Deep Reinforcement Learning Techniques in Algorithmic Trading

Wang Junrui, Jeremy (A0252720N) (E0958308@u.nus.edu)

Yuvaraj Kumaresan (A0236940A) (E0758665@u.nus.edu)

Sarji Elijah See Bona (A0255894L) (sarjibona23@u.nus.edu)

AY24/25 Semester 1

Contents

1 Introduction, project overview and objectives

1.1 Financial markets and trading

Financial markets have been an integral part of human society since ancient times. Today, these markets have a value measured in the hundreds of trillions, with global equity markets alone being worth 123.82 trillion USD (WFE, 2024). Millions of investors buy and sell a multitude of assets such as stocks, currencies and commodities in these markets. This act of buying or selling assets is known as trading. The trading of these assets enables the global financial system to function.

Investors make trading decisions in an attempt to maximize such returns while taking on acceptable risk. With the rise in computing power and low latency communication in recent years, many investors have turned to automating such analysis and decision making using computers and algorithms. (The Economist, 2019)

1.2 Forecasting, beyond forecasting and reinforcement learning

Researchers have been attempting to automate analysis by using time series forecasting, using methods ranging from traditional statistical methods (Lin et. al., 2012) to machine learning approaches (Calvacante et. al., 2016) and eventually deep learning approaches (Sezer et. al., 2020).

However, automating analysis is only one component of building an automated trading agent. Investors still need to decide on how to automate the act of trading itself (Calvacante et. al., 2016). **This is where reinforcement learning models are useful**. Reinforcement learning models automate analysis and decision making by having the agent decide directly upon the best trade to take given any particular state of the market (Zhang, Zohren & Roberts, 2019).

1.3 Project overview and objectives

In this paper, we aim to explore in detail reinforcement learning techniques used in algorithmic trading by reviewing existing literature on the topic. We begin by introducing the concept of a Markov Decision Process (MDP) and reviewing how existing research models trading as a MDP. Then, we introduce the different types of reinforcement learning models used in algorithmic trading as well as notable works which makes use of these models. We will then be presenting our review of 2 papers in depth, **Financial Trading as a Game: A Deep Reinforcement Learning Approach** (Huang, 2018) and **Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy** (Yang, Liu, Zhong & Walid, 2020).

2 Modeling of trading problems as Markov Decision Processes

2.1 Overview of Markov Decision Processes

A **Markov decision process** (MDP) is a mathematical framework used to model decision-making problems where the outcomes are **stochastic**, i.e., partly random and partly within the control of a decision making agent. The goal of an MDP is to provide a way for the agent to choose the best action at each step to achieve a long-term goal, despite uncertainty about the future. An MDP consists of:

- **States**: S - Environments, $s \in S$, that the agent can find itself in.
- **Actions**: A - Actions, $a \in A$, which an agent can choose to take.
- **Transitions**: P - Probabilities, $P_a(s, s') \in P$, of reaching a state s' from a prior state s by performing action a.
- **Rewards**: R - Incentives, $r(s, a, s') \in R$ for reaching a state s' from a prior state s by performing action a.

2.2 Turning Trading into MDPs: States, Actions and Rewards

States: In our literature review, when researchers attempt to capture a trading problem as a MDP, a state typically includes the following components: non-market features, market-features and trading position of the agent. **Non-market features** are environmental parameters that are not directly related to the market e.g time, date and day of week. **Market features** are parameters obtainable from the market, such as the OHLCV (Open, High, Low, Close, Volume) data of a particular asset. The **trading position** of the agent are the assets currently owned by the agent.

Actions: Actions typically represent the possible decisions the trading agent can make at any state. Most studies we reviewed typically have 3 types of actions, buying (going long), holding (going neutral) or selling (going short) on an asset.

Rewards: We found that rewards are defined in various ways, such as the profit or loss from executing a trade, the portfolio's value at the end of a trading period, or risk-adjusted returns. But typically, they are defined as a value related to the difference in portfolio value before and after an action, incentivizing the agent to maximize returns. To better model markets, some models might also penalize actions leading to significant losses or excessive risk, incorporating factors like transaction costs, slippage, or volatility (Chan, E., 2009)

2.3 Limitations and challenges in modelling trading as MDPs

High Dimensionality and Complexity: The state and action spaces for trading can be vast. In real life, humans are able to perform a large amount of actions on financial markets involving a many different types of assets. Furthermore, a wide range of market indicators are used for technical analysis by humans in the real world. Managing this high dimensionality requires feature selection or dimensionality reduction techniques to avoid computationally intractable models, this introduces an element of human decision making into the loop and mean that these agents are not true replicas of human traders.

Transaction Costs and Market Impact: Real-world trading involves costs like transaction fees and slippage, which can significantly impact profitability. MDP models need to account for these factors to produce realistic results. However, accurately predicting these costs is challenging, as they vary based on liquidity and trading volume.

Risk Management: Trading strategies need to balance risk and reward, as high returns are often associated with increased volatility. Incorporating risk management into MDPs can be complex, as it requires carefully defined reward functions and constraints to prevent excessive risk-taking.

3 Reinforcement learning approaches for trading MDPs

After turning trading into MDPs, we now look at the reinforcement learning models employed by researchers to optimize such MDPs. In the context of trading, these reinforcement learning models all attempt to train the agent to make the best decision/action possible given any particular state in order to maximize trading returns. These reinforcement learning models generally fall into 3 categories, Critic-only, Actor-only and Actor-Critic models (Zhang, Zohren & Roberts, 2019).

3.1 Critic-only models

Critic-only models are represented by on Q-Learning and its deep learning counterpart, Deep Q-Learning. Critic-only models seek to optimize the state action-value function Q, which represents how good a particular action a is in a particular state s.

$$Q^{\pi}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q(s',a')] \tag{1}$$

The goal is to find an target Q function Q^* that satisfies the Bellman optimality equation:

$$Q^*(s,a) = \mathbb{E}[R(s,a) + \gamma \max_{a'} Q^*(s',a')] \tag{2}$$

From Q^*, the optimal policy, the best action given a particular state, π^* can be obtained:

$$\pi^*(s) = \arg\max_{a} Q^*(s,a) \tag{3}$$

In Q-Learning, the Q function is updated iteratively via direct computation until convergence via the equation below. Note that the term $R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)$ is called the Bellman error with α being the learning rate.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{4}$$

However, when state and action spaces are large, it could be unfeasible to compute all Q values directly. Therefore, researchers employ deep Q-Learning, where instead of deriving an exact function, the Q function is approximated using gradient descent. Deep Q-Learning typically employs 2 neural networks, a main network Q_θ and a target network Q_{θ^-}. The main network first computes an estimate of the Q function of a particular state and action, denoted $Q_\theta(s,a)$. The target network computes an estimate of the target Q value, given: (Fan et. al., 2020)

$$Y(s,a) = R(s,a,s') + \gamma \max_{a'} Q_{\theta^-}(s',a') \tag{5}$$

Gradient descent (minimization) is then performed on the mean-squared Bellman error, given (for n Q values):

$$L(\theta) = \frac{1}{n} \sum_{s \in S, a \in A} [Y(s,a) - Q_\theta(s,a)]^2 \tag{6}$$

During training, the target network parameters θ^- are updated with the parameters of the main network θ every few episodes of training at a rate τ:

$$\theta^- \leftarrow (1-\tau)(\theta^-) + \tau(\theta) \tag{7}$$

By minimizing the Bellman error, our predicted Q-value converges on the target value Y, equivalent to converging on values that would satisfy the Bellman optimality equation. In other words, our Q function converges on the target function Q^*.

Notably however, there are some issues with Deep Q-Networks, which suffers from variability during training. Some proposed solutions include Double Q-Learning, which aims to reduce overestimations of Q^* by decomposing action selection and action evaluation (Hasselt, H.V., Gaez, A. & Silver, D., 2016). This is done by writing:

$$Y(s,a) = R(s,a,s') + \gamma Q_{\theta^-}(s', \arg\max_{a'} Q_\theta(s',a')) \tag{8}$$

Compared to regular deep Q-learning (equation 6), where selection and evaluation of the best next action a' are both performed by the target network Q_{θ^-} through $\max_{a'} Q_{\theta^-}(s',a')$, double Q-learning decouples action selection and evaluation by having the main Q-network Q_θ select a' using $\arg\max_{a'} Q_\theta(s',a')$ before having the target Q-network evaluate it.

3.2 Actor-only models

Actor-only models such as the REINFORCE algorithm focus on applying the Policy Gradient Theorem and Monte Carlo methods to optimize the policies of the MDP directly without value functions in order to obtain π^*. In these models, a parameterized policy π_θ is optimized to maximize the expected returns, denoted by $J(\pi_\theta)$: (OpenAI, n.d.)

$$J(\pi_\theta) = \mathbb{E}[R(\tau)|\pi_\theta] \tag{9}$$

τ represents a trajectory, sequence of actions and states, whilst $R(\tau)$ is the summation of rewards across the trajectory. Gradient ascent is performed to optimize θ: (OpenAI, n.d.)

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)|_\theta \tag{10}$$

The term $\nabla_\theta J(\pi_\theta)$ is referred to as the policy gradient, and is defined as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right] \tag{11}$$

The term $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is the log-gradient term, which represents how much a small change in the policy parameters θ affects the probability of the action a_t taken in the state s_t. By weighing the log-gradient term with $R(\tau)$, trajectories with higher rewards are emphasized. Actions that resulted in higher rewards get a stronger influence on the gradient update. In practice, $\nabla_\theta J(\pi_\theta)$ is approximated using the sample mean $\hat{g}$, obtained by collecting a set of trajectories $\mathcal{D} = [\tau_i]_{1,\ldots,N}$ and then performing the following computation: (OpenAI, n.d.)

$$\hat{g} = \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) \tag{12}$$

3.3 Actor-Critic models

Finally, Actor-Critic models have features of both actor-only and critic-only models. Examples of which include Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO) and Deep Deterministic Policy Gradient (DDPG).

In A2C models, the policy π_θ is updated in real time by employing 2 networks, an actor network that chooses actions, and a critic network that evaluates the chosen action by the policy network using the advantage function A_{adv}. The advantage function is defined as the difference in reward between the chosen action for a particular state by the policy network and average action in the state. In this case, just like in actor-only methods, we still perform gradient ascent to optimize θ, but instead of using $R(\tau)$ to weigh different actions, we use A_{adv} to weigh actions:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A_{adv}(s_t, a_t)\right] \tag{13}$$

$$A_{adv}(s,a) = R(s,a,s') + \gamma V(s'|w) - V(s|w) \tag{14}$$

In PPO models, the policy is optimized incrementally to prevent sudden, large deviations from the previous policy. By using a clipped objective function $L^{CLIP}(\theta)$ that limits how much the probability ratio $r_t(\theta)$, which is the ratio of the probability of taking an action under the new policy π_{θ_n} compared to the old policy π_{θ_o} can deviate from 1. By doing so, PPO limits the updates to the policy, ensuring that training steps are "proximal" and stable. (Schulman et. al., 2017) PPO also employs a critic network, which computes the advantage functions for the current policy. This leads to an objective function as denoted below:

$$L^{CLIP}(\theta) = \mathbb{E}\left[\min\left(\frac{\pi_{\theta_n}(a,s)}{\pi_{\theta_o}(a,s)} A_{adv}(S,A), clip\left(\frac{\pi_{\theta_n}(a,s)}{\pi_{\theta_o}(a,s)}, 1-\epsilon, 1+\epsilon\right) A_{adv}(S,A)\right)\right] \tag{15}$$

Gradient ascent is then performed on the objective function $L^{CLIP}(\theta)$ to update the policies. Gradient descent is then performed on the critic network to better fit the value functions used to compute advantages (OpenAI, n.d.).

In DDPG, the model concurrently optimizes the Q function with parameters ϕ and a policy with parameters θ. The critic network in DDPG performs Q-Learning by minimizing the mean squared Bellman Error as described in equation 6. Policy learning in DDPG involves learning a deterministic policy π_θ which gives an action that maximizes the current Q function Q_ϕ (OpenAI, n.d.). This can be done by performing gradient ascent on the parameters of the policy to solve:

$$\max_\theta \mathbb{E}_{s\in\mathcal{D}}[Q_\phi(s, \pi_\theta(s))] \tag{16}$$

4 Evaluation metrics for automated trading agents

Cumulative Return: This measures the total return over a period, providing a straightforward measure of profitability. It's computed as the difference in portfolio value from the start to the end of the trading period (Lai, K.& Cao, H., 2021).

Sharpe Ratio: The Sharpe ratio measures the risk-adjusted return of a strategy by dividing the mean return by the standard deviation of returns. This metric is valuable in evaluating whether the strategy achieves high returns without incurring excessive risk (Lai, K. & Cao, H., 2021).

Win Rate and Profit Factor: These metrics provide insights into the consistency of the strategy. The win rate measures the proportion of profitable trades, while the profit factor compares the sum of winning trade profits to the sum of losses, indicating the strategy's efficiency in generating profits relative to losses (Lai, K.& Cao, H., 2021).

Comparison against baseline strategies: The trading agent is also frequently evaluated against baseline human strategies such as the "buy and hold" strategy to compare its performance with investors using other forms of analysis to invest.

5 Review of Financial Trading as a Game: A Deep Reinforcement Learning Approach (Huang, 2018)

In this paper, a DRQN agent was trained and used to make decisions for foreign-exchange trading, performing a trading action at the beginning of every time step/day.

5.1 MDP formulation

States: The state space was modeled as a $198-$dimensional vector.
It includes a $3-$dimensional time feature: (Minute, Hour, Day) in sinusoidal encoding. Sinusoidal encoding converts values t to the form:

$$\sin\left(2\pi\frac{t}{T}\right) \tag{17}$$

T is the number of values in the domain of t. For instance, day of week in sinusoidal encoding would be in the form $\sin\left(2\pi\frac{t}{7}\right)$, where $t \in \{0, 1, 2, 3, 4, 5, 6\}$ for Monday to Sunday respectively.
The state space also includes a 192 dimensional vector for Market features. 16 features were extracted from OHLCV market data for 12 currency pairs.
The final component of the state space is a 3 dimensional position feature that represents if the agent is short, $[1, 0, 0]$, neutral, $[0, 1, 0]$ or long $[0, 0, 1]$

Actions: The agent could choose to go long, short and neutral on a currency pair.

Rewards: The reward function for the MDP is defined as the log returns of the agent's portfolio at each time step: $r_t = \log(\frac{v_t}{v_{t-1}})$, with v_t representing the agent's portfolio value at timestep t.

5.2 Reinforcement learning technique used

Double Deep Q-learning is employed and the target function which gradient descent is performed on is an action augmentation loss in the form:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}[||r + \gamma Q_{\theta^-}(s', \arg\max_{a'} Q_\theta(s', a')) - Q_\theta(s, a)||^2] \tag{18}$$

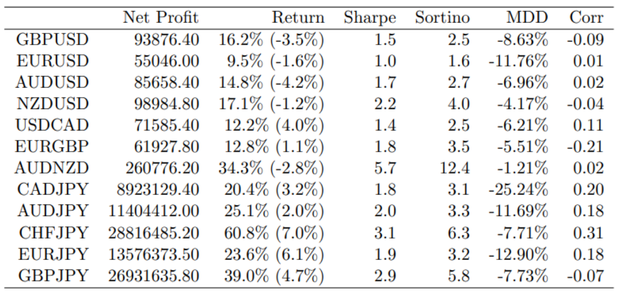
Q_{θ^-} denotes the target network. The author computes the reward signal for all possible actions. Since rewards are computed for all possible actions, the Q values for all possible actions are updated instead of just the one which was chosen by the agent.

5.3 Model architecture

The author uses a neural network with 4 layers. The first 2 are hidden linear layers of 256 neurons with ELU activation. The third layer is a LSTM layer of 256 neurons, and the fourth layer is a linear layer with 3 output units. Weight initialization was based on the schemes implemented in He et. al. (2015) and Le et. al. (2015), all biases in the network are set to 0 except for the forget gate of the LSTM, which is set to 1.

5.4 Evaluation and results

- **Evaluation metrics**: The author evaluated the agent based on annualized net profit, returns, Sharpe ratio and Sortino ratio. The net profit and returns measure directly the performance of the agent, whilst the Sharpe and Sortino ratios measure the risk adjusted returns of the agent.
- **Results**: The agent is profitable for most of the currency pairs under 0.15 basic points of spread. The author notes that profitable strategies cannot be discovered under 0.2 basic point of spread for currency pair USDCAD and EURGBP. For all currency pairs, the agent was able to obtain a Sharpe ratio of above 1, which is considered acceptable to good by investors (Maverick, 2024).



	Net Profit	Return	Sharpe	Sortino	MDD	Corr
GBPUSD	93876.40	16.2% (-3.5%)	1.5	2.5	-8.63%	-0.09
EURUSD	55046.00	9.5% (-1.6%)	1.0	1.6	-11.76%	0.01
AUDUSD	85658.40	14.8% (-4.2%)	1.7	2.7	-6.96%	0.02
NZDUSD	98984.80	17.1% (-1.2%)	2.2	4.0	-4.17%	-0.04
USDCAD	71585.40	12.2% (4.0%)	1.4	2.5	-6.21%	0.11
EURGBP	61927.80	12.8% (1.1%)	1.8	3.5	-5.51%	-0.21
AUDNZD	260776.20	34.3% (-2.8%)	5.7	12.4	-1.21%	0.02
CADJPY	8923129.40	20.4% (3.2%)	1.8	3.1	-25.24%	0.20
AUDJPY	11404412.00	25.1% (2.0%)	2.0	3.3	-11.69%	0.18
CHFJPY	28816485.20	60.8% (7.0%)	3.1	6.3	-7.71%	0.31
EURJPY	13576373.50	23.6% (6.1%)	1.9	3.2	-12.90%	0.18
GBPJPY	26931635.80	39.0% (4.7%)	2.9	5.8	-7.73%	-0.07

Figure 1: Annualized return of DRQN agent across different currency pairs (Huang, 2018)

6 Results of replication of DRQN model specified by Huang (2018)

In the future work section of his paper, Huang (2018) stated that he would like to apply reinforcement learning techniques in other trading scenarios, including long-term equity investment. Our attempt at replication performs such work.

6.1 Overview

We follow the DRQN formulation in Financial Trading as a Game: A Deep Reinforcement Learning Approach (Huang, 2018) as closely as possible while adapting it to trade individual equities and equity indexes on the New York Stock Exchange (NYSE) and NASDAQ. Training and test data ranging from $01 - 01 - 2015$ to $12 - 31 - 2023$ was obtained from YFinance. In particular, data from $01 - 01 - 2025$ to $12 - 31 - 2022$ was used as training data, whilst data from $01 - 01 - 2023$ to $12 - 31 - 2023$ was used as test data.

6.2 MDP formulation

State: We represent each state as a 11 dimensional vector. 8 market features (as the author did not specify which 16 features he used), including OHLCV data, RSI, rolling 20/50 day averages were extracted, and a

3-dimensional position feature which represents the current position of the agent. The author of the original paper included features for all 12 currency pairs within the state space as he believes that there could be correlations present in different pairs of currencies. (Huang, 2018) We make no such assumptions and trade based on market data of the selected equity or equity index itself only. Furthermore, since we are adapting the model for long term equity trading, the time feature is less important as long term value investors are unlikely to pay attention to when they make a particular trade.

Action: The action space consisted of 3 possible actions for the agent in any given state: Buy, Hold or Sell. This is similar to the original paper.

Rewards: The same reward formula specified in section 6.1 was used. Commission fees and spreads for stock trading differs from broker to broker and stock to stock. Hence, we use arbitrary fee of 0.1% of the trade value and a spread of 0.001 for spreads.

6.3 Model architecture and parameters

Reinforcement learning approach: The same reinforcement learning technique (Double Q-Learning) was implemented in our replication attempt with the same action augmentation loss to perform gradient descent on.

Model architecture: The model architecture and weight initializations follows the original paper exactly.

Training and simulation hyperparameters: We used the same training hyperparameters as the original paper, though we did experiment with different simulation hyperparameters.

6.4 Results

Asset traded	Agent best return observed	Agent average return	Market performance for asset
AAPL	6.391%	3.946%	49.67%
MSFT	2.764%	1.582%	56.06%
NVDA	63.68%	47.28%	235.6%
SPY	9.104%	7.153%	25.85%

Table 1: Performance of replicated model with trade size of 100000, 1000 episodes of training

Asset traded	Agent best return observed	Agent average return	Market performance for asset
AAPL	11.315%	6.846%	49.67%
MSFT	12.74%	7.914%	56.06%
NVDA	124.0%	85.80%	235.6%
SPY	24.93%	16.54%	25.85%

Table 2: Performance of replicated model with trade size of 250000, 1000 episodes of training

6.5 Discussion

Result evaluation: In general, the replicated model severely underperforms the market in terms of annualized returns even when we select for the best returns we observed across 10 training and test sessions. In particular, it struggles when trade sizes are smaller, for instance at 100000 as specified by the original paper. Though the agent did not make a loss in any training and test session, we attribute this to 2023 being excellent for equities, as demonstrated by the market performances of the assets we chose. Therefore, even if our agent were to make actions at random, it would still be likely to turn a profit.
Patterns observed: We noted that for many training sessions, the agent would tend towards staying neutral no matter the market state, leading to the flatlining of returns like in Figure 2. We also noted that the agent liked to case trends, leading to a very notable drawdown in many of our sessions between trading day 130 and 210, an example of which can be seen in Figure 3.

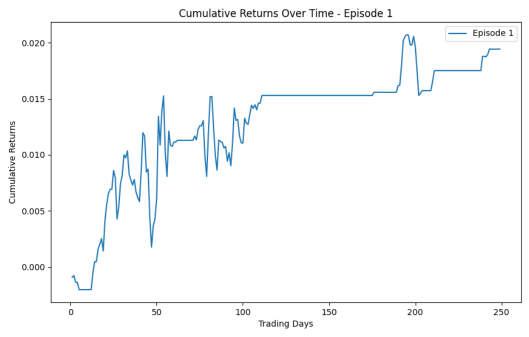


Figure 2: Agent returns on AAPL flatlining due to preference of staying neutral.

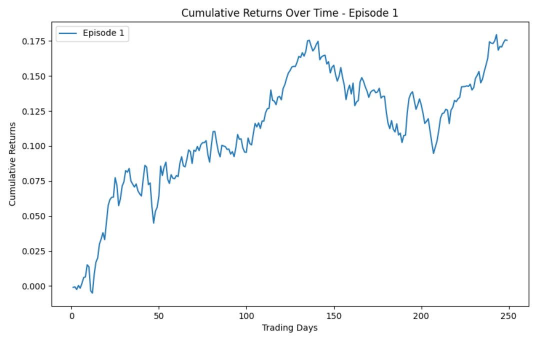


Figure 3: Agent returns on SPY showing significant drawdown between trading day 130 and 210.

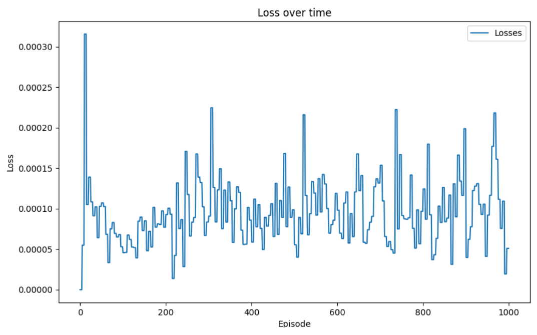


Figure 4: Training session where the action augmentation loss did not converge at 1000 episodes

Possible reasons for results and observations: We acknowledge that it is possible that we made mistakes in replicating the author's model, thus being unable to achieve satisfactory returns. If we assume that our model replication was satisfactory, some possible reasons for our observations could be that our chosen MDP formulation isn't suitable or doesn't capture market behavior well for equity trading. It is also possible that a lack of training could have played a role in our results, since we observed that loss convergence was highly unstable. Sometimes, the action augmentation loss does not converge even at 1000 episodes. Finally, we did not perform hyperparameter searching, and thus it is possible that current hyperparameters are unsuitable for our set-up.

7 Review of Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy (Yang et al., 2020)

In this paper, an ensemble trading strategy was proposed to maximize investment returns in the stock market. Here, deep reinforcement learning agents are trained using three actor-critic algorithms from the package Stable Baselines-3: Advantage Actor Critic (A2C), Deep Deterministic Policy Gradient (DDPG), and Proximal Policy Optimization (PPO). Then they are integrated into an ensemble strategy that combines the best features of each agent, which is used to trade stocks.

7.1 MDP Formulation

States: A continuous action space was used to model the trading of multiple stocks on the 30 component companies in the Dow Jones Industrial Average (DJIA). The state space was represented using a 181-dimensional vector consisting of the following components:

- $b_t \in \mathbb{R}_+$: Available balance at time t.
- $\boldsymbol{p_t} \in \mathbb{R}_+^{30}$: Adjusted close prices of each stock.
- $\boldsymbol{h_t} \in \mathbb{Z}_+^{30}$: Shares owned of each stock.

The last 120 dimensions of the 181-dimensional state vector consist of four technical indicators calculated for each of the 30 stocks: Moving Average Convergence Divergence (MACD), Relative Strength Index (RSI), Commodity Channel Index (CCI), and Average Directional Index (ADI).

Actions: For a single stock, the action space is defined as $\{-k, \ldots, -1, 0, 1, \ldots, k\}$ where $k \leq h_{max}$, the preset maximum amount of shares for each buying action. Note that a positive integer means to buy, negative means to sell, and 0 means to hold. It is then normalized to $[-1, 1]$ to accommodate A2C and PPO policy definitions, which is on a Gaussian distribution.

Rewards: The reward function is defined as the change in portfolio value as follows:

$$r(s_t, a_t, s_{t+1}) = (b_{t+1} + \boldsymbol{p_{t+1}^T} \boldsymbol{h_{t+1}}) - (b_t + \boldsymbol{p_t^T} \boldsymbol{h_t}) - c_t,$$

where c_t is the transaction cost, assumed here to be $c_t = \boldsymbol{p}^T \boldsymbol{k_t} \times 0.1\%$.

7.2 Reinforcement Learning Techniques Used

7.2.1 Advantage Actor Critic (A2C)

A2C algorithm is an actor-critic method that improves policy gradient updates by utilizing an advantage function to reduce variance. The synchronized gradient update increases training data diversity, is cost-effective, faster, and performs better with large batch sizes, which are attributes that contribute to A2C's stability in stock trading applications. This strategy was used by G et al. (2023) which demonstrated a commendable ability to effectively forecast the optimal course of action for stocks, namely whether to buy, sell, or hold, with the aim of maximizing profits.

7.2.2 Deep Deterministic Policy Gradient (DDPG)

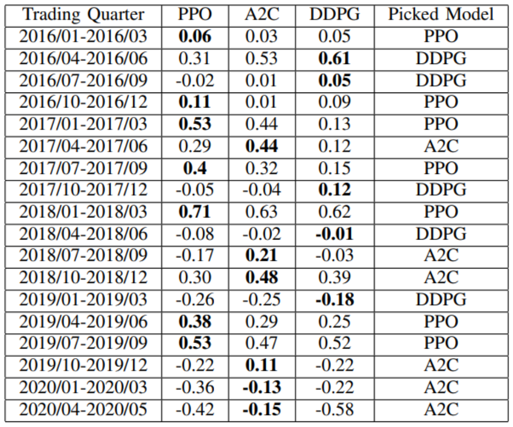
DDPG tries to maximize investment returns by effectively handling continuous action spaces, making it well-suited for trading environments. DDPG integrates the strengths of both Q-learning and policy gradient methods, employing neural networks as function approximators to learn directly from observations, thereby overcoming the curse of dimensionality associated with Q-learning approaches like DQN. It was shown in Liu et al. (2018) that a trained agent with a pure DDPG strategy outperforms the Dow Jones Industrial Average and min-variance portfolio allocation method in accumulated return. The comparison on Sharpe ratios indicates that this method is also more robust in balancing risk and return.

7.2.3 Proximal Policy Optimization (PPO)

PPO enhances the stability of policy network training in reinforcement learning by limiting the magnitude of policy updates through a clipping mechanism. This approach prevents large, destabilizing changes at each training step, resulting in a more stable and efficient learning process. PPO is selected for stock trading applications due to its stability, speed, and ease of implementation and tuning. Its ability to maintain stability while optimizing policy updates makes it a powerful tool to enhance decision-making processes in trading.

7.3 Results and Evaluation

The data from the package *yfinance* from 01/01/2009 to 09/30/2015 is used for training, and the data from 10/01/2015 to 12/31/2015 is used for validation and tuning of parameters. This 3-month validation rolling window is used to pick the best performing agent with the highest Sharpe ratio, which is used to predict and trade for the next 3 months. The agent is then tested on its performance on trading data from 01/01/2016 to 05/08/2020. The agent continues to train and validate while in the trading stage, since this helps the agent to better adapt to the market dynamics.



Trading Quarter	PPO	A2C	DDPG	Picked Model
2016/01-2016/03	**0.06**	0.03	0.05	PPO
2016/04-2016/06	0.31	0.53	**0.61**	DDPG
2016/07-2016/09	-0.02	0.01	**0.05**	DDPG
2016/10-2016/12	**0.11**	0.01	0.09	PPO
2017/01-2017/03	**0.53**	0.44	0.13	PPO
2017/04-2017/06	0.29	**0.44**	0.12	A2C
2017/07-2017/09	**0.4**	0.32	0.15	PPO
2017/10-2017/12	-0.05	-0.04	**0.12**	DDPG
2018/01-2018/03	**0.71**	0.63	0.62	PPO
2018/04-2018/06	-0.08	-0.02	**-0.01**	DDPG
2018/07-2018/09	-0.17	**0.21**	-0.03	A2C
2018/10-2018/12	0.30	**0.48**	0.39	A2C
2019/01-2019/03	-0.26	-0.25	**-0.18**	DDPG
2019/04-2019/06	**0.38**	0.29	0.25	PPO
2019/07-2019/09	**0.53**	0.47	0.52	PPO
2019/10-2019/12	-0.22	**0.11**	-0.22	A2C
2020/01-2020/03	-0.36	**-0.13**	-0.22	A2C
2020/04-2020/05	-0.42	**-0.15**	-0.58	A2C

Figure 5: Sharpe Ratios of the Three Algorithms Over Time

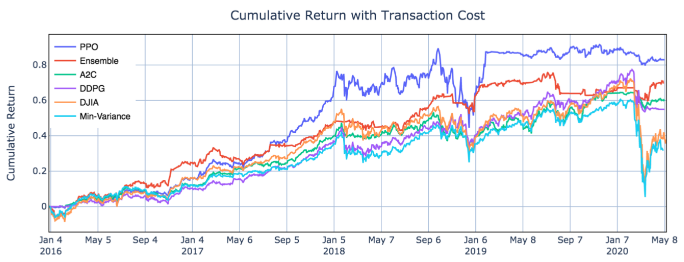


Figure 6: Cumulative return curves of our ensemble strategy compared with other strategies

As seen in Figure 5, the model picked usually does not stay the same for more than 6 months, with the exception of the last 9 months where stocks crashed due to the Covid-19 pandemic as seen in Figure 5, in which the A2C model was used. Because of this, it is observed that A2C is good at handling a bearish or declining market. Also, it is seen that PPO is preferred when facing a bullish market. DDPG performs similar but not as good as PPO, but it can be used as a complementary strategy to PPO in a bullish market. The ensemble, with a Sharpe ratio of 1.30, managed to outperform the three individual strategies and the benchmark, which is the DJIA. A2C, DDPG, PPO, and DJIA had Sharpe ratios of 1.12, 0.87, 1.10, and 0.47 respectively.

Therefore, such an ensemble strategy can be useful for making decisions on stock-trading, for it can adapt to market conditions. A possible enhancement is to add more Deep Reinforcement Learning models from Stable Baselines-3 such as Soft Actor-Critic (SAC) and Twin Delayed DDPG (TD3).

8 Responsible AI considerations

Explainable models: In both papers we reviewed researchers make use of widely known reinforcement learning techniques that are well studied and documented, and thus the behavior of both agents can be readily explained.
Thorough testing: In both papers, a controlled environment was provided to observe the model's behavior across different market conditions without affecting real financial markets. This ensures the model's strategies are sound before any potential deployment in real markets.
Ease of replication: Huang (2018) provided substantial detail including parameters used, pseudo code and model architecture that allowed for relatively easy replication of his model, whilst Yang et al (2020) provided a link to their source code repository.
Conservative trading strategies: Both papers evaluate model performance based on risk adjusted returns. The agent in Yang et al.'s (2020) paper is also constrained by a turbulence index, where the agent will sell all assets if the market becomes too volatile to minimize losses. This promotes a more cautious trading style for the agents, which may contribute to more stable trading behavior that is less likely to have large drawdowns. This is valuable should any investor wish to deploy these agents on live trading.
Open source data usage: Both papers made use of open source information that can be easily accessed, thus not providing any unseen advantages for the researchers, allowing others to replicate their research as well.

9 Appendix

9.1 Summary of other papers reviewed

In this section we present some papers we reviewed, aside from the 2 which we will be doing an in-depth dive in the latter sections.

9.1.1 Application of Deep Reinforcement Learning on Automated Stock Trading

Chen & Gao, 2019

- **Reinforcement learning model used**: Deep Recurrent Q-Network (DRQN)
- **States**: Sequences of closing price data over a sliding window of 20 days of the S&P 500 ETF.
- **Actions**: Buy shares, sell shares or do nothing
- **Rewards**: Difference between the next day's closing price and the current day's closing price
- **Evaluation and results**: The agent was able to outperform baseline strategies such as Buy & Hold and another random action-selected DQN trader, giving an annual return of about 22%.

9.1.2 An intelligent financial portfolio trading strategy using deep Q-learning

Park, Sim & Choi, 2019

- **Reinforcement learning model used**: Deep Q-Network (DQN)
- **States**: Comprised of market data and technical indicators for I different types of assets and the portfolio of the agent
- **Actions**: The agent could choose to buy, go neutral or sell any of the I assets, for 3^I total actions.
- **Rewards**: Change in the portfolio value at the end of the next period relative to the static portfolio value (if nothing was done)
- **Evaluation and results**: The DQN agent was tested against baseline strategies such as Buy & Hold, Random Selected Strategy, Momentum Strategy and Reversion strategy on two portfolios, one containing US based assets such as the SPDR S&P 500 ETF, iShares Russell 1000 Value ETF and the iShares Microcap ETF and the other containing South Korea based assets, such as the KOSPI 100 index, Midcap KOSPI index and Microcap KOSPI index. The final portfolio value of the DQN agent for the US based portfolio was 15.69% higher than that of the B&H strategy, 33.74% higher than that of the RN strategy, 21.81% higher than that of the MO strategy, and 114.47% higher than that of the RV strategy. The DQN agent was also able to outperform all baseline strategies for the Korea based portfolio as well.

9.1.3 Deep Reinforcement Learning for Trading

Zhang, Zohren & Roberts, 2019

- **Reinforcement learning model used**: Policy gradient (The paper itself had 3 different models, we choose to review the policy gradient method used)
- **States**: Market features extracted from 50 ratio-adjusted continuous futures contracts from the Pinnacle Data Corp CLC Database over a rolling window of 60 days and the agent's position.
- **Actions**: The agent could choose to go long, short and neutral.
- **Rewards**: Volatility adjusted daily profits
- **Evaluation and results**: The agent was able to outperform the baseline models such as a Long-Only strategy and classical time series momentum strategies across most asset classes except equity indexes, where a long-only strategy was the best performing.

9.2 Parameters used in inancial Trading as a Game: A Deep Reinforcement Learning Approach

Huang, 2018

The following training and simulation hyperparameters were used by the authors. T is the length of state sequences used by the authors to train the agent on, whilst the parameter τ represents the rate at which the target network is updated:

- **Learning Timestep (T):** 96
- **Replay Memory Size (N):** 480
- **Learning Rate:** 0.00025
- **Optimizer:** Adam
- **Discount Factor:** 0.99
- **Target Network (τ):** 0.001
- **Initial Cash Balance:** $100,000^2$
- **Trade Size:** $100,000$
- **Spread:** 0.08
- **Trading Days per Year:** 252

9.3 Code repository

Link to replication of models: https://github.com/Yuvaraj0702/3263research

9.4 Division of work

- Jeremy: Report sections $1, 3, 5, 6$, Replication of DRQN model specified by Huang (2018)
- Yuvaraj: Report sections $2, 4, 6, 8$, Replication of DRQN model specified by Huang (2018)
- Sarji: Report sections: $2, 5, 7$, Testing of Ensemble model specified by Yang et al. (2020)

10 References

Calvacante R.C., Brasilero R.C., Souza V.L.F., Nobrega J.P. & Oliveira A.L.I. (2016). Computational Intelligence and Financial Markets: A Survey and Future Directions. *Expert Systems With Applications, 55 (2016), 194-211.* https://doi.org/10.1016/j.eswa.2016.02.006

Chan, E. (2009). Quantitative trading: How to build your own algorithmic trading business. Wiley Trading.

Chandak et al.,(2020) *Towards Safe Policy Improvement for Non-Stationary MDPs* https://people.cs.umass.edu/ pthomas/-papers/Chandak2020b.pdf

Chatfield, C. (1975). *The Analysis of Time Series: Theory and Practice.* (5^{th} ed.). Springer-Science.

Chen, L. & Gao, Q. (2019). Application of deep reinforcement learning on automated stock trading. *2019 IEEE 10th International Conference on Software Engineering and Service Science, 29-33*
doi: 10.1109/ICSESS47205.2019.9040728.

Fan, J., Wang, Z., Xie, Y. & Yang, Z. (2020). A Theoretical Analysis of Deep Q-Learning. (arXiv: 1901.00137). arXiv.
https://doi.org/10.48550/arXiv.1901.00137

G, R., Sagar, S., Naranayan, V., Binu, D., Selby, N., & Thomas, S. (2023). Advantage Actor-Critic Reinforcement Learning with Technical Indicators for Stock Trading Decisions. SSRN. https://dx.doi.org/10.2139/ssrn.4596132

Hasselt, H.V., Gaez, A. & Silver, D. (2016). Deep reinforcement learning with double Q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence, 30(1).*
https://doi.org/10.1609/aaai.v30i1.10295

Huang, C. Y. (2018). Financial trading as a game: A deep reinforcement learning approach (*arXiv:1807.02787*). arXiv. http://arxiv.org/abs/1807.02787

He K., Zhang, X., Ren, S. and Sun, J. (2015) Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE international conference on computer vision, 1026–1034, 2015.* Doi: 10.1109/ICCV.2015.123

Lai, K.& Cao, H. (2021). Financial engineering and quantitative analysis: Metrics for algorithmic trading strategies. Journal of Financial Economics, 3(1), 27–49.

Le Q.V., Jaitly N., Hinton G.E. (2015) A simple way to initialize recurrent networks of rectified linear units. (arXiv: 1504.00941) https://doi.org/10.48550/arXiv.1504.00941

Levinson, M. (2018). *The Economist Guide To Financial Markets 7th Edition: Why they exist and how they work* (7^{th} ed.). Economist Books.

Lin, C-S., Chiu S-H. & Lin, T-Y. (2012) Empirical mode decomposition-based least squares support vector regression for foreign exchange rate forecasting. *Economic Modelling, 29 (6), 2583-2590.*
https://doi.org/10.1016/j.econmod.2012.07.018

Liu, X., Xiong, Z., Zhong, S., Yang, H., & Walid, A. (2018). Practical deep reinforcement learning approach for stock trading. arXiv.org. https://doi.org/10.48550/arXiv.1811.07522

Maverick, J.B. (2024) *What the Sharpe Ratio Means for Investors*, Investopedia.
https://www.investopedia.com/ask/answers/010815/what-good-sharpe-ratio.asp OpenAI. (n.d.) *Deep Deterministic Policy Gradient*, OpenAI

https://spinningup.openai.com/en/latest/algorithms/ddpg.html

OpenAI. (n.d.) *Part 3: Intro to Policy Optimization*, OpenAI
https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html

OpenAI. (n.d.) *Proximal Policy Optimization*, OpenAI
https://spinningup.openai.com/en/latest/algorithms/ppo.html

Park, H., Sim, M.K., Choi, D.G. (2019) An intelligent financial portfolio trading strategy using deep Q-learning. (arXiv:1907.03665). arXiv.
https://doi.org/10.48550/arXiv.1907.03665

Pricope, T-V. (2021). Deep Reinforcement Learning in Quantitative Algorithmic Trading: A Review (arXiv:2106.00123). arXiv.
https://doi.org/10.48550/arXiv.2106.00123

The Economist (2019, October). *The rise of the financial machines.*
https://www.economist.com/leaders/2019/10/03/the-rise-of-the-financial-machines

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017) Proximal Policy Optimization Algorithms (arXiv: 1707.06347). arXiv.
https://doi.org/10.48550/arXiv.1707.06347

Sezer, O.B., Gudelek, M.U. & Ozbayoglu, A.M. (2020) Financial time series forecasting with deep learning: A systemic literature review: $2005 - 2019$. *Applied Soft Computing, 90 (May 2020), 106181.*
https://doi.org/10.1016/j.asoc.2020.106181

Sigaud, O., & Buffet, O. (2013). *Markov decision processes in artificial intelligence.* John Wiley & Sons.

Thompson, C. (2024, August). *Fundamental vs. Technical Analysis: What's the Difference?* Investopedia.
https://www.investopedia.com/ask/answers/difference-between-fundamental-and-technical-analysis/

Wang, Z., Schaul, T., Hessel M., Hasselt H.V., Lanctot & M., Freitas, N.D. (2016) Dueling Network Architectures for Deep Reinforcement Learning. (arXiv: 1511.06581) arXiv.
https://doi.org/10.48550/arXiv.1511.06581

World Federation of Exchanges (2024, October). *Market Statistics - November* 2024.
https://focus.world-exchanges.org/issue/november-2024/market-statistics

Yang, H., Huang, K., King, I. & Lyu, M.R. (2009). Localized support vector regression for time series prediction. *Neurocomputing, 72 (10-12), 2659 - 2669.* https://doi.org/10.1016/j.neucom.2008.09.014

Yang, H., Liu, X-Y., Zhong, H. & Walid, A. (2020). Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy. *SSRN.*
http://dx.doi.org/10.2139/ssrn.3690996

Zhang, Z., Zohren, S., & Roberts, S. (2019). Deep reinforcement learning for trading (arXiv:1911.10107). arXiv.
http://arxiv.org/abs/1911.10107