Algorithmic Trading with an Ensemble of Deep Recurrent Q-Networks

Colin Snyder
UC Berkeley
colinsnyder3@berkeley.edu

Julian Faust
UC Berkeley
julian.faust@berkeley.edu

***Abstract*—This research paper presents a novel approach using deep reinforcement learning to solve the algorithmic trading problem of determining the optimal trading position at a point in time. Our approach, which incorporates and combines elements from several other papers, is a variant of the popular Deep Q-Network (DQN) algorithm and is modified significantly for algorithmic trading. We train our model using historical OHLCV stock market data from 2013-2017, validate on data from 2018, and test our model on data from 2019. Some of our changes include: using an ensemble-based approach to determine actions, a dueling network structure with an LSTM layer at the end, inspired by [1] and similar to the DRQN algorithm from [2], to help resolve partial observability issues in the financial time-series data. Our model also features an extended observation space including index and commodity prices, along with financial indicators. We test exclusively on stocks from the S&P 500 Consumer Staples Index. On the 27 stocks evaluated, our model delivered promising results, outperforming passive investment strategies.**

***Index Terms*—Artificial intelligence, deep reinforcement learning, algorithmic trading**

I. Introduction

In recent years, there has been a greater effort to frame the problem of algorithmic trading in the context of deep reinforcement learning. One major benefit of this framework is that deep reinforcement learning algorithms have shown the ability to solve complex sequential decision-making problems, as seen by the benchmark performance of such algorithms on games like Atari, Montezuma's Revenge, and others [2][3]. Our research paper is structured as follows. First, a brief review of the scientific literature around the uses of deep reinforcement learning in the algorithmic trading field is presented. Next, in Section 3, we introduce and formalize the particular algorithmic trading problem considered, and how it connects to the DQN objective. Section 4 describes the design of our trading strategy in complete detail. Section 5 describes our methodology to assess the performance of trading strategies. In Section 6, we present and discuss the results achieved by our trading strategy. Finally, Section 7 draws conclusions about our results and discusses future directions in which our paper can be extended.

II. Literature Review

As noted in [4], many scientific works in the field of algorithmic trading, especially by private FinTech firms, are not publicly available due to the huge amount of money at stake. However, the proliferation of deep reinforcement learning has led to many recent publications offering different approaches to the algorithmic trading problem. Due to the massive volume of papers, we will focus on the papers that relate most and have the most pertinence to our specific algorithm design. In [4], the authors introduced their TDQN algorithm, which adapts the DQN algorithm to the algorithmic trading problem by implementing elements such as Double DQN, Huber loss, Xavier initialization, and other regularization, preprocessing, and data augmentation techniques. In order to tackle the problem of partial observability, [2] modifies the DQN algorithm to DRQN algorithm with an LSTM layer at the end of its network structure, and [5] implements this in the context of foreign exchange trading. Ensemble methods have also been used in the context of algorithmic trading, as [6] uses an ensemble strategy using three different actor-critic algorithms. Additionally, there have been attempts to focus a paper's results on a particular industry as our paper does with the consumer staples industry, as [7] focuses on the application of DRL techniques to algorithmic trading specifically for stocks in the field of energy.

III. Problem Description

A. RL Objective

Reinforcement learning is concerned with the sequential interaction of an agent with its environment. At each time step t, the RL agent retrieves an observation o_t, executes the action a_t resulting from its RL policy $\pi(a_\mathrm{t}|h_\mathrm{t})$, where

$$h_\mathrm{t} = \{(o_\tau, a_\tau, r_\tau) \mid \tau = 0, 1, ..., t\}$$

is the RL agent history, and then receives a reward r_t as a consequence of its action. Our objective is to maximize the expected discounted sum of rewards over an infinite time horizon, represented by the following equations, where γ is called the discount factor and tuned to give the optimal tradeoff between short-term and long-term behavior.

$$\pi^* = \arg\max_{\pi} \mathbb{E}(R \mid \pi)$$

$$R = \sum_{t=0}^{\infty} \gamma^t r_t$$

B. Observations

At every time step t, the RL agent observes the stock market whose internal state is $s_t \in S$. However, the agent is only able to collect limited information on this complex trading environment, and instead of having access to the complete state s_t, it receives $o_t \in O$. This makes the problem a Partially Observed Markov Decision Process (POMDP) rather than an Markov Decision Process (MDP), where true states are known. The observation space O should ideally include as much relevant information about the stock's price as possible. In our case, the observation space includes OHLCV data, macroeconomic indicators, commodity prices, and useful technical indicators. We include a total of 12 features:

- Closing price of the stock (unadjusted closing price, as our model makes no considerations about dividends)
- Volume of stock bought
- Closing price of the S&P 500 Index ($SPX)
- Closing price S&P 500 Consumer Staples Index ($SRCS)
- Closing price of gold (GCY00)
- Closing price of crude oil (CLY00)
- Closing price of Sugar #11 (SBY00)
- Yield of 30-Year US Treasury Bill ($TYX)
- MACD: Moving Average Convergence Divergence (calculated using closing price)
- RSI: Relative Strength Index (calculated using closing price, window of 10 days)
- CCI: Commodity Channel Index (calculated using high, low, and closing price, window of 10 days)
- ADX: Average Directional Index (calculated using high, low, and closing price, window of 10 days)

C. Actions

In the scope of our research paper, the trading frequency used is daily, meaning that the trading agent takes an action once every day. We propose the following reduced action space $A = \{0, 1\}$, similar to the one used in [4]. Additionally, we apply a constraint that our agent can only hold a single stock at a time in its inventory. This constraint also allows us to easily calculate daily returns for the Sharpe ratio, a metric to measure performance and described later in the paper. Now, at each time step, if the agent believes the stock will increase in price, it will take action 0, either buying if it has no stock in its inventory, or holding the stock if it does. If the agent believes the stock will decrease in price, it will take action 1, either taking no action if it has no stock in its inventory, or selling the stock if it does. This allows us to assess our agent's prediction ability on a stock's price without having to deal with the significant challenges presented by portfolio management. It also gives an intuitive and sensible way to calculate rewards as seen in the following subsection.

D. Rewards

The setup of our action space lends itself to a fairly straightforward means of determining rewards. If the agent takes action 0 (believes the stock price will increase), the reward is the closing price at time $t+1$ minus the closing price at time t, normalized by dividing by the initial closing price. If the agent takes action 1 (believes the stock price will decrease), the reward is the closing price at time t minus the closing price at time $t+1$, normalized by dividing by the initial closing price. This reward setup incentivizes the agent to take action 0 at time t if it thinks the price at time $t+1$ will be higher, and to take action 1 otherwise. Since the rewards are equal to the daily returns of the agent's predictions, maximizing the reward links closely to maximizing the Sharpe ratio, as seen in the next subsection. It is important to note that our model omits trading costs entirely, meaning our setup is meant to assess the agent's prediction ability on stock prices rather than to build an practical real-life implementation of a trading algorithm.

E. Trading Objective

An ideal trading strategy needs to effectively balance generating profits and mitigating risk. For this reason, the Sharpe ratio is the performance indicator most often used in the fields of finance and algorithmic trading. It is calculated as follows:

$$S_r = \frac{\mathbb{E}(R_s - R_f)}{\sigma(R_s - R_f)}$$

where:

- R_s is the trading strategy return
- R_f is the risk-free return

In order to calculate the Sharpe ratio, the daily returns achieved by the trading strategy are computed as they were in the prior section. Then, the returns are averaged and multiplied by the number of trading days in a year (252) to get the annual return mean $\mathbb{E}(R_s)$. Next, the annual risk-free return (taken to be 0.02 in our calculations) is subtracted to get the excess annual return mean $\mathbb{E}(R_s - R_f)$. To get the excess annual return standard deviation, we just take the standard deviation of the daily returns, and multiply by the square root of 252. Finally, we take a ratio of the excess annual return mean and standard deviation to get the Sharpe ratio. As noted by [4], although the real objective is the maximisation of the Sharpe ratio, the RL algorithm used actually maximises the discounted sum of rewards on an infinite time horizon. In [4], the authors state that this optimisation criterion can be seen as a relaxation of the Sharpe ratio criterion. Narrowing the gap between these two objectives and further investigating this link would be an important step to further formalizing the algorithmic trading problem as it pertains to deep reinforcement learning.

IV. Algorithm Design

This paper presents a deep reinforcement learning algorithm that attempts to solve the trading problem previously introduced. It uses an ensemble of 5 models, and runs a version of DRQN adapted to the problem of algorithmic trading with batch training on an unweighted replay buffer. We denominate the resulting algorithm the Ensemble Trading Deep Recurrent Q-Network (Ensemble TDRQN) algorithm.

A. Deep Q-Network Learning for MDP's

Q-learning is a type of model-free reinforcement learning that attempts to estimate the value of the actions in order to determine the optimal action to take. The Deep Q-Network (DQN) algorithm is a modification of Q-learning that uses a Deep Neural Network (DNN) to represent Q-values and an experience replay buffer with iterative updates to the Q-function via updates to the DNN's parameters. Via the experience replay buffer (sampled i.i.d for batch training), the algorithm is considered to be off-policy. For the sake of brevity, the DQN algorithm is not extensively presented in this paper. The original publications ([9] and [10]), as well as subsequent papers ([8] and [11]), describe this algorithm in sufficient depth.

B. Deep Recurrent Q-Network Learning for POMDP's

Because DQN evaluates the current state with no information about previous states, when states are said to be partially observed on some or all time steps, it cannot make inferences using observations on temporally close states that include some of the unobserved information. DRQN, as described in [2], attempts to solve this problem by adding an LSTM layer at the end of the neural network, where the output of the LSTM cell at one time step is used to help compute the output of the LSTM cell at the next time step. This allows the agent to uncover patterns hidden in the time series data.

C. Hidden-States and Cell-States

The nature of the RNN architecture means that the algorithm must now involve the manipulation of hidden-states and cell-states so that during training the model can be returned to the recurrent state it was at when it originally saw the sample. Further work on the algorithm might also include updates to the hidden states after the observations are evaluated again in training.

For each observation, which includes the 10 time steps leading up to the current one, the LSTM layer automatically handles the progression of the hidden-states and cell-states. However, unless otherwise specified, the model starts each sample at a zero start state. Instead, when the model is initialized, corresponding zero hidden-states and cell-states are initialized as well. These are passed in along with the first observation and action to the Q-function, which outputs a Q-value and the next hidden-states and cell-states. These are then passed in when evaluating Q-values for the next observation, and so on.

D. Modifications

- **Ensemble learning:** In order to address statistical error and decrease overfitting to incorrect nonlinear functions, rather than training a single model and target network, two ensembles of 5 networks each were trained, one ensemble for the models and one for the target networks. The ensemble of models tallies its votes for each action and selects the action with the majority of the votes.
- **Deep Neural Network architecture:** Each model in the ensemble uses the dueling network architecture of [8], with an LSTM layer after the dense stack, similarly to [1]. For the dueling network, two separate streams are created using dense layers learning the value and advantage estimates, respectively. These layers are then aggregated to add the value and advantage estimates to get the Q value estimates. Splitting value and advantage functions is advantageous because in every update of the Q values in the dueling architecture, the value stream V is updated, while in a single-stream architecture only the value for one of the actions is updated. This more frequent updating of the value stream in our approach allocates more resources to V, and thus allows for better approximation of the state values, whose accuracy is essential for good performance in Q-learning [8]. The exact network architecture used for our models is as follows:
  - Dense layer (size 128)
  - Dense layer (size 128)
  - Dense layer (size 64)
  - Dense layer (size 1), Dense layer (size 2)
  - Lambda layer (size 1), Lambda layer (size 2)
  - Aggregation
  - LSTM layer (size 32)

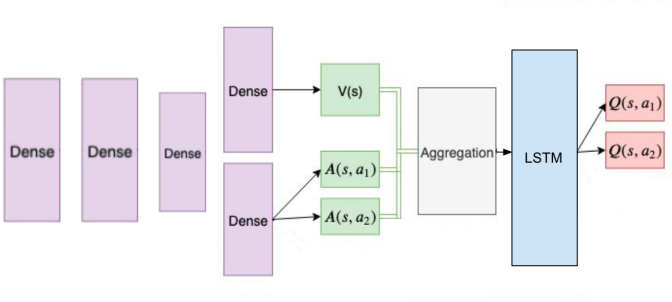

Fig. 1. Dueling Network Architecture with LSTM Layer

- **Target networks:** DQN algorithms commonly have instability and potential Q-value overestimation problems that stem from the fact that the same network used to approximate reward of potential actions on the current time step is also used to approximate maximum future reward. To address this issue, we used an ensemble of target networks to evaluate maximum future reward, which updates to the current ensemble of models every 100 steps.
- **Baseline:** Also to address potential overestimation of future Q-values, we subtract from the estimation of maximum future rewards an estimation of average future rewards. This modification attempts to teach the model not just to improve with respect to a zero sum but to improve with respect to its current learning.
- **He normal initialization:** Instead of initializing the weights completely randomly like in the original DQN algorithm, we use He normal initialization, which uses

random weights that differ in range depending on the size of the previous layer of neurons. improve the algorithm convergence.

- **Huber loss:** As in [4], we used Huber loss rather than the MSE loss used in the original DQN algorithm.
- **Decaying epsilon:** We used decaying epsilon-greedy exploration, with an initial epsilon of 1.0, decay of 0.995, and minimum epsilon of 0.001.
- **Other hyperparameters:** For our results, we trained on 3 episodes, with a batch size of 32, discount factor (γ) of 0.5, and learning rate of 0.001. As seen by the experimental results from [4], showing improvements in test results for training on episodes up to around episode 20, our results may actually further improve with more episodes of training, but due to computation time constraints we were only able to run 3 episodes for each stock due to our sizeable testbench.

V. Results

In order to reach robust conclusions, this research paper has a testbench of 27 of the 32 stocks in the S&P 500 Consumer Staples Index (all stocks in the index with valid data reaching back to the start of 2013). For each stock, we train a separate model, but all the algorithm hyperparameters remain unchanged over the entire testbench.

TABLE I
Testbench

	Company Name	Ticker
1	Altria Group	MO
2	Archer Daniels Midland	ADM
3	Campbell Soup Company	CPB
4	Church & Dwight Company	CHD
5	Clorox Company	CLX
6	Coca-Cola Company	KO
7	Colgate-Palmolive Company	CL
8	Conagra Brands Inc	CAG
9	Costco Wholesale	COST
10	Estee Lauder Companies	EL
11	General Mills	GIS
12	Hershey Foods Corp	HSY
13	Hormel Foods Corp	HRL
14	J.M. Smucker Company	SJM
15	Kellogg Company	K
16	Kimberly-Clark Corp	KMB
17	Kroger Company	KR
18	Mccormick & Company Inc	MKC
19	Mondelez Intl Inc	MDLZ
20	Monster Beverage Cp	MNST
21	Pepsico Inc	PEP
22	Philip Morris International Inc	PM
23	Procter & Gamble Company	PG
24	Sysco Corp	SYY
25	Tyson Foods	TSN
26	Wal-Mart Stores	WMT
27	Walgreens Boots Alliance	WBA

Finally, the trading horizon is the seven year span from 2013-2019 and is divided into both training, validation, and test sets as follows:

- Training set: 01/01/2013 $\rightarrow$ 12/31/2017.
- Validation set: 01/01/2018 $\rightarrow$ 12/31/2018.
- Test set: 01/01/2019 $\rightarrow$ 12/31/2019.

In addition to calculating the Sharpe ratio for each of these stocks, we calculate the Sharpe ratios for two passive investment strategies, Buy and Hold (B&H), and Sell and Hold (S&H). Sharpe ratios for these strategies are calculated in the same way as they were for our trading strategy.

TABLE II
Sharpe Ratio Results

	Ticker	Our Sharpe	B&H Sharpe	S&H Sharpe
1	MO	**1.5192**	0.0951	-0.2548
2	ADM	**1.7272**	0.6184	-0.8177
3	CPB	**2.6041**	1.8052	-1.9710
4	CHD	0.3147	**0.4406**	-0.6403
5	CLX	**1.4951**	0.0728	-0.2847
6	KO	**1.8480**	0.9350	-1.1690
7	CL	0.3049	**0.8867**	-1.1345
8	CAG	0.7584	**1.4294**	-1.5397
9	COST	1.7321	**2.2105**	-2.4599
10	EL	1.2939	**1.3718**	-1.8813
11	GIS	1.3454	**1.7774**	-1.9932
12	HSY	0.6745	**1.9984**	-2.2439
13	HRL	**1.1380**	0.4628	-0.6831
14	SJM	**1.2997**	0.5048	-0.6956
15	K	**2.4835**	0.9296	-1.1123
16	KMB	0.3474	**1.0573**	-1.2619
17	KR	0.1483	**0.2826**	-0.4235
18	MKC	**1.7276**	1.1008	-1.2957
19	MDLZ	1.8504	**2.0449**	-2.3049
20	MNST	**1.2040**	1.0523	-1.2025
21	PEP	1.1035	**1.5367**	-1.8248
22	PM	0.6522	**1.0062**	-1.1713
23	PG	0.7571	**1.8627**	-2.1051
24	SYY	**2.5913**	1.9986	-2.2534
25	TSN	0.9834	**2.2445**	-2.4098
26	WMT	1.1565	**1.6141**	-1.8926
27	WBA	-0.1223	-0.4650	**0.3172**
	Average	**1.2199**	1.1435	-1.4059

Our results were overall quite promising, as our strategy outperformed both passive investment strategies despite operating in a very bullish market (as seen by the excellent 1.1435 mean Sharpe ratio for B&H and poor -1.4059 mean Sharpe ratio for S&H). Although there were only 12 stocks where our strategy's Sharpe ratio outperformed the B&H strategy's Sharpe ratio, compared to 15 stocks where the B&H strategy had a higher Sharpe ratio, the mean Sharpe ratio of our strategy was still higher. Our results are excellent when compared to those obtained by the TDQN algorithm from [4], which achieved a mean Sharpe ratio of 0.396 on a similarly sized testbench. Of course, it is important to note that the overall high Sharpe ratio of our strategy is somewhat attributable to the overall trend of price increasing over time for the stocks in our testbench, as the testbench from [4] had a mean Sharpe ratio for the B&H strategy of only 0.375.

However, there is some evidence that our algorithm was able to find real patterns in the data, and that the positive Sharpe ratio results were not just a consequence of the bullish consumer staples market in 2019. In fact, our strategy's Sharpe ratios had a Pearson's Correlation with the B&H strategy Sharpe ratios of only 0.35189, meaning it frequently performed well in cases where the B&H strategy performed poorly, and vice

Algorithm 1 Ensemble TDRQN

Initialize the ensemble of model networks
Initialize the ensemble of target networks
for episode = 1 to M **do**
  Initialize zero hidden and cell states as h_0 , c_0
  Retrieve o_1 from the environment
  for t = 1 to N **do**
    Select a random action a_t from the action space with probability ϵ'
    Otherwise, select a_t as the mode of the actions returned in the following:
    for i in ensemble **do**
      Calculate

$$a_{i,\,t} = \arg\max_{a} Q_i(o_t, a, h_{i,t-1}, c_{i,t-1})$$

      and save the last hidden and cell states of each network as $h_{i,\,t}$ and $c_{i,t}$
    end for
    Interact with the environment using action a_t and retrieve observation o_{t+1} and reward r_t
    Store the experience e_t in the replay buffer:

$$e_t = (o_t, a_t, r_t, o_{t+1}, h_t, c_t)$$

    if t >= batch size **then**
      if t mod T' == 0 **then**
        Set $\theta_i^{target} = \theta_i$ for each network pair in the ensembles
      end if
      Sample from the replay buffer a batch B of randomly sampled experiences $e_t's$
      for i in ensemble **do**
        Set $a_i = \arg\max_a Q_i(o_{t+1}, a, h_{i,t}, c_{i,t})$
        Calculate

$$V_{o_{t+1}} = Q_i^{target}(o_{t+1}, a_i, h_{i,t}, c_{i,t}) - mean_a Q_i^{target}(o_{t+1}, a, h_{i,t}, c_{i,t})$$

        Set label

$$y_i = r_t + \gamma * V_{o_{t+1}}$$

      end for
      Compute gradients for each θ_i based on Huber loss and update them
      Decay learning rate ϵ
    end if
  end for
end for

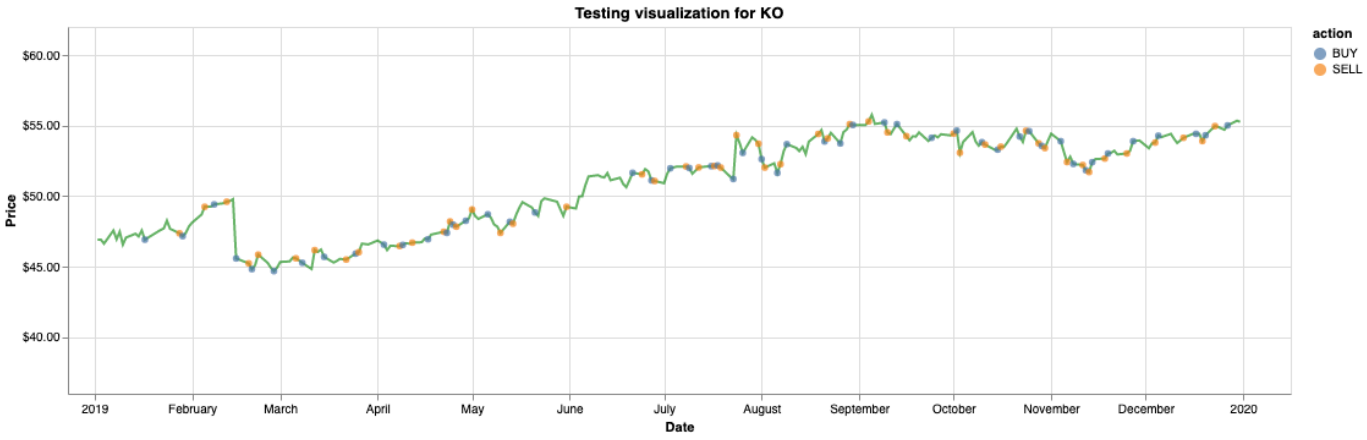

Fig. 2. Coca-Cola stock price and buy-sell action visualization

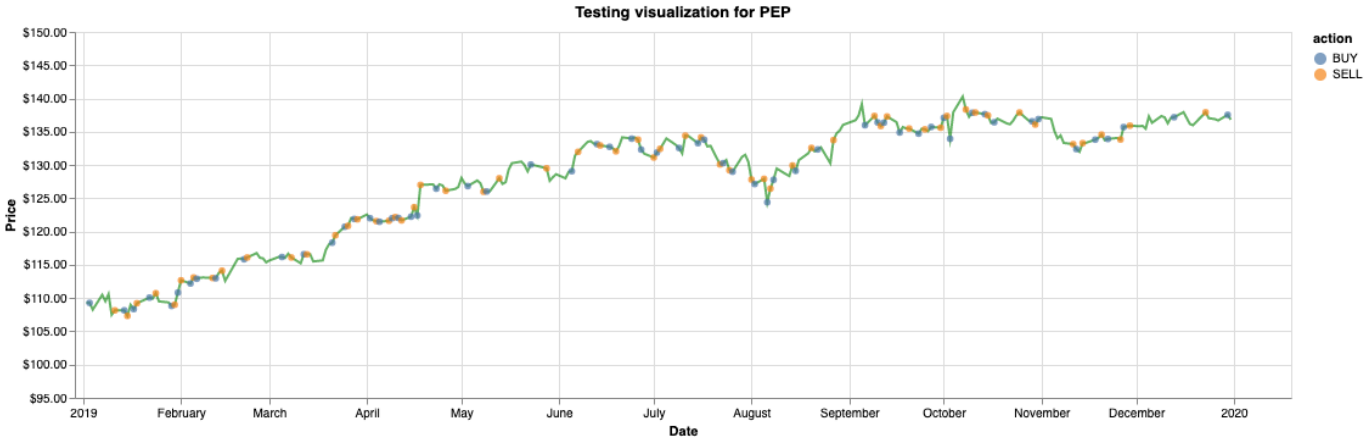

Fig. 3. Pepsico stock price and buy-sell action visualization

versa. This is in stark contrast to the results from [4], where the TDQN algorithm's Sharpe ratios had a 0.9696 Pearson's Correlation with the B&H Sharpe ratios.

TABLE III
SHARPE RATIO PEARSON'S CORRELATION

	OUR SHARPE	B&H SHARPE	S&H SHARPE
OUR SHARPE	1	0.35189	-0.41901
B&H SHARPE	0.35189	1	-0.9158
S&H SHARPE	-0.41901	-0.9158	1

Another piece of evidence to support our algorithm's effectiveness is its performance on the 11 stocks with a B&H Sharpe ratio under 1. On this filtered set of stocks, our algorithm has an impressive mean Sharpe ratio of 1.1051, while the B&H mean Sharpe ratio drops to 0.4330. This suggests that our algorithm was able to generate profits that did not just come from mirroring the B&H strategy and capitalizing on a bullish market, but rather that it was able to take advantage of patterns in the data. This also suggests that even in a poor trading year for consumer staples stocks, our algorithm may still perform quite well, although this premise needs to be tested empirically (possibly on the highly volatile trading year of 2020). Another interesting thing to note is the actual buy-sell patterns of our strategy, which changed drastically once we implemented the recurrent structure in the algorithm with the LSTM layer. Prior to adding the LSTM layer, our model essentially alternated between taking action 0 and action 1, seemingly buying and selling at random without any consistency across days. After adding the LSTM layer, our model began to take actions in a more sensible way, and there were prolonged sequences in time (each stock had a period of around 10 days) where it would consistently choose one action over another. This can also be seen in the visualizations of the trading action taken on Coca-Cola (KO) and Pepsico (PEP) stock above.

CONCLUSION

This paper presents the Ensemble Trading Deep Recurrent Q-Network algorithm (Ensemble TDRQN), an application of deep reinforcement learning to the algorithmic trading problem. Evaluated on 27 stocks in the S&P 500 Consumer Staples Index, the algorithm delivers promising results, posting a higher Sharpe ratio than passive investment strategies on average despite a test bench with a bullish market. Notably, it is able to achieve excellent Sharpe ratio performance on a filtered set of stocks with B&H Sharpe ratios under 1, demonstrating very different dynamics than the B&H strategy. The success of our algorithm can be attributed to a few factors. First, the DQN algorithm provided a solid starting point to build on, achieving decent results in the algorithmic trading problem in [4]. Second, modifying DQN to DRQN by adding an LSTM layer passing hidden and cell states changed our model's behavior significantly, as it began to consider sequences in time rather than individual steps, and was better able to process the financial time-series data. [5] had success using a similar recurrence-based approach with an LSTM layer, and our paper adds to the mounting evidence that recurrence can be a very powerful tool for the algorithmic trading problem. Third, using ensembles to determine actions proved to be an incredibly helpful tool in addressing variance in the data, a well-known conclusion in Deep RL and one backed in the trading context by [6], although this paper used a different training algorithm for each model in the ensemble. Finally, our extended observation space, with index and commodity prices, along with technical and macroeconomic indicators, contained many features that may have been helpful to determine patterns. Choosing an ideal observation space is a real challenge, since our observation space was constant across the testbench, but the optimal observation space varies depending on the stock. The observation space could be further improved by adding more commodity prices and financial indicators, along with some sort of information based on news (using Twitter volume and sentiment as features is one very interesting direction). Other changes, such as target network updates, baselines, and the dueling network structure were also essential to obtain our results. Outside of an improved observation space, the performance of our algorithm could

still be improved significantly. The easiest and most obvious way to do this would be to train on more episodes, as our training was cut short by the computational time constraint. Additionally, taking more time to properly tune certain hyperparameters would likely improve our results. Some more significant changes that may improve our model are using Double Q-Learning rather than the current target network implementation, adding artificial trajectories (as described in [4]), adding Gaussian noise to the data, properly implementing prioritized replay rather than sampling uniformly from the replay buffer, and updating the initial hidden and cell states for the start of each sample in training. Also, since our algorithm was able to effectively capitalize on the overall volatility of stocks, modifying the observation space and testing on an industry other than consumer staples might yield better results, since consumer staples is an industry with low volatility. Finally, it is also important to note that many changes still need to be made before our algorithm can be a practical trading algorithm, as we make no considerations about dividends or tradings costs, and completely omit the real challenges of portfolio management through our simplified action space.

REFERENCES

[1] S. Kapturowski, G. Ostrovski, W. Dabney, J. Quan, and R. Munos. Recurrent experience replay in distributed reinforcement learning. In International Conference on Learning Representations (2018).

[2] M. J. Hausknecht, P. Stone, Deep Recurrent Q-Learning for Partially Observable MDPs, CoRR abs/1507.06527 (2015).

[3] H. van Hasselt, A. Guez and D. Silveress. Deep Reinforcement Learning with Double Q-learning. CoRR abs/1509.06461 (2015).

[4] T. Theaté and D. Ernst. An Application of Deep Reinforcement Learning to Algorithmic Trading, CoRR abs/1507.06527 (2020).

[5] C. Yi Huang. Financial Trading as a Game: A Deep Reinforcement Learning Approach. CoRR abs/1807.02787 (2018).

[6] H. Yang, X. Liu, S, Zhong, and Anwar Walid. Deep reinforcement learning for automated stock trading: An ensemble strategy. ACM International Conference on AI in Finance (ICAIF) (2020).

[7] I. Boukas, D. Ernst, T. Th´eate, A. Bolland, A. Huynen, M. Buchwald, C. Wynants, B. Cornelusse, A Deep Reinforcement Learning Framework for Continuous Intraday Market Bidding, CoRR abs/2004.05940 (2020).

[8] Z. Wang, N. de Freitas, M. Lanctot, Dueling Network Architectures for Deep Reinforcement Learning, CoRR abs/1511.06581 (2015).

[9] C. J. C. H. Watkins, P. Dayan, Technical Note: Q-Learning, Machine Learning 8 (1992) 279–292.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, et al. Human-Level Control through Deep Reinforcement Learning, Nature 518 (2015) 529–533.

[11] H. P. van Hasselt, A. Guez, D. Silver, Deep Reinforcement Learning with Double Q-Learning, CoRR abs/1509.06461 (2015).