

Deep Reinforcing Learning for Trading with $TD(\lambda)$

ZiHao Wang and Qiang Gao

Department of Electronic and Information Engineering, Beihang University, Beijing 100191, China

Keywords: Trading Decision, Reinforcement Learning, Partially Observable Markov Decision Process, $TD(\lambda)$, Online.

Abstract: In recent years, reinforcement learning algorithms for financial asset trading have been extensively studied. Since in financial markets the state observed by the agent is not exactly equal to the state of the market, which could affect the performance of reinforcement learning strategies, the trading decision problem is a Partially Observable Markov Decision Process(POMDP). However, few studies have considered the impact of the degree of Markov property of financial markets on the performance of reinforcement learning strategies. In this paper, we analyze the efficiency and effectiveness of Monte Carlo(MC) and temporal difference(TD) methods, followed by analyze how $TD(\lambda)$ combines the two methods to reduce the performance loss caused by partially observable Markov property with the bootstrap parameter λ and truncated horizon h . Then considering the non-stationary nature of the financial time market, we design a stepwise approach to update the trading model and Update the model online during the transaction. Finally, we test the model on IF300(index futures of China stock market) data and the results show that $TD(\lambda)$ performs better in terms of return and sharpe ratio than TD and MC methods and online updates can be better adapted to changes in the market, thus increasing profit and reducing the maximum drawdown.

1 INTRODUCTION

Over the past few decades, financial trading has been a widely studied topic, and various trading methods have been proposed to trade in financial markets such as fundamental analysis (Singhvi, 1988; Abarbanell and Bushee, 1998), technical analysis (Lim et al., 2019; Murphy, 1999) and algorithmic trading (Schwager, 2017; Chan, 2013) and so on. In recent years, reinforcement learning has been widely applied in many areas and has achieved remarkable success in several application fields, including autonomous driving (Cultrera et al., 2020; Shalev-Shwartz et al., 2016), games (Littman, 1994; Vinyals et al., 2019), resource scheduling (Mao et al., 2016). At the same time, the technology is increasingly being used in financial transactions to generate higher returns (Deng et al., 2016; Carapuço et al., 2018; Huang, 2018; Lei et al., 2020; Moody et al., 1998).

Neuneier et al. (Mihatsch and Neuneier, 2002) combines the Q-learning algorithm in reinforcement learning with neural networks to make trading decisions. In the process, they modeled the trading process as a Markov decision process (MDP), with instantaneous profit as the reward function. Inspired by DQN in solving game problems, Chien Yi Huang et

al. (Huang, 2018) modeled the trading decision problem as an MDP process and adopted DQN for the model to make trading decisions. Stefan Zohren et al. (Zhang et al., 2020) also modeled the trading decision problem to MDP and used three reinforcement learning algorithms to test in a variety of financial markets, and all achieved higher performance than traditional technical indicator strategies.

The above study defaults that the trading decision process is an MDP, i.e., the state of the financial market environment is observed by the agent is be Markovian. And the TD method that exploits the Markov property of the environment is used for training. However, there are many factors that affect the price changes of financial assets, including economic development level, money supply, interest rate, inflation level, balance of payments, population structure, etc. from a macro perspective. Specific to the price impact of a single financial asset, it includes the company's industrial strength, financial status, bullish and bearish behavior of participating traders, and even financial news, market sentiment and other factors. Obviously, according to the public data available in the market, the above effective information affecting the price of financial assets cannot be fully covered. And the state would be informationally perfect only if it re-

tained all information about the history and thus could be used to predict futures as accurately as could be done from the full history. In this case, the state S_t is said to have the full Markov property. But in financial market environment, the states of agents can not be Markov but may approach it as an ideal. Therefore, financial asset trading is not a standard Markov decision process, but a typical partially observable Markov decision process.

Furthermore, the TD method exploits Markov property of the environment (Sutton and Barto, 2018), so the effectiveness of the TD method is poor in the financial market environment that does not fully meet the Markov property. Although the MC method does not exploit the Markov property, it can only be used for non-continuous tasks and can only be updated until the end of the curtain, so the efficiency is low and requires huge training data for training. Therefore, for financial trading problems with high training speed requirements and tight data, MC method is not applicable.

Recently, some scholars have considered the partially observable Markov property of financial markets. Chenlin et al. (Chen and Gao, 2019) modeled the transaction decision problem to POMDP and used LSTM to make the POMDP close to the MDP and improve the impact of partial observability. Although some scholars want to improve the performance impact of some observable financial markets, no studies have optimized the RL training method to reduce the performance degradation caused by partial observability.

In this paper, to address the above issues, we describe the trading decisions problem as a POMDP and acknowledge that the agent cannot fully observe the information of the financial market. At the same time the $TD(\lambda)$ method, which combines TD and MC, is used for training to improve the performance degradation brought by partial observables and to investigate the effects of the bootstrap parameters λ (which could trade off TD and MC) and the truncated horizon h (which could regulating the time period of bootstrap) on performance. At the same time, considering the non-stationary nature of the financial time market, in order to be able to adapt the strategy to the changes of market conditions as much as possible, we use a stepwise approach to update the strategy and make the strategy update online in the test set. Experimental results of our idea on IF300 data show that the $TD(\lambda)$ method outperforms the TD and MC methods, and that online updating avoids generating large maximum retractions.

2 METHODOLOGY

In this section, we model the trading decisions as POMDP and introducing our settings of RL including state spaces, action spaces and reward functions, and describe the RL algorithm used in this paper, PPO. Then we analyze the characteristics of RL training methods, TD, MC and $TD(\lambda)$ method. Finally, we study how does the bootstrapping parameters λ with truncated horizon h affect the efficiency and effectiveness of the $TD(\lambda)$ method.

2.1 POMDP Formalisation

In financial markets, security prices are influenced by many factors, such as macroeconomic policies and microeconomic activities, which contain information about unpredictable events and the trading behavior of all market participants. Therefore, it is difficult for an agent to observe the complete information of the financial market and the trading process can be considered as a POMDP.

POMDP is a generalization of Markov decision process. The POMDP is proposed mainly because in realistic environments, the state observed by the intelligence is not equal to the full state of the current environment, which can affect the learning efficient and effective of reinforcement learning strategies.

Therefore, we model the transaction decision problem as a POMDP in which the agent interacts with the environment at discrete time steps. At each time step t , the agent obtains the observed value of the environment as the agent state S_t . The more historical information this state can represent, the higher the degree of Markovian properties it contains. The agent selects an action A_t according to this state, and receives a reward R_t returned by the environment at the next time step, while the agent enters the next state S_{t+1} . A trajectory $\tau = [S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots]$ is generated during the interaction of the agent with the environment. At time step t , the objective of RL is to maximize the expected return, denoted as G_t at time t .

$$G_t = \sum_{k=t+1}^T \gamma^{k-t-1} R_k \quad (1)$$

State Space. The characteristics of financial markets are always contained in the historical prices of assets, and technical indicators are extracted from the characteristics of prices, so this paper not only uses spreads as the characteristics of the state space but also uses technical indicators as part of the state space. A list of our features is below:

- Normalised spread series,

- Action at $t - 1$ moment A_{t-1}
- MACD indicators shows the relationship between exponential moving averages of security prices on different time scales.

$$MACD_t = DEA_t - DIF_t \quad (2)$$

$$DIF_t = m(p_{t-12:t}) - m(p_{t-26:t}) \quad (3)$$

$$DEA_t = m(DIF_{t-9:t}) \quad (4)$$

$m(x_{t-n:t})$ is an exponentially weighted moving average of x with time scale n , p_t is close price at time t .

- RSI indicators is overbought (above 80) or oversold (below 20) by measuring the recent rise or fall in contract prices

$$RSI = \frac{AUL}{AUL + ADL} * 100\% \quad (5)$$

AUL is sum of the returns from the last n time steps, ADL is sum of the losses of the last n time steps. We set the periods n is 20 days. days in our state representations.

Action Space. A discrete action space is used, a simple action set $\{-1, 0, +1\}$, where each value directly represents a position, i.e. -1 corresponds to a short position, 0 to no position and $+1$ to a long position.

Reward Function. The reward function depends on profits R_t which defined as

$$R_t = a_t(p_t - p_{t-1}) - bp \cdot p_t(a_t - a_{t-1}) \quad (6)$$

Transaction costs are necessary because transaction fees and slippage are inevitable in real transactions, and defining transaction costs in the reward function is a major advantage of reinforcement learning. bp (basis point) is a parameter used to calculate transaction fees and slippage.

2.2 RL Algorithm

PPO is proposed to solve the problem of performing one asymptotic update for each data sample obtained from PG interaction with the environment, and optimizing a “surrogate” objective function using stochastic gradient ascent. PPO consists of two networks: an actor network that outputs the policy and a critic network that uses the action value function to measure the goodness of the chosen action in a given state. We can update the policy network by maximizing the “surrogate” objective function.

$$L^{CLIP}(\theta) = E_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)] \quad (7)$$

where \hat{A}_t is the advantage function defined as:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \dots + (\gamma\lambda)^{(T-t+1)}\delta_{T-1} \quad (8)$$

$$\delta_t = R_t + \gamma V(S_{t+1}|\omega) - V(S_t|\omega) \quad (9)$$

Whereas standard policy gradient methods perform one gradient update per data sample(Zoph et al., 2018), PPO update more times by add $r_t(\theta)$ which denote the probability ratio to the formula

$$r_t(\theta) = \frac{\pi_\theta(a_t|S_t)}{\pi_{\theta_{old}}(a_t|S_t)} \quad (10)$$

To compute the advantage functions, we use the critic network and train the network using gradient descent minimizing TD-error

$$L(\omega) = E_t[(G_t - V(S_t|\omega))^2] \quad (11)$$

There are many methods to calculate the cumulative discount return G_t in the current state, such as MC, TD and $TD(\lambda)$. Different application scenarios should choose different calculation methods, because the efficiency and effectiveness of the models differ between different calculation methods.

2.3 RL Training Methods

2.3.1 Monte Carlo

The Monte Carlo method requires sampling sequences from the interaction of the agent with the environment and solves the reinforcement learning problem based on the average sample return, which is defined as follows.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-1} R_T \quad (12)$$

The Monte Carlo method must wait until the end of the episode to be updated, because only then is the return known. But some applications have long episodes, and postponing all learning until the end of the episode is too slow. The financial trading decision problem is one such task with a long sequence, so the efficiency of MC is lower. However, they may be less harmed by violations of Markov properties. This is because they do not update their value estimates based on the value estimates of subsequent states. In other words, this is because they do not perform bootstrap.

2.3.2 Temporal Difference

Temporal Difference methods update their estimates based to some extent on other estimates. They learn a guess from a guess, which is called bootstrap method. The formula is as follows

$$G_t = R_{t+1} + \gamma V_t(S_{t+1}) \quad (13)$$

TD exploits Markov property, their aim is to find fully correct estimates of the maximum likelihood model for Markov processes. Therefore, TD is less effective for non-Markov environments. However the TD method only needs to wait one time step to update, and it learns from each transition without regard to what action is subsequently taken, whereas the MC method must wait until the end of an episode to update. Therefore, the efficiency of TD method is better than MC method.

2.3.3 $TD(\lambda)$

$TD(\lambda)$ methods unify and generalize TD and MC methods. $TD(\lambda)$ methods update with λ -return which utilizes multi-step jackpot with bootstrapping. The formula is as follows:

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_T \quad (14)$$

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}) \quad (15)$$

The $TD(\lambda)$ method approximates the MC method when λ equals 1, and it approximates the TD method when λ equals 0. MC converges slowly but has better effectiveness with partially observable Markov decision process, while TD converges quickly but has poor effectiveness with partially observable Markov decision process. By adjusting bootstrapping parameter λ it is possible to balance TD and MC and reduce the performance degradation caused by partially observable Markov.

However, λ -returns depend on n -step returns for arbitrarily large n , and thus can be computed only at the end of the episode, as in the MC method. But, the longer the reward time, the weaker this dependence becomes, since each delayed step drops $\lambda^n \gamma$. Then, approximations can be made by truncating the sequence after some time steps and replacing subsequent missing rewards with estimates. Therefore the truncated horizon h the greater the proportion of estimated values.

In general, we define the truncated λ -return for time t , given data only up to some later horizon, h , as

$$G_{t:h}^\lambda = (1 - \lambda) \sum_{n=1}^{h-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{h-t-1} G_t \quad (16)$$

$$0 \leq t < h \leq T.$$

2.3.4 Performance Analysis of $TD(\lambda)$

To illustrate the idea of $TD(\lambda)$ methods could reduce the impact of performance degradation brought by

partially observable Markov property, we choose different bootstrapping parameter λ and truncated horizon h for the experiment. For these illustrative examples, we trained the model 2000 times on three years of historical data, and we consider the average cumulative returns of the first 100 epochs as the evaluation index of efficiency, and the average cumulative returns of the last 100 epochs as the evaluation index of effectiveness. The experimental results are shown in Figure 1.

As can be seen in 1(a), when considering the efficiency of the model, the more available value of λ is on the left side because at this time the $TD(\lambda)$ method tends to TD method, while when considering the effectiveness of the model, as can be seen in 1(b), the more available value of λ is on the right side, because at this time the $TD(\lambda)$ method tends to MC method, but when λ equals to 1, the efficiency and effectiveness drops sharply because $TD(\lambda)$ methods produce a pure Monte Carlo, which converge slower and training iterations 2000 times still for convergence to a better position.

When the truncated horizon h is small, the time period of bootstrap is shorter, the weight of the estimate is larger, and the effectiveness is poorer. As the field of truncated horizon increases, the time period for performing the bootstrap increases. There is a significant change in state during this time period and the loss generated by this change can propagate and affect multiple state updates, so the efficiency is higher.

3 EXPERIMENTS AND RESULTS

We design experiments to test our ideas. In this section, we present the experimental setup. Then the performance of the $TD(\lambda)$ method, MC and TD methods are compared.

3.1 Description of Dataset

We use data on IF300, our dataset ranges from 2002 to 2022, which are shown in Figure 2. We use 11 years of data for training to get the model. We then retrain our model every 3 years, using 3 years of data to update the parameters. And for the next 3 years, make the model parameters are fixed or updated online as a way to perform trading simulations. So in total, our test set is from 2013 to 2022.

3.2 Training Schemes for RL

In our work, we use the $TD(\lambda)$ method and consider effectiveness and efficiency separately of the re-

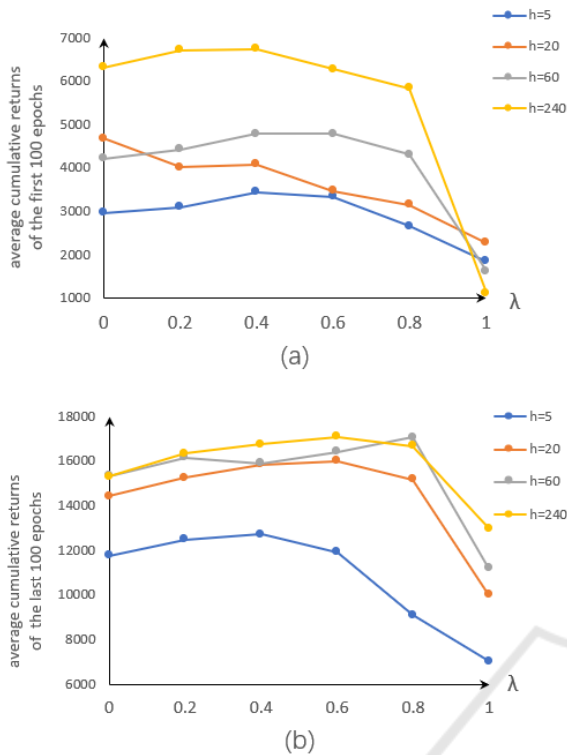


Figure 1: The efficiency and effectiveness of $TD(\lambda)$ varies with bootstrapping parameter λ and truncated horizon h :(a) efficiency. (b)effectiveness.



Figure 2: Daily closing price of IF300.

inforcement learning strategy and choose different parameters.

On the train data set, in order to be able to train a model with the best possible initialization, we chose the parameters for 6000 training iterations when the effectiveness of the $TD(\lambda)$ method was good.

On the test data set, we consider the effectiveness of the RL methods and retrain our model offline 2000 times at every 3 years. Model parameters are then fixed for the next 3 years.

Furthermore, considering the non-stationary nature of the financial time market, in order to be able to adapt the strategy to the changes of market conditions as much as possible. Model parameters are then updated online for the next 3 years, and in order to avoid the model training time is long and thus miss timing

of financial asset trading, which leads to a large loss, the parameters of $TD(\lambda)$ should be chosen when efficiency is better, and the model is updated in the test set by training 100 epochs in this paper.

Table 1: Values of hyperparameters for different different methods.

Hyperparameter	Value
learning rate α	0.001
discount factor γ	0.99
bootstrapping parameters λ with efficiency	0.2
truncated horizon h with efficiency	240
bootstrapping parameters λ with effectiveness	0.6
truncated horizon h with effectiveness	240
parameter accounting for transaction fees and slippage bp	1

3.3 Experimental Results

We test both Buy and Hold and our methods between 2013 and 2022, and We evaluate the performance of this trading strategy using annualised trade return, maximum drawdown and Sharpe ratio metrics.

- **ER**: the annualised trade return focuses on the profitability of a trading strategy.

$$ER = E(R_t) * 240 \tag{17}$$

- **MDD**: the maximum drawdown represents the maximum loss from any peak in the trading process,

$$MDD = MAX(\frac{K_{t_i} - K_{t_j}}{K_{t_i}}, t_i < t_j) \tag{18}$$

- **SR**: the annualised Sharpe ratio compares the return of a trading strategy with its risk.

We show our results in Table 2, where volatility scaling is used for each method. This allows the volatility of the different methods to reach the same target, so we can directly compare these metrics such as annualised, maximum drawdown and sharpe ratio. And due to MC only could update until the end of the curtain, it cannot update online.

Figure 3 illustrates the profit curves for the Buy and Hold and different RL training methods for the test period 2013 to 2022. And Buy and Hold refers to a long position that begins during the test period and continues through the end of the test period. The Buy and Hold also represents the change in price of the IF300.

Table 2: Experiment results for the trading strategies volatility targeting.

Trading strategy	Update method	Method	ER	MDD	Sharpe
RL	offline	MC	0.288	0.564	0.855
		TD	0.284	0.504	0.833
		$TD(\lambda)$	0.315	0.562	0.937
	online	TD	0.296	0.472	0.869
		$TD(\lambda)$	0.336	0.518	1.003
Buy and Hold	/	/	0.096	1.013	0.281

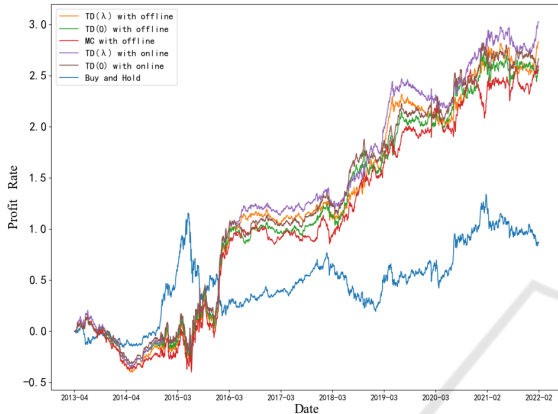


Figure 3: The cumulative trade returns for different trading strategies and methods.

We can see that the RL algorithm provided better performance for buy and hold for most of the time except for the period from May 2014 to June 2015. This is because the price of the IF300 continued to rise during this period. Therefore, the smartest trading strategy in these situations is to hold long positions and keep them. However, in other periods of high price volatility, the buy-and-hold strategy performed the worst. the RL algorithm was able to perform better in these markets by going long or short at a reasonable point in time.

Table 2 shows the performance results of Buy and Hold, and the different RL training methods when update the model offline and online in the test set quantitatively. The Buy and Hold has the lower annualised trade return 0.096 than the RL Algorithm. And the $TD(\lambda)$ method outperforms both MC and TD method under financial markets with partially observable Markov properties. The annualised trade return of $TD(\lambda)$ is 0.315 when update strategy offline and 0.336 when update strategy online. At the same time, online updates increase the profitability and decrease the MDD compared to offline updates. This is because the online update allows the strategy to avoid taking a large-loss action when it encounters a similar state later in the trading process after experiencing a large-loss state. These show that online updates allow

better reduce of MDD compared to offline.

The Sharpe ratio is an indicator that evaluates a combination of profit and trading risk. It can be seen from Table 2 that the $TD(\lambda)$ method when update online has the highest Sharpe ratio (1.003). Therefore, a good performance in term of the Sharpe ratio is expected for the $TD(\lambda)$ method with update online.

4 CONCLUSIONS

In this paper we analyze the specificity of financial transactions and model the transaction decision process as a partially observable Markov decision process. We discuss the respective performance characteristics of MC and TD methods and demonstrate that TD method is more influenced by the Markov properties of the environment, and the lower the Markov property degree, the less effective the strategy model trained using the TD method is. On the contrary, MC method is less affected by Markov property, but MC method puts the learning at the end of the curtain, so the efficiency is poorer. Therefore, in this paper, we use $TD(\lambda)$, which combines the two methods, for testing and analyze the effect of parametric lambda and truncated horizon h on $TD(\lambda)$. Considering the non-stationary nature of the financial time market, We test the model with update online for IF300 data from 2003 to 2022 and verify that the RL strategy outperforms the Buy and Hold and that the $TD(\lambda)$ method outperforms MC and TD. Furthermore, online updates allow the model to keep up with changes in financial market conditions as much as possible.

In the next work, we will investigate adding a feature extraction model before the RL model and investigate more technical indicators so that the states can represent more historical information. Second, different reward functions, such as using Sharpe ratio DDL, are considered so that the model can improve profitability while also considering factors such as risk. In addition, to further validate the effectiveness of our model, we will extend the experiment to other markets, such as stocks, futures, FX, etc.

REFERENCES

- Abarbanell, J. S. and Bushee, B. J. (1998). Abnormal returns to a fundamental analysis strategy. *Accounting Review*, pages 19–45.
- Carapuço, J., Neves, R., and Horta, N. (2018). Reinforcement learning applied to forex trading. *Applied Soft Computing*, 73:783–794.
- Chan, E. (2013). *Algorithmic trading: winning strategies and their rationale*, volume 625. John Wiley & Sons.
- Chen, L. and Gao, Q. (2019). Application of deep reinforcement learning on automated stock trading. In *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, pages 29–33. IEEE.
- Cultrera, L., Seidenari, L., Becattini, F., Pala, P., and Del Bimbo, A. (2020). Explaining autonomous driving by learning end-to-end visual attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 340–341.
- Deng, Y., Bao, F., Kong, Y., Ren, Z., and Dai, Q. (2016). Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3):653–664.
- Huang, C. Y. (2018). Financial trading as a game: A deep reinforcement learning approach. *arXiv preprint arXiv:1807.02787*.
- Lei, K., Zhang, B., Li, Y., Yang, M., and Shen, Y. (2020). Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading. *Expert Systems with Applications*, 140:112872.
- Lim, B., Zohren, S., and Roberts, S. (2019). Enhancing time-series momentum strategies using deep neural networks. *The Journal of Financial Data Science*, 1(4):19–38.
- Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier.
- Mao, H., Alizadeh, M., Menache, I., and Kandula, S. (2016). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*, pages 50–56.
- Mihatsch, O. and Neuneier, R. (2002). Risk-sensitive reinforcement learning. *Machine learning*, 49(2):267–290.
- Moody, J., Wu, L., Liao, Y., and Saffell, M. (1998). Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting*, 17(5-6):441–470.
- Murphy, J. J. (1999). *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin.
- Schwager, J. D. (2017). *A Complete Guide to the Futures Market: Technical Analysis, Trading Systems, Fundamental Analysis, Options, Spreads, and Trading Principles*. John Wiley & Sons.
- Shalev-Shwartz, S., Shammah, S., and Shashua, A. (2016). Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*.
- Singhvi, S. (1988). How to make money in stocks: A winning system in good times or bad. *Management Review*, 77(9):61–63.
- Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
- Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.
- Zhang, Z., Zohren, S., and Roberts, S. (2020). Deep reinforcement learning for trading. *The Journal of Financial Data Science*, 2(2):25–40.
- Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710.