Automated Stock Trading using Trust Region Policy Optimization

Chima Ezeilo
University of Texas at Austin
chimae@utexas.edu

Charles Nimo
University of Texas at Austin
nimo@utexas.edu

Video: https://youtu.be/zOqBT7gHhWQ
Github: https://github.com/princenimo/DRL-for-Automated-Stock-Trading-.git

Abstract

Everyday millions of traders around the world aim to trade stocks to make money. However, stock trading has never been easy. Stock prices depend on multiple factors and it is very difficult to develop a good strategy and make decisions such as when to buy? when to hold? and when to sell? Lately, Deep Reinforcement learning (DRL) agents have proven to show great promise in games such as Chess and Go. However, it remains a big challenge to design a profitable strategy in a complex and dynamic stock market. This paper explores the application of Trust Region Policy Optimization (TRPO), a policy gradient method,to learn an optimal strategy for high portfolio automated stock trading. We model the stock trading process as a Markov Decision Process (MDP) and then formulate our trading goal as a maximization problem. This agent learns to automatically position itself to win the market, specifically, it decides where to trade, at what price, and what quantity. We also train three other reinforcement learning algorithms, Soft Actor Critic (SAC), Proximal Policy Optimization(PPO), and Twin Delayed DDPG (TD3) to serve as baselines to compare their performances.In this experiment, we choose the Dow Jones Industrial Average (DJIA) 30 constituent stocks as they are the most popular stocks for portfolio allocation. We demonstrate the credibility and advantages of TRPO in financial markets for strategic decision making in portfolio allocation.

1 Introduction

The stock exchange floor was long ago the main hub for market transactions. It was central to traders and brokers that actually did the buying, selling, and negotiating on the physical exchange floor. However, this was before the evolution of electronic trading platforms. Over the last decade, markets have seen the widespread adoption of Automated Trading Systems that are used to make investment decisions in a fully automatized way with greater speeds than any human equivalent. Profitable automated stock trading is crucial to investment companies and hedge funds. Hedge funds such as Citadel, Blackrock Advisors, Renaissance Technologies, and several others are employing mathematicians, physicists, and other scientists to develop sophisticated algorithms to extract trading signals from vast amounts of data and automate trading. However, very rarely do these hedge funds publish their profit-generating ”secret sauce” to keep their competitive advantage and not much can be found in literature.

Reinforcement learning has become increasingly popular over the last few years. Notably, MIT Technology Review has chosen Deep Learning and reinforcement learning as some of the most influential breakthrough technologies in the last decade. Most recently deep learning and reinforcement learning has been combined together to form the field of deep reinforcement learning (DRL) which has shown great promise in various fields ranging from games and robotics to business and finance. In 2017, Google DeepMind developed AlphaGo Zero which managed to consistently beat the world's best go players without human intervention (Adebiyi et al., 2014). Later, in 2019 researchers at Facebook and Carnegie Mellon University developed Pluribus, a poker bot that uses DRL to beat the world's leading poker players (Brown and Sandholm, 2019).

The stock market contains many interference factors, rapid change, and insufficient periodic data. Having such incomplete information, A single-objective supervised learning model would have difficulty dealing with such serialization decision problems. Thus reinforcement learning is one of the effective ways to solve such problems. Moreover, Deep Reinforcement learning has been one of the leading applications of artificial intelligence in the world of finance, portfolio optimization, and stock trading. Most of the work in this area is fairly new. Of late, many recent strategies have been explored using different recent advancements in DRL.

(Zhang et al., 2019) applied DQN, PG, and A2C algorithms to trade continuous futures contracts and Theate and Ernst present a novel trading strategy using Deep Q-Networks (DQN) (Théate and Ernst, 2020).

In this work, we model the stock trading process as a Markov Decision Process (MDP), taking into account the stochastic and interactive nature of the trading market. The agent learns to trade stocks over a period of time and aims to maximize profits. Trading with a large account, at any given time (episode), an agent observes its current state (balance, opening high/low prices, closing price,trading volume, technical indicators, and multiple levels of granularity), selects and performs an action (buy/sell/hold), observes a subsequent state and receives some reward signal (difference in portfolio position). We train an agent that decides where to trade, at what price, and what quantity. For this agent, we apply Trust Region Policy optimization (TRPO) an On-policy reinforcement algorithm on the environment to perform portfolio allocation of the Dow Jones 30 constituents stocks. Additionally, we train other policy gradient methods PPO, TD3, and SAC to serve as baselines to evaluate their performances against this approach.

2 Related Works

Automated stock trading has been examined extensively through the years. Traditional methods, such as those in (Yan and Guosheng, 2015) focus on making stock predictions based on time-series analysis models like Kalman Filters, autoregressive models and logistic regression analysis (Adebiyi et al., 2014). For these kind of models,provided an indicator of stock price, they can represent it as a stochastic process and take the historical data to fit the process.

However these approaches for stock prediction have three main limitations. Firstly, these models rely heavily on the selection of indicators which are usually chosen manually and even harder to optimize without expertise finance knowledge. Secondly, the hypothesized stochastic processes aren't always compatible with the real world volatile stocks. Finally, the models can only consider a few indicators because their inference capability increases exponentially as the number of indicators increases. Thus, it is very difficult for these models to describe stock that is influenced by many factors.

There have been other approaches in the past which have developed automated stock trading strategies. In (Feng et al., 2004) there are two agents, the first is designed to exploit market volatility without considering the directions of price movement. It uses a pair of buy and sell orders of the same volume on a single stock at the same time without predicting the future movement of the stock. The second agent only uses technical analysis where it is designed to do the opposite of the initial strategy of buying stock when the price goes up and selling when it goes down.

There have been other successful strategies that make use of reinforcement learning to achieve good performance in non stationary environments using little knowledge given. In this work, they use tile coding (Yu and Stone, 2003; Feng et al., 2004) to allow for generalization to unseen instances of the continuous state-action space. The second agent uses linear regression model of market dynamics to guide order placement. This helps it to find long-term trends in price fluctuations. The third approach presented in this research combines seeks to combine both regression-based price prediction and market making to execute successful trading strategies. However, in comparison to our work these traditional methods fail to fully capture a wide range of factors that influence market performance (Sherstov and Stone, 2005).

More modern methods, however, use a variety of machine learning strategies to tackle the automated trading problem. Many approaches aimed to combine Reinforcement Learning with Deep Learning and Natural Language Processing models such as Recurrent Neural Networks (RNN), Gated Recurrent Units (GRUs), and Long-Short Term Memory cells (LSTMs). In (Chen and Gao, 2019), the authors used a variety of a Deep Q-Network (DQN) called Deep Recurrent Q-Network (DRQN) which uses an RNN at the basis of the DQN to process temporal sequences better. Similarly, the authors in (Wu et al., 2020) introduced GRUs to extract features from the stock market for its Deep Reinforcement Q-Learning System. The GRUs took raw data in the form of Opening, High, Low, Close Values (OHLCV) and technical indicators such as Moving Average (MA), Exponential Moving Average (EMA), and Moving Average Convergence/Divergence (MACD) to capture the patterns or features behind the dynamic states of the market. In (Deng et al., 2017), the authors implemented a deep RL based algorithmic trading agent using real-time data to question the possibility of beating human experienced human traders. They also used an RNN to extract information from technical indicators to feed to the RL agent as well as Deep and Fuzzy Learning for feature extraction.

Similar to all three authors we will also be implementing a Deep Reinforcement Learning approach. However, we will have a much looser restriction on the amount of stocks we trade (one stock in the first two authors' case), and will work with a portfolio containing a variety of different stocks; unlike the third author which restricted their model to 3 contracts (2 commodities: Sugar and Silver, and one stock-index future IF). Additionally, through FinRL we also utilize backtesting which can collect a variety of risk measurements, such as Sharpe ratio, which weren't considered in the first authors' study.

In (Liu et al., 2020b), the authors proposed an ensemble system using deep RL for portfolio management. The ensemble model has 3 algorithms at its core (PPO, A2C, DDPG), and works by having a growing training window and a rolling validation window time period which are chosen continuously to pick the best performing algorithm over the validation set. The motivation behind this is that each trading agent is sensitive to different types of trends. One agent performs well in a bullish trend, but acts bad in a bearish trend while another is adjusted to a more volatile market. We differ by not using an ensemble strategy, but testing the performance of TRPO with A2C, PPO, DDPG, and DQN as our baseline. Although, similar to the authors we both tackle the issue of automated portfolio investment.

While the use of DL with RL to automate stock trading looks promising, one aspect that is commonly overlooked is the impact of real-world events on the value of different stocks. For example, with the rise of COVID-19 many airline stocks decreased sharply in value. Additionally, cryptocurrencies such as Dogecoin also experienced an increase in value in some point of time after tweets from Elon Musk and exposure to TikTok. In (Azhikodan et al., 2019), the authors combined a deep RL approach with sentiment analysis to predict future movements based on financial news. A Recurrent Convolutional Neural Network was used for classifying news sentiment while DDPG was used for the RL agent. However, unlike the authors we will be accounting for risk by using measurements such as the Sharpe Ratio. Also, we will not be incorporating any sentiment analysis into our model (maybe for future work).

3 Methods

3.1 Environment

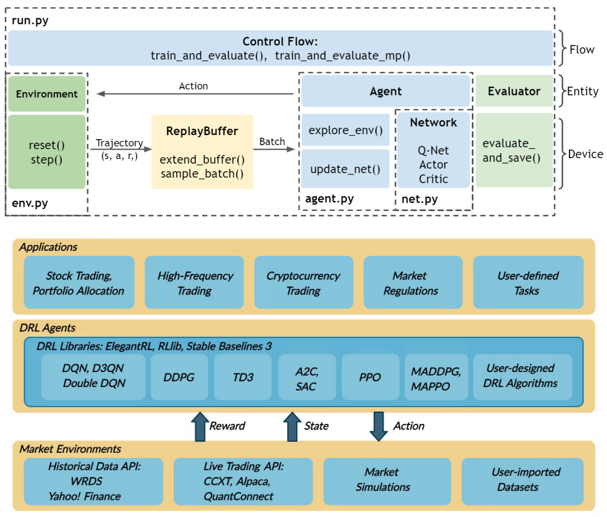

Figure 1: (Top) Elegant RL Framework and (Bottom) Fin RL Framework

The environment we will be working in is the Elegant-Fin RL environment created by the AI4Finance Foundation (Liu et al., 2021b, 2020a, 2021a); an overview of the environments are shown in Figure 1. These environments are designed for single stock trading, with the trading process being modeled as a Markov Decision Process (MDP) and the trading goal gets formulated as a maximization problem. The state space consists of a variety of information (features) a trader may use to execute a trade. This information consists of the balance at time t, the number of shares (for each stock) at time t, a variety of technical indicators (ie. high, low, open, close, volume, relative strength index), a turbulence index to measure extreme asset price fluctuation, and the Chicago Board Options Exchange's (CBOE) Volatility Index (VIX) which measures a constant 30-day expected volatility of the market. The action space is a D dimensional vector (D being the number of stocks owned) in which the values ranges from $\{-k, ..., -1, 0, 1, ..., k\}$ where k denotes the number of shares to buy/sell for any given stock. A negative value means the agent will sell k shares, a positive value means the agent will buy k shares, and a value of 0 means the agent will hold onto its shares for that stock. The reward function is the change in portfolio value when an action a is taken in state s and the agent arrives at the new state s' ($r(s, a, s') = v' - v$, where v' and v represent the portfolio values at state s and s' respectively).

3.2 Data Collection

The data for the stocks are collected using the Yahoo Finance API. The stock data can come from a variety of stock index markets such as Dow Jones, NASDAQ, SP 500, Hang Send, CSI 300, and many more. When retrieving the data, the start and end date for the trading information can also be specified as well as which tickers to collect (ie. AAPL and MSFT). For this study we do a comparative evaluation of TRPO against other reinforcement learning algorithms - TD3, PPO, and SAC to serve as baselines against this approach.

Table 1: Technical Indicators

Technical Indicator	Definition
Open	Price at beginning of trade period
High	Maximum Price in trade period
Low	Minimum Price in trade period
Close	Price at end of trade period
Moving Average Convergence Divergence	Relationship between two moving averages
Bollinger Upper Band	Std. Dv. Level Above Moving Price
Bollinger Lower Band	Std. Dv. Level Below Moving Price
Relative Strength Index (30)	Magnitude of recent price changes over 30 days
Commodity Channel Index (30)	Current price level relative to average price level over 30 days
Directional Movement Index	Magnitude of price directionality
Close Simple Moving Average (30)	Average of Closing prices over 30 days
Close Simple Moving Average (60)	Average of Closing prices over 60 days

Table 2: Collected Measures

Measure	Meaning
Annual Return	Return of an investment over a year
Cumulative Return	Total Return from a certain period of time up to the present
Annual Volatility	Variance of returns over a year
Sharpe Ratio	Return of an investment compared to its risk
Calmar Ratio	Measures risk-adjusted performance
Max Drawdown	Indicator of downside risk over a specified time period
Omega Ratio	Weighted risk-return ratio for a given level of expected return
Sortino Ratio	Variation of the Sharpe ratio that only factors in downside risk
Tail Ratio	Ratio between the 95th and (absolute) 5th percentile of the daily returns distribution
Daily Value at Risk	Predicts the greatest possible losses over a daily time frame.

3.3 Trust Region Policy Optimization

TRPO is an on-policy, policy gradient algorithm which updates policies while satisfying a constraint on how close the new/old policies are allowed to be (Schulman et al., 2015; Achiam, 2018). The constraint is in terms of the Kullback-Leibler divergence (measure of how close probability distributions are), and the pseudocode for the algorithm is shown in Figure 2. The constraint is called the trust region constraint and is added to make sure that our agent is in a safe region. In short, in TRPO, we take a step toward the direction that improves our policy and maximizes our reward but we also need to make sure that the trust region constraint is satisfied. It uses conjugate gradient descent to optimize the network parameter while satisfying the constraint. TRPO guarantees monotonic policy improvement and has also achieved excellent results in various continuous environments.

```
Algorithm 1 Trust Region Policy Optimization
1: Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2: Hyperparameters: KL-divergence limit $\delta$, backtracking coefficient $\alpha$, maximum number
   of backtracking steps $K$
3: for $k = 0, 1, 2, ...$ do
4:     Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
5:     Compute rewards-to-go $\hat{R}_t$.
6:     Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based
       on the current value function $V_{\phi_k}$.
7:     Estimate policy gradient as
           $\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$
8:     Use the conjugate gradient algorithm to compute
           $\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k,$
       where $\hat{H}_k$ is the Hessian of the sample average KL-divergence.
9:     Update the policy by backtracking line search with
           $\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k,$
       where $j \in \{0, 1, 2, ...K\}$ is the smallest value which improves the sample loss and
       satisfies the sample KL-divergence constraint.
10:    Fit value function by regression on mean-squared error:
           $\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$
       typically via some gradient descent algorithm.
11: end for
```

Figure 2: Pseudocode for the Trust Region Policy Optimization algorithm

3.4 Proximal Policy Optimizaton

Proximal Policy Optimization, abbreviated as PPO is an on-policy algorithm. It can be applied towards environments with both discrete and continuous action spaces. There are two classes of PPO: PPO-Penalty and PPO-clip.

PPO-Penalty approximates a KL-constrained update like TRPO but penalizes the KL-divergence in the objective function if the new policy is different from the old policy. It automatically adjust the penalty coefficient during training to scale appropriately.(Schulman et al., 2017)

```
Input: initial policy parameters $\theta_0$, initial KL penalty $\beta_0$, target KL-divergence $\delta$
for $k = 0, 1, 2, ...$ do
    Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
    Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
    Compute policy update
        $\theta_{k+1} = \arg\max_\theta \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta||\theta_k)$
    by taking $K$ steps of minibatch SGD (via Adam)
    if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \geq 1.5\delta$ then
        $\beta_{k+1} = 2\beta_k$
    else if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \leq \delta/1.5$ then
        $\beta_{k+1} = \beta_k/2$
    end if
end for
```

Figure 3: PPO with Adaptive KL Penalty

PPO-clip, in comparison to PPO-penalty doesn't have a KL-divergence term. It uses a specialized clipping in the objective function to limit the size of the update so that the difference between the new and old policies remain small (Schulman et al., 2017)

```
Input: initial policy parameters $\theta_0$, clipping threshold $\epsilon$
for $k = 0, 1, 2, \dots$ do
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update
      $\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$
  by taking $K$ steps of minibatch SGD (via Adam), where
      $\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k} \left[ \sum_{t=0}^{T} \left[ \min(r_t(\theta)\hat{A}_t^{\pi_k}, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right) \hat{A}_t^{\pi_k}) \right] \right]$
end for
```

Figure 4: PPO with Clipped Objective

3.5 Twin Delayed DDPG

Twin Delayed Deep Deterministic policy Gradient or TD3 for short has taken over for the replacement of DDPG, a actor-critic algorithm. TD3 addresses the stability and inefficiency challenges found in DDPG. It learns two Q networks rather than one which helps to overcome the maximization bias. During training, the policy and the target networks are updated more slowly, hence where the reason for the word "delayed" in the algorithm's name. (Fujimoto et al., 2018)

```
Initialize critic networks $Q_{\theta_1}$, $Q_{\theta_2}$, and actor network $\pi_\phi$
with random parameters $\theta_1$, $\theta_2$, $\phi$
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ to $T$ do
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
  $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
  Store transition tuple $(s, a, r, s')$ in $\mathcal{B}$

  Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \mathrm{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \mathrm{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if $t$ mod $d$ then
    Update $\phi$ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for
```

Figure 5: Twin Delayed DDPG Algorithm

3.6 Soft Actor Critic

Soft Actor Critic (SAC) method is an algorithm that uses stochastic policies, entropy regularization, and a few other tricks to stabilize learning and score higher than DDPG. It uses entropy that measures how diverse/random the actions suggested by a policy are as part of the reward to encourage exploration. With entropy regularization, the agent gets a bonus reward at each time step. Learning can be accelerated later on by increasing entropy but may also prevent the policy from converging prematurely to a bad local optimum. SAC can be used with both continuous and discrete action spaces. (Haarnoja et al., 2018)

```
Initialize parameter vectors $\psi$, $\bar{\psi}$, $\theta$, $\phi$.
for each iteration do
  for each environment step do
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step do
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
  end for
end for
```

Figure 6: Soft Actor Critic Algorithm

4 Experimental Setup

Tables 4 and 5 show the stock ticker information and technical indicators that we use, respectively. Table 6 shows the additional hyperparameters we use for our algorithms. Our baseline models are: Proximal Policy Optimization (PPO) (Schulman et al., 2017), Twin Delayed DDPG (TD3) (Fujimoto et al., 2018), and Soft Actor-Critic (SAC) (Haarnoja et al., 2018). All of which are implemented in the base code for the Elegant-Fin RL environment. We adapt the TRPO implementation from (Kostrikov, 2017) and adjust it to create an agent which can train in the Elegant-Fin RL framework. We run the model for 32,768 timesteps, evaluating it once we hit the $start_eval_date$ (specified in the hyperparameter table) using backtesting to get an unbiased measure of performance for the agent. The measures we collect and their meaning are showed in table 7.

5 Results

Table 8 shows the performance of all the models in terms of the collected metrics. TD3 obtained the highest annual and cumulative returns with TRPO trailing behind; meaning that these two models made the most money after backtesting. PPO obtained the highest annual volatility, making it the riskiest agent. This is further corroborated with it also having the highest (most negative) daily value at risk, max drawdown, and tail ratio. Its tail ratio of 0.78 means that losses are 1.28 times as bad as profits; so out of all the agents, losses are not as bad as profits! TRPO gets the highest Sharpe and Calmar ratio; implying that it performs well with risk-free investments and risk-adjusted returns respectively. The Sortino ratio is similar to the Sharpe ratio so it is no surprise that TRPO also has the highest Sortino ratio as well. The omega ratio determines the chances of winning in comparison to losing for a given investment. Since TRPO has the highest omega ratio, this means it took good risks and has a high chance of winning on the risks that it takes.

Table 3: Agent Backtesting Results

Algorithm	**Ann. Return**	**Cumu. Return**	**Ann. Volatility**	**Sharpe Ratio**	**Calmar Ratio**	**Max Drawdown**	**Omega Ratio**	**Sortino Ratio**	**Tail Ratio**	**D.V. at Risk**
TRPO	23.004%	51.424%	17.735%	1.26	1.74	-13.189%	1.28	1.81	1.07	-2.146%
SAC	16.225%	35.162%	29.095%	0.66	0.42	-39.076%	1.15	0.92	0.94	-3.589%
TD3	25.047%	57.41%	30.251%	0.90	0.64	-40.009%	1.20	1.31	0.93	-3.703%
PPO	18.285%	40.006%	30.635%	0.70	0.46	-40.14%	1.16	0.98	0.78	-3.774%

6 Conclusion & Future Works

From the results, we can see that TRPO performed better than the other baselines, excluding TD3, in terms of gaining money. However, TRPO handled risk much better than TD3 did. Due to stochasticity involving hyperparameters, in the future we plan on tuning each model to its best version with their own unique hyperparameters instead of using the same ones for all of them (for all we know, these may have benefited TRPO/TD3!). Our implementation of the stock trading agent involves observations such as the balance, opening high/low prices, and closing price however other factors like open limit orders can be included in the state. We can also look to optimize the reward function beyond the the profit obtained in a particular episode by including more risk factors which differs depending on the standard deviation of the data. It would be beneficial to also compare additional reward functions such as Differential Sharpe Ratio or Asymmetrical Damping . Sentiment analysis can be incorporated to better predict the stock trends that is given as an input in order to obtain more information for decision-making. Additionally we seek to train the agents in a different environment, particularly the cryptocurrency environment where risks can be much more rewarding and punishing.

References

Joshua Achiam. 2018. Spinning Up in Deep Reinforcement Learning.

Ayodele Adebiyi, Aderemi Adewumi, and Charles Ayo. 2014. Comparison of arima and artificial neural networks models for stock price prediction. *Journal of Applied Mathematics*, 2014:1–7.

Akhil Raj Azhikodan, Anvitha G. K. Bhat, and Mamatha V. Jadhav. 2019. Stock trading bot using deep reinforcement learning.

Noam Brown and Tuomas Sandholm. 2019. Superhuman ai for multiplayer poker. *Science*, 365:885 – 890.

Lin Chen and Qiang Gao. 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.

Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. 2017. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, 28(3):653–664.

Yi Feng, Ronggang Yu, and Peter Stone. 2004. Two stock-trading agents: Market making and technical analysis. In Peyman Faratin, David C. Parkes, Juan A. Rodriguez-Aguilar, and William E. Walsh, editors, *Agent Mediated Electronic Commerce V: Designing Mechanisms and Systems*, volume 3048 of *Lecture Notes in Artificial Intelligence*, pages 18–36. Springer Verlag.

Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

Ilya Kostrikov. 2017. pytorch-trpo. `https://github.com/ikostrikov/pytorch-trpo`.

Xiao-Yang Liu, Zhaoran Wang Li, Zechu, and Jiahao Zheng. 2021a. ElegantRL: Massively parallel framework for cloud-native deep reinforcement learning. `https://github.com/AI4Finance-Foundation/ElegantRL`.

Xiao-Yang Liu, Hongyang Yang, Qian Chen, Runjia Zhang, Liuqing Yang, Bowen Xiao, and Christina Dan Wang. 2020a. FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance. *Deep RL Workshop, NeurIPS 2020*.

Xiao-Yang Liu, Hongyang Yang, Jiechao Gao, and Christina Dan Wang. 2021b. FinRL: Deep reinforcement learning framework to automate trading in quantitative finance. *ACM International Conference on AI in Finance (ICAIF)*.

Xiao-Yang Liu, Shan Zhong, and Anwar Walid. 2020b. Deep reinforcement learning for automated stock trading: An ensemble strategy. *SSRN Electronic Journal*.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. 2015. Trust region policy optimization.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms.

Alexander Sherstov and Peter Stone. 2005. Three automated stock-trading agents: A comparative study. In P. Faratin and J.A. Rodriguez-Aguilar, editors, *Agent Mediated Electronic Commerce VI: Theories for and Engineering of Distributed Mechanisms and Systems (AMEC 2004)*, volume 3435 of *Lecture Notes in Artificial Intelligence*, pages 173–187. Springer Verlag, Berlin.

Thibaut Théate and Damien Ernst. 2020. An application of deep reinforcement learning to algorithmic trading.

Xing Wu, Haolei Chen, Jianjia Wang, Luigi Troiano, Vincenzo Loia, and Hamido Fujita. 2020. Adaptive stock trading strategies with deep reinforcement learning methods. *Information Sciences*, 538:142–158.

Xu Yan and Zhang Guosheng. 2015. Application of kalman filter in the prediction of stock price.

Ronggang Yu and Peter Stone. 2003. Performance analysis of a counter-intuitive automated stock-trading agent. In *Proceedings of the 5th International Conference on Electronic Commerce*, ICEC '03, page 40–46, New York, NY, USA. Association for Computing Machinery.

Zihao Zhang, Stefan Zohren, and Stephen Roberts. 2019. Deep reinforcement learning for trading.

Table 4: Stock Tickers Symbols/Names

Symbol	Company Name
AAPL	Apple
MSFT	Microsoft
JPM	JPMorgan Chase & Co.
V	Visa
RTX	Raytheon Technologies
PG	Procter & Gamble
GS	Goldman Sach
NKE	Nike
DIS	Walt Disney
AXP	American Express
HD	Home Depot
INTC	Intel
WMT	Walmart
IBM	International Business Machines
MRK	Merck & Co.
UNH	UnitedHealth Group
KO	Coca-Cola
CAT	Caterpillar
TRV	Travelers
JNJ	Johnson & Johnson
CVX	Chevron
MCD	McDonald's
VZ	Verizon Communications
CSCO	Cisco Systems
XOM	Exxon Mobil
BA	Boeing
MMM	3M
PFE	Pfizer
WBA	Walgreens
DD	DuPont de Nemours

Table 6: Hyperparameters

Hyperparameter	Value
Gamma	0.99
Max Stock	100
Initial Capital	$1e6$
Buy Cost Pct	0.001
Sell Cost Pct	0.001
Start Date	$2009-01-01$
Start Eval Date	$2019-01-01$
End Eval Date	$2021-01-01$
Batch Size	1024
Target Step	4096
Repeat Times	8
Market	Dow Jones
Tau	0.97
l2 reg	0.001
max k1	0.01
damping	0.1

Table 5: Technical Indicators

Technical Indicator	Definition
Open	Price at beginning of trade period
High	Maximum Price in trade period
Low	Minimum Price in trade period
Close	Price at end of trade period
Moving Average Convergence Divergence	Relationship between two moving averages
Bollinger Upper Band	Std. Dv. Level Above Moving Price
Bollinger Lower Band	Std. Dv. Level Below Moving Price
Relative Strength Index (30)	Magnitude of recent price changes over 30 days
Commodity Channel Index (30)	Current price level relative to average price level over 30 days
Directional Movement Index	Magnitude of price directionality
Close Simple Moving Average (30)	Average of Closing prices over 30 days
Close Simple Moving Average (60)	Average of Closing prices over 60 days

Table 7: Collected Measures

Measure	Meaning
Annual Return	Return of an investment over a year
Cumulative Return	Total Return from a certain period of time up to the present
Annual Volatility	Variance of returns over a year
Sharpe Ratio	Return of an investment compared to its risk
Calmar Ratio	Measures risk-adjusted performance
Max Drawdown	Indicator of downside risk over a specified time period
Omega Ratio	Weighted risk-return ratio for a given level of expected return
Sortino Ratio	Variation of the Sharpe ratio that only factors in downside risk
Tail Ratio	Ratio between the 95th and (absolute) 5th percentile of the daily returns distribution
Daily Value at Risk	Predicts the greatest possible losses over a daily time frame.

Table 8: Agent Backtesting Results

Algorithm	**Ann. Return**	**Cumu. Return**	**Ann. Volatility**	**Sharpe Ratio**	**Calmar Ratio**	**Max Drawdown**	**Omega Ratio**	**Sortino Ratio**	**Tail Ratio**	**D.V. at Risk**
TRPO	23.004%	51.424%	17.735%	1.26	1.74	-13.189%	1.28	1.81	1.07	-2.146%
SAC	16.225%	35.162%	29.095%	0.66	0.42	-39.076%	1.15	0.92	0.94	-3.589%
TD3	25.047%	57.41%	30.251%	0.90	0.64	-40.009%	1.20	1.31	0.93	-3.703%
PPO	18.285%	40.006%	30.635%	0.70	0.46	-40.14%	1.16	0.98	0.78	-3.774%