online]. Available: https://github.com/SharmaVidhiHaresh/Backtesting-Trading-Strategies-with-Python. [Accessed: 01-Jan-2021].

[14] A. King, "Getting Started¶," *TensorTrade 1.0.1-beta documentation*. [Online]. Available: https://www.tensortrade.org/en/latest/overview/getting_started.html. [Accessed: 02-Dec-2020].