Survey of Reinforcement Learning Techniques for Financial Trading

Alexander Dennington, ad2n18@soton.ac.uk, 29534267

1 Abstract

In this review, a number of papers are analysed in the area of algorithmic trading with reinforcement learning and their techniques deconstructed according to the common elements that trading agents should look to incorporate. It is found that many of these papers have constructed high-performing models using adapted versions of common RL algorithms, and use similar reward functions and state/action spaces in their techniques. However, it is also found that not all papers include provisions for transaction costs and other market frictions which could limit the generalisation performance of the agent in a real world trading environment.
While it is a challenge to directly compare the model performances, it is found that the most promising models are based around the PPO algorithm, given its low resource demands and fast convergence without sacrificing model performance.

2 Introduction

Algorithmic trading is the application of algorithms in the field of financial products trading in order to optimize the allocation of capital and consequently maximise the performance of returns on investment.[15] Trading has represented a number of challenges to overcome, including its inherant stochasticity, poor observability of the trading environment, and when reducing the complexity of the trading environment and representing it in quantatative terms. Algorithmic trading has presented a solution to these problems, and has a number of advantages over other forms of trading, such as faster execution of trades and the diversification of risk. [9] Therefore, a lot of work has been focused on this area to produce better algorithms for trading, including exploring the potential utilisation of reinforcement learning agents to interpet the trading environment and make trading decisions (e.g. buy, hold, sell).
In algorithmic trading, there has been a heavy emphasis on feature extraction, which requires significant domain knowledge in order to hand-craft relevant features from the data. However, researchers have found that reinforcement learning algorithms can be applied to

the sequential real world problems of trading without requiring feature extraction or trade-strategy designs that previous methods relied on. Reinforcement learning (RL) methods can be catagorised into three distinct types: actor-only, critic-only and actor-critic methods. Actor-only methods such as Policy Gradient [6] find the optimal parameters for its policy network via interacting with the environment [6]. Critic-only methods such as TD and Q-Learning extract an optimal policy indirectly by performing value-iteration. Actor-critic methods combine the two approaches by having an actor to find the optimal policy and a critic to evaluate the performance of the actor. Actor-critic methods are known to provide more stable results than the two individual methods. [6]

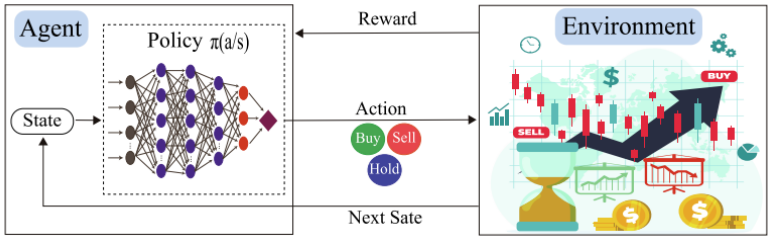

Figure 1: Diagram of RL in Financial Trading from [7]

Reinforcement learning is a highly active field within the realms of artificial intelligence due to its ease of applicability and flexibility. A huge number of research papers have been published in the last few years that utilise RL to solve problems from all sources of inspiration. Financial Trading is no exception to the rule since, due to its sequential nature, easily interpretable reward function and simple action-space, it seems highly suited to be exploited by reinforcement learning. However, there are still a few challenges that researchers are hoping to overcome. This review will highlight some of the latest literature on the topic of RL in Financial Trading, and how researchers have overcome some of these challenges while struggling to overcome others.

3 Survey

The research papers in this literature review have suggested a number of unique and novel RL algorithmic methods for training agents to perform trading decisions based on market information and historical data. These range from Deep Q-Networks to Actor-Critic Methods that have been adapted to the unique environment of financial trading. The following sections will seek to deconstruct the methods from these papers and evaluate their effectiveness for trading, and areas that could be improved.

3.1 Feature Extraction/Noise Reduction

One of the most difficult abstractions involved in interpreting financial markets is the task of extracting information from the numerous graphs and charts representing individual stock prices, market indexes and market performance/health indicators. This is mainly because of the sheer amount of noise present in such data, caused by speculative high frequency trading causing chaotic and uninformative fluctuations in financial time-series. However, behind the noisy movements there are beneficial trends that can be extracted using certain statistical methods and which can then be used by an RL agent to make more informed decisions. This task has been taken on-board by a lot of the latest methods in RL Trading. Li et al. created a method which utilises DQNs and A3C (Asynchronous Advantage Actor-Critic) in [1] which uses stacked-denoising autoencoders (SDAEs) to find a lower-dimensional latent space to represent the data, and in doing so, removes unnecessary random noise from the data. This is similar to Principal Component Analysis (and under certain conditions is identical) at removing the least informative dimensions from the data and projecting it into a lower dimensional space that captures most of the underlying structure, whilst minimising the reconstruction error. In the case of [1], the financial data for the chosen stocks are input to the SDAEs, which output a lower-dimensional latent space representation of the financial data (containing more concentrated information) and this in turn is fed to the learning agent.

Another method utilised by Wu et al. in [2] involves using Gated Recurrent Units (GRUs), a derivative of the LSTM (Long-Short Term Memory) which are designed to work with time-series and sequential data. These modules are a significant improvement on the simplicity of the Recurrent Neural Network, which suffers from exploding/vanishing gradients and struggles to remember long term information in large time-series. GRUs improve upon LSTMs by having two gates, an update gate and a reset gate inside the GRU module which are trained to decide what information is worth remembering, and what is irrelevant. Due to this definition, it shows how GRUs can be appropriately utilised to extract meaningful and relevant information from sequential financial time-series by learning what information is worth remembering (i.e. the more informative parts of the data) and what is just superflous noise (which it will learn to forget). These more meaningful extracted features will then be given to the learning agent (either the DQN or the A3C) to use.

The authors Liu et al. in [3] tackled extracting robust features from noisy financial data by modelling the whole process as a Partially Observable Markov Decision Process (POMDP). Modelling the Quantitative Trading Problem as a Markov Decision Process is considered a realistic generalisation, since the market is full of unexpected events and high-frequency trading behaviours which lead to a lot of noise, and make the actual market states unobservable. A POMDP is therefore a good fit for such an unobservable markov process. What sets this technique apart from the previous discussed in [1] & [2] is that the authors have assumed a POMDP as a good fit to the problem, whereas the previous techniques could be considered data-preprocessing, more specifically feature extraction before training

the agent on the extracted features. However, in order to solve the POMDP formulation, the authors Liu et al. proposed using an Iterative Recurrent Deterministic Policy Gradient (iRDPG) technique, which uses GRUs as the first layer of the model to extract meaningful features from the data, similar to what is performed in [2]. Therefore, although the formulation and framing of the feature extraction is different between [3] and [2], they actually perform a very similar operation on the data to produce more robust features for the agent.

A novel method used by Briola et al. in [8] for their high frequency trading agent to reduce the noise present in their state-space features involved selecting snapshots from the price data that had the biggest differences in mid-price level. This would mean that the model doesn't focus on the random noisier price changes but rather on the more significant and informative ones.
The authors Xiong et al. in [4], Yang et al. in [5] and in Zhang et al. in [6] did not implement a noise reduction/feature extraction technique into their methods, but did formulate the trading problem as a POMDP and then go on to train their models on market summary indicators such as Moving Average Convergence Divergence (MACD), Relative Strength Index (RSI), Commodity Channel Index (CCI) and Average Directional Index (ADI), all commonly used statistics to extract trends and meaningful information from time-series.

3.2 Market Volatility/Risk Averseness

An important element to making trading decisions is recognising the signs of a market downturn and reacting accordingly to minimise damage to the portfolio value. A volatile market often occurs when there is much uncertainty, caused by global events or policy changes. Volatility can result in/be caused by market crashes (leading to people rushing to sell shares). An RL agent which is prepared for a market downturn/crash is a safer, less risky agent than one which is not. Some of the methods in this review had techniques for dealing with market volatility and crashes. Being able to deal with market volatility/extreme market conditions can lead to a safer investment opportunity in a market prone to extreme crashes. For example, Yang et al. in [5] created a turbulance indicator which, when beyond a certain threshold, would automatically force the trading agent to halt buying and proceed to sell all shares. Trading would then resume once the turbulance indicator went back below the threshold. This turbulance can be calculated using a simple matrix operation:

$$\boldsymbol{turbulance}_t = (y_t - \mu)\Sigma^{-1}(y_t - \mu)^t$$

where y_t represents stock returns at time period t, μ is the average of historical stock returns, and Σ is the covariance matrix of historical stock returns. A high turbulance indicates extreme market conditions, while a low turbulance is indicative of a stable market. This is a useful fail-safe that future trading RL techniques should look to adopt if they want to reduce the risk of their investments, especially since, in relying on an A.I agent to

perform market trades, the operator may be less aware of present market conditions and slower to react to extreme rapidly occuring market events.
A similar technique is utilised in [7], which determines the volatility/riskiness of the market, and beyond a certrain level will limit the amount of buying the agent can do in the market until it determines the market is at a less volatile level.
Other methods, such as [6], included volatility in their reward function, and made the reward volatility-scaled using the Differential Sharpe Ratio. This allows the reward to be determined mostly by actions rather than depending on market volatility, giving a better, more truthful reward rather than the Trader profiting off of randomness.

3.3 State-space

Trading reinforcement learning algorithms need to observe the trading environment before selecting an action from the action-space based on that observation. The trading environment is usually represented in a simplified manner as a vector or matrix of values called the state-space. There is no singular way to reduce the trading environment into a matrix/vector form, or to decide which market indicators should be used. However, a lot of the papers reviewed used similar techniques to represent the market data in the state-space. For instance, the authors Li et al. in [1] used a state-space composed of market variables (e.g. daily open, close, high and low prices), technical indicators (MACD, MA, EMA, ATR, & ROC), and other useful informative variables such as cash remaining. Once an observation of these variables at a time-period t has been made, the observations are denoised in the SDAEs before being used for training by the agent. The use of technical indicators as well as the market information means the agent receives a good approximation of the trading environment.
The authors Yang et al. in [5] used a similar state-space to [1], using adjusted closing-price p_t, num. of shares of each stock h_t, cash remaining, and technical indicators such as MACD, RSI, CCI and ADX (as defined in section 3.1). The use of only the close price instead of OCHL (Open Close High Low) reflects the long-term goals of this method (and over a longer time-period, OHLC prices are more highly correlated). The methods in [2] use OCHL prices as well as most of the indicators mentioned, apart from the use of Bias (deviation of market share price from MA), Volatility Volume Ratio (VR) and On Balance Volume (OBV). These additional indicators are useful for measuring market volatility and allow the agent to determine more accurately when and how much to sell, given an unstable market. The authors Liu et al. in [3] used a state space which deviated from the rest in that the trading environment was split into price, technical indicators (Buyline and SellLine using the dual thrust strategy) and cumulative account profit. The dual thrust method is a reliable trend-tracking system that uses historical prices to determine the buying/selling prodedure.[10] To condense this information into an easily comparable format, refer to the table in Figure. 2 for an exhaustive checklist of state-space features for similar models [1]-[7]. (In the table, OCHL: Open-Close-High-Low, h: Number of stocks, Cash:

	O	C	H	L	h	Cash	CAP	MA	EMA	MACD	RSI	CCI	BIAS	VR	OBV	ATR	ROC	DT
[1]	✓	✓	✓	✓		✓		✓	✓	✓						✓	✓	
[2]	✓	✓	✓	✓		✓		✓	✓				✓	✓	✓			
[3]	✓	✓	✓	✓			✓											✓
[4]	✓	✓	✓	✓	✓													
[5]		✓			✓	✓				✓	✓	✓						
[6]		✓					✓			✓	✓							
[7]	✓	✓	✓	✓	✓	✓												

Figure 2: Checklist of state-space market & technical indicators for papers [1]-[7]

Total liquid cash available for buying shares, CAP: Cumulative Account Profit, MA: Moving Avg, EMA: Exp. Moving Avg, MACD: Moving Avg Convergence Divergence, RSI: Relative Strength Index, CCI: Commodity Channel Index, VR: Volatility-Volume Ratio, OBV: On-balance Volume, ATR: Average Directional Index, ROC: Rate of Change Indicator, DT: Dual Thrust).

The authors Briola et al. in [8] trained their agent on data in a way very different to the other articles. Since the main aim of their model is to perform high frequency trading, the state space was taken from the Limit Order Book (LOB), and used the values LOB Volumes, LOB States, Current Position (Long, Short, Neutral), Mark-to-market and Bid-Ask-Spread. These are market indicators more crucial to high-frequency trading and so are more dissimilar to the features used by other papers.

In summary, the state-space is always a simplification of the real trading environment, and depending on the main goal of the algorithm, certain features will be more useful than others. However, the most common market feature that's used in all the methods is closing price (or a normalised version of it), while the most commonly used technical indicator is the Moving Average Convergence Divergence (MACD) indicator.

3.4 Action-space

The action-space used by the trading agents in all the papers is where they are most similar. Since this is a trading 'game', the potential actions are limited to, at the most basic, "Sell", "Hold" and "Buy". Where they may differ is in how many shares the agent can trade at once. Some papers, like [1], [3], [4] and [5] have made their action space so that the agent can trade multiple shares at once, $|k| \geq 1$, i.e. $a = \{-k, ..., -1, 0, 1, ..., k\}$ or $a \in [-k, k]$. Others, such as [2], [6], [7] and [8] have restricted their agent's actions so that they can only trade single shares at any one time, i.e. $a = \{-1, 0, 1\}$ or $a \in [-1, 1]$. This may be a disadvantage when compared to the unrestricted agents since a real-life trader isn't subjected to the same restrictions on number of shares to trade, and being unrestricted will allow for faster gains to be made, and a higher potential return on investment. Also, being able to trade more shares at once can reduce the impact of transaction costs when compared to making the same value of trades individually. The paper [8] has an addtional action besides "Sell", "Hold" and "Buy" which is "Daily Stop Loss". This is where the agent checks the current cumulative profit for the day, R_{day}, and if $R_{day} < 0$, the current

position is closed and trading is stopped for the rest of the day to prevent further losses. This is a good fail-safe in the event of the algorithm causing massive losses, and means the agent will not have to be under constant monitoring for such behaviour. This action particularly suits high-frequency trading since the number of trades per unit of time is much larger than the other methods, and so losses could potentially occur much more quickly if the agent isn't being monitored.
Some papers, such as Yang et al. in [5], normalize the action-space from $a = \{-k, ..., -1, 0, 1, ...k\}$ to $a \in [-1, 1]$ or just convert their discrete action space $a = \{-1, 0, 1\}$ to a continous action space $a \in [-1, 1]$ such as in [6]. This is because the algorithms A2C and PPO define their policies on a Gaussian distribtution, which needs to be normalized and symmetric.
The authors Li et al. in [1] found that, with a larger capacity for trading ($|k| > 1$), the total return on investment increased non-linearly with respect to the number of shares allowed to trade. In their paper, they found that extending the action space from $\{-1, 0, 1\}$ to $\{-3, -2, -1, 0, 1, 2, 3\}$ actually increases the cumulative returns beyond 3 times the original. This may point towards higher possible returns with an increased action space, purely limited by the amount of cash available to spend on shares.

3.5 Market Frictions

Trading in a real world environment is not as simple as buying and selling at the moment of decision. There are a few constraints to the process which a good agent would incorporate into their training process. For example, the trader may face transaction costs, slippage, delays in executing orders and lack of a buyer when selling (it is usually the job of a broker to match together a buyer and seller; they're a third party who also receives commision for their part in the deal, a part of the transaction costs).
The papers in this review incorporated the transaction costs in different ways. For instance, the authors Zhang et al. in [6] included a heuristic that transaction costs when going from a fully long to fully short position are doubled, while no transaction costs are incurred if no action is taken (i.e hold). Another apporach by Yang et al. in [5] is to simply make transaction costs 0.1% of the total trade value, and subtract it from the available cash. While this may not always reflect the true costs incurred in a trade, over many trades it seems to be a good estimate. Some papers such as Li et al. in [1] and Liu et al. in [3] have the transaction costs and slippage as part of the training data, and the costs will be applied automatically as part of the reward function in the training phase according to the data. The authors Briola at al. in [8] use the concept of spread-crossing (buy at seller's price) which would incorporate the transaction costs. Other papers, such as Xiong at al. in [4], didn't explicitly specify how their method dealt with transaction and other trading costs, so it is assumed they were ignored or assumed negligible by the authors for their study. This would not help the agent to deal with real-world trading as many trading-agent policies that are very profitable under transaction-free trading become unprofitable and incur heavy losses when the costs are introduced.

Some of the other market frictions noted above may be considered negligible by the authors of these papers since they may be very complicated to model, and in some cases don't have a significant effect on proceedings and can be safely ignored. The main market friction of transaction costs has been addressed by some papers, while others have not specified clearly how their method deals with such measures.

3.6 Algorithms

There were a huge diversity of adapted algorithms in the papers for this review, all based around and derived from the following techniques: Deep Q-Networks (DQN), Policy Gradients (PG), Advantage Actor Critic (A2C), Asynchronous Advantage Actor Critic (A3C), Deep Deterministic Policy Gradients (DDPG), and Proximal Policy Optimisation (PPO). The most popular base algorithms were the Deep Q-Network and Deep Deterministic Policy Gradient, which were utilised in some way by four of the papers, closely followed by Proximal Policy Optimisation (PPO) which was utilised by three of the papers. The least popluar method of all the papers was Asynchronous Advantage Actor Critic (A3C), which was only used by one paper [1]. However, all the papers proceeded to alter the algorithms in such a way as to make improvements on previous methods and areas they consider to be weak. Note that some papers use more than one algorithm and compare performances between the different models they've created, or create an ensemble model as in [5].

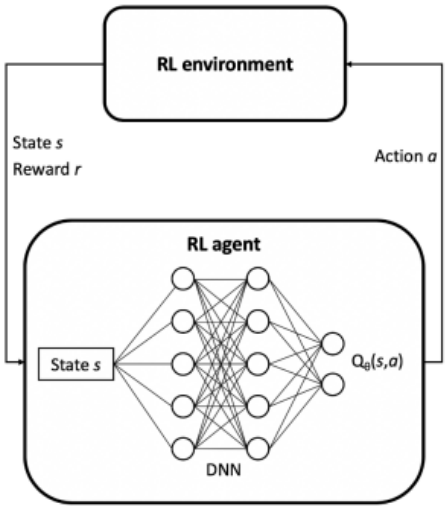

Figure 3: Diagram of DQN algorithm from [9]

Deep Q-Networks

The authors Zhang et al. in [6], Wu et al. in [2] and Li et al. in [1] used their own adapted versions of Deep Q-Networks as the basis for their trading agents (although Wu et al. created a DDPG agent as well). This technique involves the approximation of a State-action Value function $Q(s, a)$ which gives a rating to each action given a state s. The agent then proceeds to choose the action a which maximises this function, given a state s. The general process of the DQN algorithm is shown in Figure 3. The function $Q(s, a)$ is approximated using a neural network, enabling it to deal with large observation sets. The basic form of the DQN algorithm is as follows:

Algorithm 1: DQN Algorithm

1. Initialise the environment s_t;
2. Initialise the replay buffer R of capacity N, number of episodes m and number of iterations T;
3. Initialise the Q-Network with random weights θ, and the target network Q^{target} with weights $\theta' = \theta$;

for *episode in {1, ..., m}* **do**
  for *iteration in {1, ..., T}* **do**
    i. Fetch state s_t;
    ii. Select $a_t = \underset{a_t}{\text{argmax}}\{Q(s_t, a_t)\}$ using the state s_t;
    iii. Execute action a_t and receive reward r_t and then calculate new state s_{t+1};
    iv. Store the transition (s_t, a_t, r_t, s_{t+1}) in R;
    v. Sample a minibatch of size b randomly from R;
    vi. Set $y = r + \lambda Q^{target}(s_{t+1}, a_{t+1})$;
    vii. Find the derivatives of the loss function $L(\theta) = E[(y - Q(s, a))^2]$ and update the parameters θ using standard gradient descent.;
    viii Copy the Q network across to the target network every τ steps;
  end
end

As shown in the algorithm, a replay buffer R is used. This is a relatively new addition which has been shown to reduce correlations between experiences and therefore allows the DQN to generalise better. However, the authors Wu et al. in [2] have named their adapted DQN algorithm GDQN (Gated Deep Q-Network) since theirs utilises GRUs to extract meaningful features from the input financial data in order to produce a less noisy state-space for the DQN (rather than having the state-space be a direct vector of the prices and indicators).

Deep Deterministic Policy Gradients

The authors Xiong et al. in [4] and Wu et al. in [2] decided to utilise the DDPG algorithm[12], which is designed for large and continous action spaces, appropriate for their MDP modelling of the trading process. DDPGs are interpreted as an approximation to the State-Action Value function learned in the Q-Learning process. The Value-function approximated by Q-learning is ultimately attempting to output the action that maximises the value function:

$$a = \underset{a}{\text{argmax}}\{Q(s, a)\}$$

However, with a continous action-space, this becomes significantly more challenging for a computer to compute. In response to this, the DDPG algorithm was developed which approximates the action in the above equation using policy gradients. This is because the Policy Gradient algorithm learns a Policy function $\mu(s)$ directly. This function learns to output the most appropriate action given an input state s, i.e. $a = \mu(s)$. With this in mind, the DDPG is an approximation of the State-Action Value-Function:

$$Q(s, a) \approx Q(s, \mu(s))$$

In computing the optimal State-action value function, the DDPG algorithm utilises training and target networks, where the target networks are direct copies of the training networks, but they are copied at certrain time-intervals. While this seems unusual (since the training NN and target NN use the same parameters and therefore the target network is not fixed), it produces successful results, as demonstrated by Xiong at al. in [4], since their DDPG algorithm exhibited trading performance above the DJIA (Dow Jones Industrial Average) and the Min-Variance Portfolio Allocation strategy. This method was also utilised by Yang et al. in [5] as part of an ensemble model. The following is a step-by-step guide to the DDPG algorithm:

Algorithm 2: DDPG Algorithm

1. Initialise critic network $Q(s_t, a_t)$ and actor network $\mu(s_t)$ with weights θ^Q and θ^μ;
2. Initialise the replay buffer R of capacity N, number of episodes m and number of iterations T;
3. Initialise the target network Q' and μ' with weights $\theta^{Q'} = \theta^Q$ and $\theta^{\mu'} = \theta^\mu$;

for *episode in {1, ..., m}* **do**
  Choose random process for noise ϵ ;
  Fetch state s_t ;
  for *iteration t in {1,..., T}* **do**
    i. Select action $a_t = \mu(s_t) + \epsilon$;
    ii. Execute action a_t and receive reward r_t and then calculate new state s_{t+1} ;
    iii. Store the transition (s_t, a_t, r_t, s_{t+1}) in R ;
    iv. Sample a minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) randomly from R ;
    v. Set $y_i = r_i \lambda Q(s_{i+1}, \mu^{target}(s_{i+1}))$;
    vi. Update the critic network by minimising the loss $L = \frac{1}{N}\sum_{i=1}^{N}(y_i - Q(s_i, a_i))^2$;
    vii. Update the actor using $\Delta_{\theta^\mu} J(\theta) \approx \frac{1}{N}\sum_i (\Delta_a Q(s, a)|_{s=s_i, a=\mu_{s_i}} \Delta_{\theta^\mu} \mu(s)|_{s=s_i})$;
    viii. Update the target networks: $\theta^Q = \tau\theta_Q + (1-\tau)\theta^{Q'}$ and $\theta^{\mu'} = \tau\theta_\mu + (1-\tau)\theta^{\mu'}$;
  end
end

Proximal Policy Optimization

The next most popular algorithm to feature in the selected papers is the Proximal Policy Optimisation algorithm (PPO). This is a state-of-the-art method that is often the first port-of-call for those looking to create an RL agent to perform a certain task. This is because of its ease to implement and the fact it can handle both continous and discrete action-spaces. It works by optimising its policy network using a trust-region method rather than a line-search method (e.g. gradient descent). Line-search methods work by finding the steepest direction of the loss function and moving up/down that direction by a certain step-size. The search for the step-size is the crucial part of this method since too large a step will result in missing the optimum and ending up in a sub-optimal region, while too small a step will take too long to converge. Rather than utilising dynamic step-size formation methods like momentum, another way is to use trust-regions. This requires defining a trusted region around the current point in the loss-space, and finding the optimal point within that region, moving there, and repeating. The PPO method seeks to restrain the

trust region using either the KL-divergence or through clipping. It works as follows:

Algorithm 3: PPO Algorithm

1. Initialise policy parameters θ^0 ;
for *i = 1,..., T* **do**
  i. Use θ^k to interact with the trading environment and collect $\{s_t, a_t\}$ and compute the advantage $A^{\theta^k}(s_t, a_t)$ using the chosen method of advantage estimation;
  ii. Find θ_{k+1} optimizing $J_{PPO}(\theta) = E_{(s_t,a_t)\sim\pi_{\theta'}}[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}A^{\theta'}(s_t, a_t)] - \beta KL(\theta, \theta')$
  iii. Let $\theta_k = \theta_{k+1}$;
end

PPO methods are known to be more stable than other policy gradient methods and usually provide an equal if not better performance than previous state-of-the-art algorithms using less time and resources. Therefore this method was chosen by the more complex and advanced papers in this review that were published more recently, although the methods that utilised PPO here used clipping in their version of the algorithm. This seeks to restrain the policy update by choosing the minimum of the clipped and normal policy update values. The PPO has been chosen as a method by these papers because of its high level of performance while being efficient with computing resources, while also providing a restrained optimisation approach when compared to other RL methods. When the agent is interacting with the trading environment to collect trading decision observations, if the policy has a poor set of parameters, it may never recover. Therefore a more constained approach to optimization such as PPO offers will prevent the policy from updating to a poor set of parameters far from the previous policy parameters.

Advantage Actor-Critic

Lastly, the authors Yang et al. in [5] and Zhang et al. in [6] used the A2C algorithm as part of their trading agent models. This algorithm utilises both actor and critic functions by using the policy from the actor network to represent the baseline in the Bellman update equation, and the Value function from the critic Q-Network to approximate the Expected Future Discounted Reward. The advantage is then calculated by $Q^\pi(s_t^n, a_t^n) - V^\pi(s_t^n)$ where the Q-function is approximated by the actor network function:

$$Q^\pi(s_t^n, a_t^n) = E[r_t^n + \gamma V^\pi(s_{t+1}^n)] \approx r_t^n + \gamma V^\pi(s_{t+1}^n)$$

This approach is useful since, different from other policy gradient methods, it approximates the advantage function (a reward-standardised policy network function). Since the rewards have been standardised, information can be obtained not only on the value function but also on the performance relative to other instances. This gives information on how much better the policy could be, i.e the advantage, and helps to reduce the variance of the policy

network and make it more robust[5]. A3C (Asynchronous Advantage Actor Critic) is an extension of A2C where there are multiple copies of the same agent working simultaneously to update the gradients of the policy network using different data samples. This works by calculating the average gradient across all agents and passing that update to the global policy network. Then the multiple agents receive the same gradient update and repeat the procedure. The Actor Critic frameworks (A2C and A3C) are suited to stock trading due to the stability they provide in policy updates. However, the PPO algorithm mentioned previously is seen to perform better than the A2C and A3C algorithms as it is more stable and uses fewer resources, potentially making the actor-critic model redundant.

3.7 Reward Functions

One of the most crucial aspects to any reinforcement learning task is having a suitable reward function. For the trading problem, the reward function of gains from investment seems very intuitive, and for some papers this has been used. However, for the more advanced papers in this review, a more intricate reward function was chosen which utilised the gain from investment. This is because, according to [11], account profit isn't an effective reward function for quantitative trading problems.
For the basic reward function of account profit, utilised by [1], [4], [5] and [8], it was calculated by using an equation similar to:

$$r_t = p_t - p_{t-1} - c_t$$

where r_t is the reward at time t, p_t is the portfolio value at time t, p_{t-1} is the portfolio value at time $t-1$ and c_t are the transaction costs at time step t. For the methods which chose transaction costs as a fixed percentage of the total value of the trade, the transaction cost $c_t = 0.1(p_t - p_{t-1})$.

For the methods which chose a more intricate reward function, the authors Liu et al. in [3] and Zhang et al. in [6] decided to go with the 'Differential Sharpe Ratio' (D) as defined in [11] which is:

$$d_t = \frac{dSr_t}{d\eta} = \frac{\beta_{t-1}\Delta\alpha_t - \frac{1}{2}\alpha_{t-1}\Delta\beta_t}{(\beta_{t-1} - \alpha_{t-1}^2)^{\frac{3}{2}}}$$

where

$$\alpha_t = \alpha_{t-1} + \eta\Delta\alpha_t = \alpha_{t-1} + \eta(r_t - \alpha_{t-1})$$

and

$$\beta_t = \beta_{t-1} + \eta\Delta\beta_{t-1} = \beta_{t-1} + \eta(r_t^2 - \beta_{t-1})$$

The definition of the Sharpe Ratio is the ratio of excess return over a unit of total risk, i.e

$$Sr_t = \frac{\mathbb{E}[R_{t-n:t}]}{\sigma[R_{t-n:t}]}$$

The authors Wu et al. in [2] used a slight variation of the Sharpe Ratio (Sr) as their reward function, called the Sortino Ratio (SR). This is calculated as such:

$$SR = \frac{\mathbb{E}[R] - T}{D[R]}$$

The main difference between using the Sharpe Ratio and the Sortino Ratio is that the former focuses on total volatility and is better for low volatile investment whereas the latter focuses on negative volatility and is better for evaluating risk-adjusted performance.
As stated earlier, in [11] the authors put forth an argument that the profit from a trade (minus costs) was not a good reward function for a trading agent, and that better reward functions which are volatility-scaled allow the agent to learn a more effective strategy and be able to make better trading decisions while considering market volatility, an important element to financial trading.

3.8 Evaluation Procedures

In order to determine whether their methods were improvements upon standard techniques, the papers included baseline models to make comparisons against, using certrain metrics that summarised the performance of the models. In [4] and [5], the authors used the Dow Jones Industrial Average (DJIA) and Minimum Variance Portfolio Allocation Strategy as baselines, and used the Sharpe ratio and Annualised Return Value as metrics with which to compare them. The Sharpe ratio (SR) is a reliable metric which is a risk adjusted performance summary. However, the limited number of metrics and baselines in these papers means their results are less reliable.
The paper [1] also used the Sharpe ratio and Annualised Return, as well as % accuracy. However, it only used one baseline, the Long and Hold (L&H) strategy, in which the Sharpe ratio can't be calculated for. The lack of numerous baselines as well as the inability to calculate a key metric for the baseline is disappointing for this paper.
The authors of paper [2] used the trading strategy 'Turtle' as a baseline, as well as using the Sortino Ratio and Rate of Return as metrics. The use of the Turtle trading strategy is a solid baseline since it has been a highly succesful trading strategy, however they could have used more baselines in their paper to make their results more trustworthy.
In the paper [3], the authors chose to use Long & Hold and Short & Hold as baselines, while using Total Return Rate (TRR), Sharpe Ratio, Volatility and Max Drawdown as metrics. The fact the authors are using a large number of metrics is good for comparing different elements of the model, however the number and strength of the baselines could be improved upon since the Long/Short & Hold techniques are not actual trading strategies but benchmark indicators, where the trader takes a long/short position at the start of the trading period and maintains it until the end of that period.
The authors of [6] used the Long & Hold, Sign(R), $\phi MACD$ and $\widetilde{MACD}$ baselines. The

baselines involving the MACD are more solid comparisons since they are proven time-series momentum strategies. They use a high number of metrics, including: $E(R), \sigma(R)$, Downside Deviation (DD), Sharpe Ratio, Sortino Ratio, Maximum Drawdown, Calmar Ratio, % Positive Returns and Positive-Negative Returns Ratio. The high number and quality of metrics and baselines in this paper means their results have been thoroughly tested and compared, and therefore any conclusions drawn about the methods in [6] are more reliable than the previous papers.
The authors in [7] had a good evaluation technique where they compared their proposed algorithm to a Deep Q-Learning technique which has been shown to also work and produce profits. The comparisons were very graphical and demonstrated their model's superior performance to a standard DQN agent, rather than choosing a simple unintelligent baseline as most of the other methods did.
In summary, the evaluation measures in general could include more metrics, and improve upon their baselines by comparing against a state-of-the-art algorithm to showcase that their method truly is an improvement, rather than tactically picking metrics and baselines which inflate their apparent performance.

3.9 Datasets

A machine learning technique is only as good as the quality and appropriateness of the data that it learns from. A model will generalise better if the dataset it learns from covers more diverse observations and is larger in size. In the case of financial trading, since the strategy is quite similar across different assets (e.g. stocks, futures, indexes,...), training an agent on multiple types of assets is a good way to create a general and versatile trading agent which can operate in different trading environments instead of a rigid agent which is only trained to deal with specific assets. Also, since including different assets is another way to increase the amount of data and trading scenarios that the agent might face, this all benefits the trading agent.
The authors Xiong et al. in [4] used a dataset of 30 diverse stocks from the Dow Jones marketplace, dated from 01/01/09 to 30/09/18, while the authors Yang et al. in [5] used the same but over a longer time period (up to 08/05/20). The authors Zhang et al. in [6] used 50 ratio-adjusted continuous futures from the CLC Database from the period 2005-2009 and made sure to select a variety of asset classes, a good way to generalise to different asset types. These listed papers have put an emphasis on using a large number of stocks from single markets, which means that, while the agent will be well trained on a diverse number of stocks, they may not generalise well in practice if they were applied to a different stock market than the one they were trained on.
The authors Liu et al. in [3] selected IF and IC stocks and futures from the Chinese stock market. This seems to be fairly restrictive to the trading agent in terms of generalisation to other markets, however the simultaneous inclusion of stocks and futures means that it has more asset diversity than just a single type of asset.

The authors Wu et al. in [2] chose 3 US Stocks, 3 UK Stocks and 3 Chinese stocks, trying to choose a mixture that were trending or volatile, while Hirchoua et al. in [7] chose 8 stocks from a variety of asset classes (Indexes, Commodities, Cryptocurrency and Big Tech stocks) from different markets. Both these papers have balanced market diversity with another element, the former choosing to focus on building the agent to learn how to approach stock volatility while the latter chose to focus on generalising to different asset classes. Also, [7] was the only paper in this study to include a cryptocurrency training asset (Bitcoin in this instance) which is an area of heavy commerical interest due to the relative newness of the market and the volatility of the currencies.
The authors Li et al. in [1] selected 3 stocks and 2 futures from the Thomas Reuters Historical database over a 10 year time-period which they split up for training and testing. The lengthy time period allows the agent to cover a lot of historical financial events and scenarios and learn the best course of action, however, the limited number of different stocks and futures will be a disadvantage compared to the other methods, since the agent will not have as good a generalisation capability when faced with different assets or stocks than the ones trained on.
The papers in this study all included more than one type of stock and, when the historical pricing information is combined with other market indicators from the datasets, the agents have enough data to be able to produce effective trading agents with some generalisation capability.

4 Analysis

Given the large number of proposed models in this review, and the variety of techniques that they've utilised to train an RL agent, it is difficult to determine which model is the better of them all. The lack of a direct comparison between them, along with a standard dataset with which to train on, makes any statement about ranking them untrustworthy and purely conjecture. However, the trend of utilising Proximal Policy Optimisation techniques in the more recent papers in the review makes the case that this is the stronger of the algorithms. The logical pathway is that, given the prepublished and already defined state-of-the-art techniques, the latter papers [5], [7], [8] deciding to use PPOs as a way to improve upon the state-of-the-art is a clear sign that these methods are the way forward in Financial Trading with RL. Additionally, the PPO method in [7] was tested against another method not covered in this review, titled "Financial trading as a game: A deep reinforcement learning approach" by Huang, C. Y [13]. Since this approach was published in 2018, it is still fairly recent and therefore should provide a significant indicator of performance for the PPO method in [7]. The results showed that, while Huang's technique produced profits of 257.19, the proposed system in [7] produced profits of 1588.22, a much larger return, in spite of the larger number of transactions (601 compared to 331). The technique in [7] is unique among the methods in that it specifically encourages the agent to take risks

and explore different areas of its domain in order to better handle and quantify the risk, leading to better decision making. According to the authors, the agent attempts to learn strong actions in risky and uncertain states.[7] However, the downside to this method is that it is not optimised for portfolio trading, and therefore is limited by what it can do. Research has shown that trading with a portfolio significantly reduces risk [14] and so a potential improvement on this state-of-the-art trading algorithm is to enter the domain of portfolio management. In fact, multiple papers came to the same conclusion that future research pathways should put a heavy emphasis on portfolio management. Also, some methods didn't include a clear indication of incorporating transaction costs, or the method of incorporating them was approximate and therefore not as reliable as methods which explicitely included true transaction costs that came as part of the training dataset. Also, the numerous metrics and baselines made it difficult to directly compare performances of the algorithms. The researchers in this area could collaborate more on a communication framework which can benefit both the researchers and observers, since the researchers will better be able to determine their own model performance, and make corresponding adjustments, while observers will better be able to determine which techniques are best for utilisation as a trading agent.

5 Conclusion

In this review, a number of papers have been analysed and their techniques deconstructed according to the common elements that trading agents incorporate. It was found that many of these papers had constructed highly performing models using adapted versions of common RL algorithms, and used similar reward functions and state/action spaces in their techniques. However, it was also found that not all papers included provisions for transaction costs and other market frictions which could limit the generalisation performance of the agent in a real world trading environment.
While it was challenging to directly compare model performance, it was found that the most promising models were based around the PPO algorithm, given its low resource demands and fast convergence without sacrificing model performance. In future works, papers in this area should focus their attention on Portfolio Management since this would allow the potential for greater returns at a lower risk.

References

[1] Yang, Li., Zheng, W., & Zheng, Z. "Deep Robust Reinforcement Learning for Practical Algorithmic Trading" *IEEE Access, vol. 7, pp. 108014-108022, 2019* doi: 10.1109/ACCESS.2019.2932789

[2] Wu, X., Chen, H., Wang, J., Troiano, L., Loia, V., Fujita, H. "Adaptive stock trading strategies with deep reinforce-

ment learning methods", *Information Sciences, Volume 538, 2020, Pages 142-158, ISSN 0020-0255* https://doi.org/10.1016/j.ins.2020.05.066. (https://www.sciencedirect.com/science/article/pii/S0020025520304692)

[3] Liu, Y., Liu, Q., Zhao, H., Pan, Z., & Liu, C. "Adaptive Quantitative Trading: An Imitative Deep Reinforcement Learning Approach". *Proceedings of the AAAI Conference on Artificial Intelligence, 34(02), 2128-2135.* https://doi.org/10.1609/aaai.v34i02.5587

[4] Xiong, Zhuoran and Liu, Xiao-Yang and Zhong, Shan and Yang, Hongyang and Walid, Anwar "Practical Deep Reinforcement Learning Approach for Stock Trading". *CoRR, abs/1811.07522, 2018* https://arxiv.org/abs/1811.07522

[5] Yang, H., Liu, X., Zhong, S., & Walid, A., "Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy". https://ssrn.com/abstract=3690996

[6] Zhang, Z., Zohren, S., & Roberts, S. "Deep Reinforcement Learning for Trading" *The Journal of Financial Data Science Apr 2020, 2 (2) 25-40* DOI: 10.3905/jfds.2020.1.030

[7] Hirchoua, B., Ouhbi, B., Frikh B., "Deep reinforcement learning based trading agents: Risk curiosity driven learning for financial rules-based policy" *Expert Systems with Applications, Volume 170, 2021, 114553, ISSN 0957-4174,* https://doi.org/10.1016/j.eswa.2020.114553.

[8] Briola, A., Turiel, J., Marcaccioli, R., Aste, T., "Deep Reinforcement Learning for Active High Frequency Trading" arXiv 2021

[9] Thibaut, T., Ernst, D. *"An application of deep reinforcement learning to algorithmic trading"* Expert Systems with Applications Vol. 173 (2021)

[10] GAEA Trading *Quantitative Strategy Research Series One — The Dual Thrust* https://medium.com/@gaea.enquiries/quantitative-strategy-research-series-one-the-dual-thrust-38380b38c2fa

[11] Moody, J., Saffell, M. *(2001). Learning to trade via direct reinforcement.* IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council. 12. 875-89. 10.1109/72.935097.

[12] Lillicrap, Timothy P. and Hunt, Jonathan J. and Pritzel, Alexander and Heess, Nicolas and Erez, Tom and Tassa, Yuval and Silver, David and Wierstra, Daan *"Continuous control with deep reinforcement learning"* arXiv 2015 doi: 10.48550/ARXIV.1509.02971

[13] Huang, C. Y. (2018) Financial trading as a game: A deep reinforcement learning approach arXiv preprint arXiv:180702787; arXiv:1807.02787.

[14] Nick Lioudis The Importance of Diversification https://www.investopedia.com/investing/importance-diversification

[15] James Chen, Gordon Scott, Hans Daniel Jasperson Algorithmic Trading https://www.investopedia.com/terms/a/algorithmictrading.asp#: :text=Algorithmic%20trading%20is%20a%20process,to%20the%20market%20over%20time