Deep Reinforcement Learning for Optimal Stock Trading in Various Market Trends

Peter Li[*] Tony Lee[*]

Abstract

Stock trading plays a crucial role in the growth of industry, commerce and the overall economy. However, it is extremely challenging to obtain an optimal strategy in a dynamic market and thus, it is often considered a high-risk financial investment. Reinforcement Learning provides mechanisms to navigate this large and complex space. In this project, we measure the asset-management performance of various deep reinforcement learning algorithms in different market trends.

1. Introduction

Training intelligent agents for stock trading is a time-honored topic, notably beginning with the expansion of commercial computer technology into finance in the 1990s. Nowadays, it is easy for anyone with access to the web to set up an online brokerage account and begin trading from their own personal device. Prices are readily accessible and the time between each price update has decreased significantly, often occurring within fractions of a second. This has made it nearly impossible for a human to respond and act on these rapidly changing conditions, while a machine can do so with relative ease. Additionally, computers have the advantage of taking the emotional aspect out of trading as even the most experienced financial traders may find themselves operating suboptimally due to their own human emotions.

The traditional approach in training a trading agent is performed in two steps (Rubinstein, 2002). First, the expected returns of the stocks and the covariance matrix of the stock prices are computed. Second, the best portfolio allocation is then found by either maximizing the return of the portfolio or minimizing the risk for a range of returns. The best trading strategy is then extracted by following the best portfolio allocation. Although the algorithm itself may seem simple, the approach can be very complicated to implement in an actual trading environment where variables such as price and transactions costs fluctuate.

Presently, most research on applying machine learning to trade financial assets is devoted to supervised learning techniques, such as the utilization of neural networks to predict future returns of financial assets and stock trading (Atsalakis & Valavanis, 2009). A whole set of supervised methods have been applied to these particular tasks, among them, support vector machines (Kim, 2003), tree-based algorithms (Booth et al., 2014) (Krauss et al., 2017) and nearest neighbor classification (Teixeira & De Oliveira, 2010). The general idea behind the majority of these works is as follows: first, a predictive model (e.g. a neural network or a random forest) is trained on historical data to forecast the asset's price change using a set of explanatory variables or features. Second, these forecasts are fed into a trading module to derive the actual trading action (e.g. buy stock) once the forecast surpasses a certain threshold.

Most supervised approaches are still following the traditional method. Many of the prediction models use CNN, LSTM or other RNN models and are able to successfully predict the future prices of stocks. However, the supervised methods have two major shortcomings: 1. Stock trading is a dynamic environment that demands quick decision-making and precise control in order to make money in a market. 2. Most supervised learning implementations for stock trading target just a single stock price and cannot manage an entire portfolio. In financial analytics, portfolio management is essential for risk control, as it provides the ability to balance risk and opportunity across the entire portfolio.

Another approach is to model the problem as a Markov Decision Process (MDP) and apply dynamic programming (White, 1969) to solve for the optimum strategy. However, the scalability of this model is limited due to the giant state space of the stock market (Neuneier, 1998).

For this project, we propose to use deep reinforcement learning to overcome the aforementioned shortcomings of the the supervised machine learning approaches. Reinforcement learning enables the combination of the "prediction" and the "portfolio construction" task into one integrated step, closely aligning the machine learning problem with the objectives of the investor. At the same time, important constraints, such as transaction costs, market liquidity, and the investor's degree of risk-aversion can be conveniently taken into account (Fischer, 2018). In particular, we are applying state-of-the-art reinforcement learning algorithms to the stock portfolio management problem and focusing on how these algorithms perform in various market trends. For the scope of this paper, we focused on the Dow Jones stock market.

There are many advantages in leveraging deep reinforcement learning for this particular problem. The first advantage is that deep neural networks possess the ability to exploit the unknown underlying structure of the input distribution and automatically learn the features. Therefore, we can convert the complicated covariance matrix of stock prices from the traditional approach to acquire even richer feature representations. The second advantage is that deep reinforcement learning algorithms allows us to explore large and even continuous state spaces, which algorithms such as dynamic programming could not. In particular, this is reflected in recent progress in research with Deep Q-Network(DQN) (Mnih et al., 2013a) and Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015). DQN and DDPG have also been applied to build a stock trading agent (Xiong et al., 2018) (Liang et al., 2018).

2. Related Work

As mentioned in the introduction, previous research has been done to apply deep reinforcement learning to stock trading. The following is a brief introduction of the previous work.

2.1. State-Of-The-Art Deep Reinforcement Learning Approaches

Reinforcement learning is the study of how an agent can interact with its environment to learn a policy which maximizes expected cumulative rewards for a given task. Recently, reinforcement learning has experienced dramatic growth in attention and interest due to promising results in controlling continuous systems in robotics (Lillicrap et al., 2015), playing Go (Silver et al., 2016), Atari (Mnih et al., 2013b), and competitive video games (Vinyals et al., 2017). In addition, policy gradient methods with neural network function approximators have been particularly successful in continuous control (Lillicrap et al., 2015), (Schulman et al., 2015), and (Schulman et al., 2017).

2.2. Machine Learning on Trading Financial Assets

As previously mentioned, there have been efforts to apply supervised learning to trade financial assets and stock trading ((Atsalakis & Valavanis, 2009), (Kim, 2003), (Booth et al., 2014) (Krauss et al., 2017), (Teixeira & De Oliveira, 2010)). For the scope of this project, we are instead using deep reinforcement learning to directly predict stock prices. We optimized for the future rewards so that the estimates and trading actions are integrated into one approach. We also handled the shortcomings of some supervised learning algorithms and its inability to handle a large state space by using reinforcement learning algorithms that can train agents to perform actions over a continuous space.

2.3. Stock Trading Features on Deep Learning

In *Deep Direct Reinforcement Learning for Financial Signal Representation and Trading*, some pioneer explorations have been done to apply reinforcement learning and deep learning to build a framework for financial signal processing and online trading, which entails using function approximation, deep recurrent neural network and other methods to directly approximate and evaluate the learned policy (Deng et al., 2017). They leveraged the deep recurrent neural network to automatically learn features for stock trading or technology indicators. Because of the generalization that deep neural networks provide, the feature technology indicators were defined by the algorithm without having to predefine them.
For this project, our approach to applying deep neural networks differs in that we directly apply neural networks for the value function and policy function approximation or policy gradient deep Reinforcement learning. Thus, the neural networks are part of the policy gradient and not in a separate network. The structure of the network becomes clearer and more direct, allowing us to easily adjust the structure, open to more advanced networks, alter reward functions, etc.

2.4. DDPG on Stock Trading

The paper, *Practical Deep Reinforcement Learning Approach for Stock Trading* (Xiong et al., 2018), explored the potential of training a Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) agent to learn stock trading strategy. Their results show that the trained agent outperforms the Dow Jones Industrial Average and min-variance portfolio allocation method in accumulated return. In addition, the comparison to Sharpe ratios indicate that their method is more robust than the predecessors in balancing risk and return. In a similar direction, we experimented using DDPG and other state-of-the-art policy gradient algorithms to gain insight into how well the different approaches perform specifically in various economic trends.

3. Background

3.1. Baselines and a Lucrative Oracle

We considered three different baseline algorithms (bandwagon, risk-averse and the Dow Jones Industrial Average). The first two baselines are greedy algorithms that mimic stock investing behaviors commonly seen among investors, but differ in how the locally optimal choices of buying stocks are made. The bandwagon baseline observes for a d-days period the top n companies that had the highest growth rate, liquidates its current assets and invests its money on the top n companies. The risk-averse baseline is a similar algorithm except it chooses the top n companies with the lowest variances in closing price over d-days period of time.

The oracle for our project has complete knowledge of the future prices of the Dow index, but with the following two limitations:

1. The oracle can trade once per day and can only sell after purchasing stocks.
2. The oracle can purchase stocks based on the Dow average alone (i.e. cannot make granular purchases on individual stocks).

We implemented an iterative solution where the oracle keeps the stocks if the Dow Index increases the following day and shorts if the price falls the next day.

3.1.1. Baseline and Oracle Results

We tested both the greedy baselines, Dow Jones Industrial Average (DJI) baseline and oracle implementation using the 2019 Dow Jones stocks with a starting fund of \$10000. For both greedy approaches, we set the observing period to 45 days, as this is the adequate time for the algorithm to gauge the market to purchase stocks for 5 companies. The bandwagon baseline ended with a total portfolio value $\$10248.22$ compared to the $\$10896.30$ of the risk-averse baseline. The risk-averse baseline performed better by $\$648.08$. The DJI baseline did even better by yielding a total portfolio value of $\$11466.60$. In striking contrast, the oracle yielded a final portfolio value of $\$34097.72$.

3.1.2. Deciding a Baseline

In the end, we decided to make the DJI baseline to be the lower bound when evaluating the performance of our deep reinforcement learning implementation. For most time periods, the DJI baseline outperformed the greedy baseline algorithms. In addition, most prior research that apply reinforcement learning algorithms for the stock trading problem typically use the DJI as the baseline because it is a widely used benchmark for financial asset portfolio evaluation.

3.2. Challenges

Judging from the gap between the DJI baseline and oracle, there are two major challenges when training an intelligent agent to manage a stock portfolio:

- The state space for this particular problem is infinite, as an investor can put varying amount of dollars into any of the Dow 30 companies. This problem is especially displayed when our oracle, which knows exactly when and how much to hold or short, greatly outperformed our baseline implementations, which rely only on immediate, past statistics.
- The environment is complex and changes dynamically. There are so many external factors (e.g. news, Tweets from presidents, war, etc.) that affect the market.

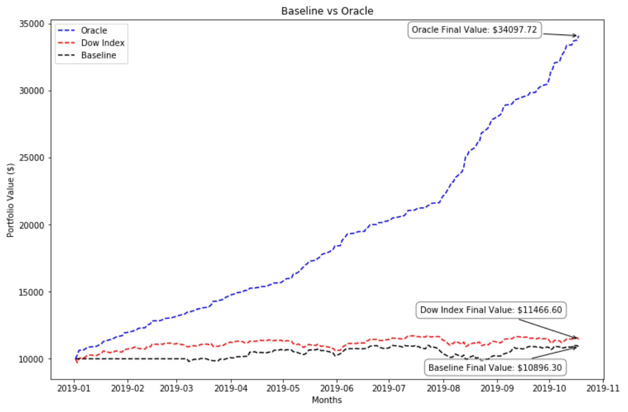

Figure 1. Comparison of the baseline algorithms versus the oracle. Out of the baseline algorithms, the DJI baseline performed the best. The lookahead oracle completely outperformed the baseline algorithms.

3.3. Reinforcement Learning for Stock Trading

Reinforcement learning (RL) is “learning what to do and how to map situations to actions, so as to maximize a numerical reward signal” (Sutton & Barto, 2018). In a typical framing of a RL scheme, an agent executes an action in an environment, which is then interpreted into a reward and a representation of the state which is fed back into the learning agent. In our particular case, the stock market itself is the environment from which the agent will learn the dynamics of the market to reach its goal to efficiently trade stocks to maximize expected returns. Furthermore, we consider the stock trading problem in discrete time steps $t = 0, 1, 2, 3, ...$, where at each time step, t, the agent is a new state given by the environment. The possible states are denoted as $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of possible states and the next possible action the agent can take (buy, hold or sell stock) is denoted as $a_t \in \mathcal{A}(s_t)$, where $\mathcal{A}(s_t)$ is the set of actions available in state s_t. The trading agent chooses an action according to policy π_t, where $\pi_t(s)$ represents the action chosen if $s_t = s$. On the subsequent time step $t + 1$, the agent receives a reward r_{t+1} as a consequence of action a_t, and information about the new state s_{t+1}.
The goal of the agent is to maximize its expected cumulative

rewards r_t. Therefore, the return is the sum of rewards

$$R_t = \sum_{k=0}^{T} r_{t+k+1} \tag{1}$$

obtained from a given time period t to some terminal time period T.
When $T < \infty$, it is an episodic task, and for the stock trading problem, we limited our approach to an episodic task.
It is often valuable to consider such optimization problems as having Markov property, which in turn implies that an agent is not expected to know everything important from all time points, but rather remember everything important that it has already experienced. This assumption allows us to apply some RL algorithms to our stock trading problem and obtain meaningful results (Sutton & Barto, 2018).

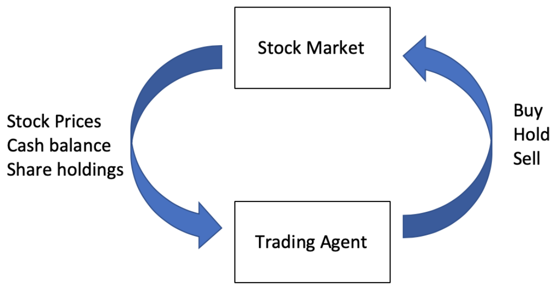

Figure 2. Reinforcement Learning for Stock Trading

We modeled the stock trading process as a Markov decision process (MDP), considering the stochastic and interactive nature of the trading market:

- State $s = [p, h, b]$: A set that includes the information of the prices of stocks $p \in R_*^D$, the amount of holdings of stocks $h \in Z_*^D$, and the remaining balance $b \in R_*$, where D is the number of stocks that we consider in the market and Z_* denotes non-negative integer numbers, and R_* denotes non-negative real numbers.

- Action a: A set of actions on all D stocks. The available actions of each stock include selling, buying, and holding, which result in decreasing, increasing and no change of the holdings h, respectively.

- Reward $r(s, a, s')$: The change of the portfolio value when action a is taken at state s and arriving at the new state s. The portfolio value is the sum of the equities in all held stocks $p^T h$ and balance b.

- Policy $\pi(s)$: The trading strategy of stocks at state s. It is essentially the probability distribution of a at state s.

- Action-value function $Q_\pi(s, a)$: The expected reward achieved by action a at state s following policy π.

The dynamics of the stock market is described as follow. With subscript t to denote time, the available actions on stock d is as follows:

- Selling: k ($k \in [1, h[d]]$, where $d = 1, ..., D$) shares can be sold from the current holdings, where k must be an integer. In this case, $h_{t+1} = h_t - k$.

- Holding: $k = 0$ and it leads to no change in h_t.

- Buying: k shares can be bought, which leads to $h_{t+1} = h_t + k$. In this case, $a_t[d] = k$ is a negative integer.

It is also important to note that we limit the stock purchases so that a negative balance cannot happen. That is, without loss of generality, at each step t, we sell shares first and then, buy stocks based on the current balance b. If remaining balance is not enough, we buy $min(\frac{b}{p_t[d]}, a_t[d])$ shares of d.
On the initial time step $t = 0$, the stock prices p_0 are set to the initial prices and b_0 is the initial fund available for trading. Furthermore, the h and $Q_\pi(s_t, a_t)$ are initialized as 0, and $\pi(s)$ is uniformly distributed among all actions for any state. $Q_\pi(s_t, a_t)$ is learned through interacting with the stock market.

3.4. Trading Goal as Return Maximization

According to Bellman Equation, the expected reward of taking action at is calculated by taking the expectation of the rewards $r(s_t, a_t, s_{t+1})$, plus the expected reward in the next state s_{t+1} (Bellman, 1966). Based on the assumption that the returns are discounted by a factor of γ, we have

$$Q_\pi(s_t, a_t) = \mathbf{E}_{s_{t+1}}[r(s_t, a_t, s_{t+1}) + \gamma \mathbf{E}_{a_{t+1}\ \pi(s_{t+1})}[Q_\pi(s_{t+1}, a_{t+1})]] \tag{2}$$

Our goal is to design a trading strategy that maximizes the investment return at a target time t_f in the future,i.e., $p_{t_f}^T h_t + b_t$, which is also equivalent to $\sum_{t=1}^{t_f - 1} r(s_t, a_t, s_{t+1})$. Thus, we format the problem to optimize the policy that maximize the action-value function $Q_\pi(s_t, a_t)$.
To evaluate and improve policies, most reinforcement learning methods use value functions which assign values to each state given a specific policy. The state value function is defined as

$$V_\pi(s) = \mathbf{E}_\pi[R_t | s_t = s] = \mathbf{E}_\pi[\sum_{k=0}^{T} \gamma^k r_{t+k+1} | s_t = s] \tag{3}$$

and the action value function is

$$Q_\pi(s,a) = \mathbf{E}_\pi[R_t|s_t = s, a_t = a] = \mathbf{E}_\pi[\textstyle\sum_{k=0}^{T} \gamma^k r_{t+k+1}|s_t = s, a_t = a] \tag{4}$$

Note that $\mathbf{E}_\pi[.]$ denotes the expectation given that the agent continues to follow the policy π.

For fairly small-scale problems, we can solve this optimization problem through dynamic programming if the model and reward is known, and Monte Carlo methods if the model or reward is not known. However, for larger problems or more importantly, continuous state and action space problems, it is unrealistic for each state to be visited enough times to produce a meaningful average and also computationally impractical to store the estimates. Therefore, we rely on function approximations of the Q_π and V_π functions through parameter updates. Concretely, the agent's goal is to choose actions such that it maximizes the expected cumulative reward that it will receive.

4. Approach

4.1. Deep Policy Gradient Methods

We applied state-of-the-art deep policy gradient methods to optimize our return. Since we are using a deep neural network for function approximations, we modified our expected reward from training parameters θ to $\rho(\theta, s_0)$, using the policy gradient theorem:

$$\nabla_\theta \rho(\theta, s_0) = \sum_s \mu_{\pi_\theta}(s|s_0) \sum_a \nabla Q_{\pi_\theta}(s,a) \tag{5}$$

Here, $\mu_{\pi_\theta}(s|s_0) = \sum_t \gamma^t P(s_t = s|s_0)$. All of our approaches are training a deep neural network with parameter θ to approximate value functions.

Deep Deterministic Policy Gradient (DDPG) uses actor-critic methods which estimate $Q_\pi(s,a)$ and optimize a policy that maximizes the Q-function (Lillicrap et al., 2015). It stores each transition in a replay buffer, maintains a separate target network (Mnih et al., 2013c) and applies batch normalization (Ioffe & Szegedy, 2015). These technique are essential for stability when training. To explore the action space, DDPG adds noise when selecting an action.

Trust Region Policy Optimization (TRPO) uses conjugate gradient descent (Shewchuk et al., 1994) as the optimization method with a KL constraint:

$$\mathbf{E}[KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \leq \delta \tag{6}$$

(Schulman et al., 2015). The KL divergence between the new and the old policy must be lower than δ, where δ is the size of the trust region. TRPO estimates trajectory advantages (Sutton et al., 2000). Proximal Policy Optimization (PPO) is similar to TRPO, but uses only first-order optimizations for a simpler implementation.

Algorithm 1 DDPG with variance penalty

```
Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with random weight $\theta^Q$ and $\theta^\mu$;
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
for episode= 1, $M$ do
  Initialize a random process $\mathcal{N}$ for action exploration;
  Receive initial observation state s1;
  for t = 1, T do
    Select action $a_t = \mu(s_t|\theta^\mu)$ according to the current policy and exploration noise;
    Execute action $a_t$ and observe reward $r_t$ and state $s_{t+1}$;
    Update reward $\overline{r}_t = r_t - \beta * log(1 + \sigma_t)$;
    Update variance $\sigma_t$ for step $t$ use updated reward $\overline{r}_t$;
    Store transition $(s_t, a_t, \overline{r}_t, s_{t+1})$ in $R$;
    Sample a random mini-batch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$;
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta\mu')|\theta Q')$;
    Update critic by minimizing the loss:
        $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:
        $\nabla_{\theta^\mu} J \approx$
        $\frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}(s|\theta^\mu)|_{s_i}$
    Update the target networks:
        $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
        $\theta^{\mu'} \leftarrow \tau\theta^{\mu'} + (1-\tau)\theta^{\mu'}$
  end for
end for
```

One important thing to note that both PPO and TRPO have the potential of converging to the local optimum instead of the global optimum. We hope to alleviate this by running the algorithms over multiple epochs.

5. Experiments

5.1. Stock Trading Data

Stock trading data is readily available with many different webservices offering through their API. Alpha Vantage (Torres) offers a free API to obtain real time financial data. For this project, we used Alpha Vantage to acquire daily trading data for the Dow Jones stock market and the Dow Jones Industrial Average (DJI) index data.

Date	Tic	Open	High	Low	Close	Adjusted Close	Volume
2019-01-02	AAPL	154.89	154.85	154.23	157.92	156.05	37039700
2019-01-02	IBM	122.01	115.98	111.69	115.21	111.236	4239900
2019-01-02	MSFT	99.55	101.75	98.94	101.12	99.9857	35329300

Figure 3. A few rows of Dow Jones stock data retrieved from Alpha Vantage

5.2. Experiment Environment

OpenAI has open-source software that makes it easy for researchers and developers to apply and test their reinforcement learning implementations. In particular, OpenAI Gym is a toolkit for developing and comparing various reinforcement learning algorithms (Brockman et al., 2016). Through the Gym API, consumers can create and register their own RL environment. In addition, OpenAI Baselines is a set of high-quality implementations of reinforcement learning algorithms that is also well-integrated into the Gym environment (Dhariwal et al., 2017).
With a TensorFlow backend, we implemented a stock trading environment using the states and buy-and-sell actions defined above, which was then plugged into Gym. For validation purposes, we then created a test environment which read the Dow Jones Industry Average data and compared it to the agent's return.

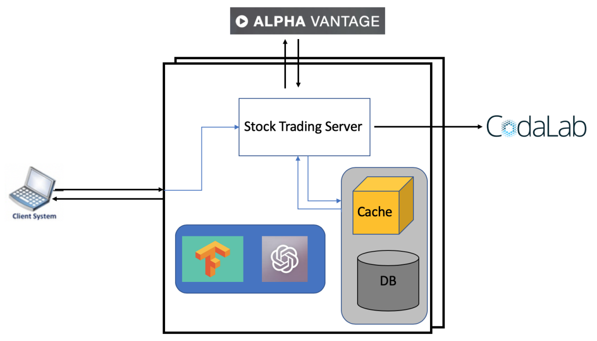

Figure 4. Architectural diagram of our smart stock trading system.

5.3. Experiment Settings

For all experiments, the starting cash for the agent was \$10000. From our initial experiments, a high starting cash input (e.g. \$100,000) led to huge variances in performance and great deviations away from the true convergence.
For the data, we split the Dow Jones stock dataset (January 2001 to August 2019) into a training, validation and testing set. We trained the RL agent from January 2001 to April 2008 using the DDPG, TRPO and PPO algorithms. Since the goal of our project is to train an agent to yield maximum return over various market trends, we had three separate validation sets, one for each trend: downward, upward and wave. Below is an enumeration of the four-month time periods we used for the validation set.

- Upward trend: March 2009 - June 2009
- Wave trend: May 2008 - August 2008
- Downward trend: September 2008 - December 2008

We then used the validation sets to determine the best hyperparameters for each of the three algorithms. In addition, we treated the RL algorithm itself as a hyperparameter for each market trend. For example, if algorithm A outperforms algorithms B and C for an upward trend, we would use algorithm A for our final tests in that specific market trend. For the training and validation phase, we ran the algorithms over 20 full epochs and used the statistics of the agent's returns (in particular the mean) to decide the performance of the algorithm. For the testing phase, we ran the algorithms over 50 runs using the previously trained model with fixed hyperparameters. The following is the four-month periods we used for the test set.

- Upward trend: January 2019 - April 2019
- Wave trend: May 2019 - August 2019
- Downward trend: September 2018 - December 2018

We also persisted the trade sequence the agent makes over a given time period and daily snapshots of its stock portfolio for each and every run.

5.4. Result

5.4.1. Post Validation

Using the trained agent, the agent yielded the following results for the various market trends reflected in the three validation sets. For downward trend (9/1/2008 - 12/13/2008), DDPG yielded a mean rate of return (MRoR) of -18.80%, PPO yielded a MRoR of -6.41% and TRPO yielded a MRoR of -5.70%. This is compared to the rate of return (RoR) of -24.73% of the DJI baseline. For upward trend (3/1/2009 - 6/30/2009), DDPG yielded a MRoR of 21.87%, PPO yielded a MRoR of 7% and TRPO yielded a MRoR of 7.47%. This is compared to the RoR of 24.89% of the DJI baseline. And finally, for wave trend (5/1/2008 - 8/31/2008), DDPG yielded a MRoR of -6.57%, PPO yielded a MRoR of -2.91% and TRPO yielded a MRoR of -2.85%. This is compared to the RoR of -11.27% of the DJI baseline.
Just from the validation phase of our experiment, there are few key observations to make:

1. All three RL algorithms performed better than the baseline in a down and wave market. In particular, TRPO yielded the best results in both trends (~8% better RoR than the baseline in a wave trend and ~18% better in a down market).
2. The three RL algorithms, plus the baseline only yield a profit in an uptrend market.
3. DDPG yield different results than TRPO/PPO in all three trends. In particular, DDPG does better in a

upward trend market by ˜15% and comes close to the baseline's performance. On the other hand, PPO and TRPO outperform DDPG in a wave and down market, ˜3% and ˜11% respectively.

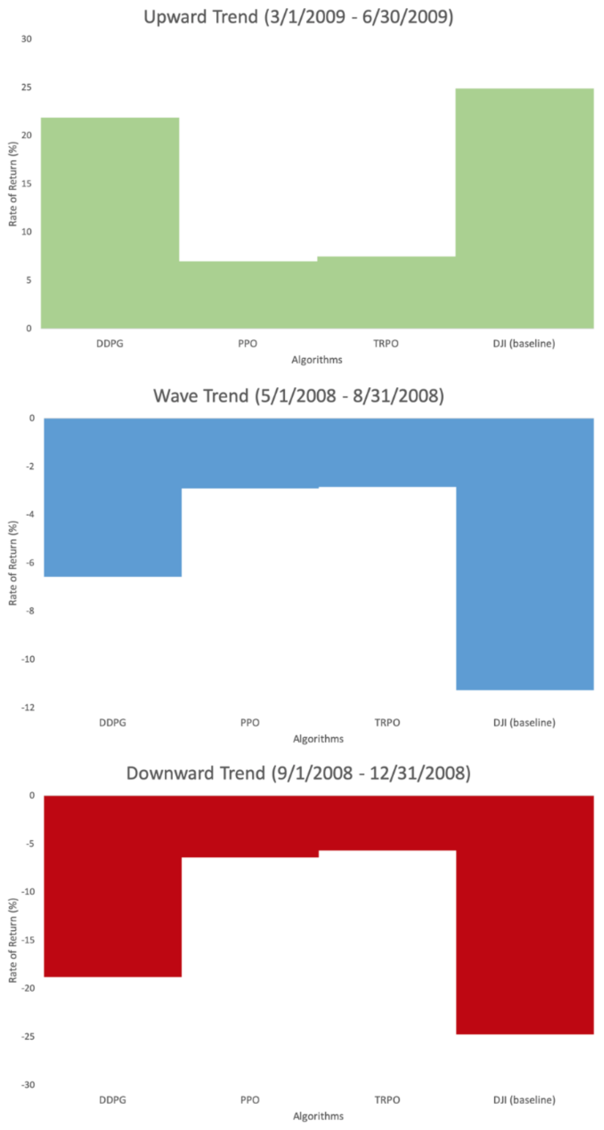

Figure 5. Comparison of DDPG, TRPO and PPO against the DJI baseline for the different validation sets. The y-axis of the chart is the mean rate of return for the starting cash of $10000.

In regards to the first observation, we believe the RL agents were able to outperform the baseline in a wave trend market due to the fact that wave trends are far more prevalent in the stock market than a upward or downward spike. This allows the agent to learn and navigate the fluctuating, yet relatively stable market. It is also interesting to see that the agent outperformed the baseline in an downward trending market. We believe this is due to fact that the agent was exposed to the 2007-2008 economic crisis towards the end of its training period.

For the second observation, we believe that with our current implementation of the RL algorithms, the agent is not intelligent enough to find hidden gems in such a rough market and tends to lose money, as compared to its performance in a upward-trending market.

Finally, we believe that DDPG performs better in a upward trending market than TRPO/PPO because of its sample efficiency, which is a prominent characteristic of off-policy methods. As stated previously, the agent was exposed to more wave and even down trends than upward spikes. DDPG was able to better navigate in a up-market even with less exposure to the particular trend during the training period. Additionally, we believe the stability and reduction of exploration noise over the training period for TRPO and PPO allows the agent to yield a better return in a down market, but explores less in a up-market.

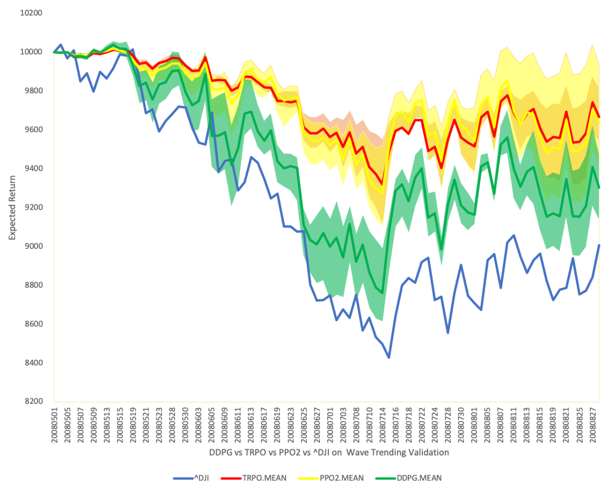

Figure 6. Comparison of daily portfolio values of DDPG, TRPO, PPO and DJI baseline during the wave-trending validation phase. Each algorithm started with $10000.

5.4.2. Final Tests

Based on the outcome of the validation phase of our experiment, we tested the trained TRPO-agent on a different, more recent downward and wave trend and the trained DDPG-agent for a more recent upward trend in the Dow 30 stock market (see *Figure 7* for the results).

The results confirm all the observations from the validation phase. Again the RL agents perform slightly better than the baseline in a down and wave market and underperform in a up market, TRPO fails to yield a profit in a down market

Trend	Algorithm	Minimum	Maximum	Std. Dev.	Median	Mean	DJI Baseline
Up	DDPG	10860.10	11835.44	246.19	11349.28	11341.36	11374.16
Down	TRPO	8548.36	9497.03	241.51	8951.34	8987.34	8886.40
Wave	TRPO	9685.92	10536.46	185.97	10184.46	10184.00	9989.84

Figure 7. Statistics for resulting portfolio values over 50 runs of the test set using the selected models from the validation phase.

and lastly, DDPG performs the best in a up market with a larger return than TRPO in a wave or down market.

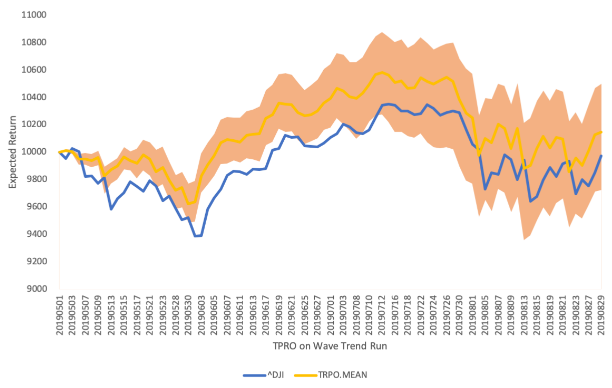

Figure 8. Comparison of daily portfolio values of TRPO and DJI during the wave trending testing phase. Each algorithm started with $10000.

6. Conclusion and Future Work

Our result shows that the recently developed deep reinforcement learning approaches, specifically DDPG, TRPO and PPO, are able identify the market dynamics with the trained agents having the potential to beat the Down Jones Index Average. In addition, market trends affect the performance of the RL agents and certain characteristics of the state-of-the-art RL algorithms are illuminated through various economic climates.
In the future, we want to explore how adding more features to our reward function (e.g. frequency of trades, variance of portfolio values, etc.) affects the performance of the three RL algorithms. We also want to come up with a theoretical approach to somehow alleviate the problem of a high starting cash value. Finally, we hope shift the focus and replicate the results not only in different stock markets (e.g. NASDAQ), but also apply it to other investment markets, such as cryptocurrency and real estate investing.

7. Code and Stock Dataset

The code for this project can be found in the following GitHub repository:

https://github.com/peter6888/cs221_project

The Dow Jones stock dataset, fetched from Alpha Vantage, can be found under the *Data_Daily_Stock_Dow_Jones_30* folder in the GitHub repository.

8. CodaLab Worksheet

The experiments and results of this paper are reproducible and can be found in the following CodaLab worksheet:

https://worksheets.codalab.org/worksheets/0x36e5257cae7c446b94be17076d643150

9. Acknowledgements

We would like to thank Andrew Tan for providing us mentorship and guidance for this project.

References

Atsalakis, G. S. and Valavanis, K. P. Surveying stock market forecasting techniques–part ii: Soft computing methods. *Expert Systems with Applications*, 36(3):5932–5941, 2009.

Bellman, R. Dynamic programming. *Science*, 153 3731: 34–7, 1966.

Booth, A., Gerding, E., and Mcgroarty, F. Automated trading with performance weighted random forests and seasonality. *Expert Systems with Applications*, 41(8):3651–3661, 2014.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Brunskill, E. When policies can be trusted: Analyzing a criteria to identify optimal policies in mdps with unknown model parameters, 2010. URL https://www.aaai.org/ocs/index.php/ICAPS/ICAPS10/paper/view/1459.

Deng, Y., Bao, F., Kong, Y., Ren, Z., and Dai, Q. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, 28(3):653–664, March 2017. ISSN 2162-237X. doi: 10.1109/TNNLS.2016.2522401.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. Openai baselines. https://github.com/openai/baselines, 2017.

Fischer, T. G. Reinforcement learning in financial markets-a survey. Technical report, FAU Discussion Papers in Economics, 2018.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Kim, K.-j. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1-2):307–319, 2003.

Krauss, C., Do, X. A., and Huck, N. Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the s&p 500. *European Journal of Operational Research*, 259(2):689–702, 2017.

Liang, Z., Chen, H., Zhu, J., Jiang, K., and Li, Y. Adversarial deep reinforcement learning in portfolio management, 2018.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning, 2015.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning, 2013a.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013b.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013c.

Neuneier, R. Enhancing q-learning for optimal asset allocation. In *Advances in neural information processing systems*, pp. 936–942, 1998.

Rubinstein, M. Markowitz's" portfolio selection": A fifty-year retrospective. *The Journal of finance*, 57(3):1041–1045, 2002.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. Trust region policy optimization. In *Icml*, volume 37, pp. 1889–1897, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017.

Shewchuk, J. R. et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

Teixeira, L. A. and De Oliveira, A. L. I. A method for automatic stock trading combining technical analysis and nearest neighbor classification. *Expert systems with applications*, 37(10):6885–6890, 2010.

Torres, R. *Alpha Vantage Stock API*. URL `https://github.com/RomelTorres/alpha_vantage`.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J., and Tsing, R. Starcraft ii: A new challenge for reinforcement learning, 2017.

White, D. J. *Dynamic programming*, volume 1. Oliver & Boyd Edinburgh, 1969.

Xiong, Z., Liu, X.-Y., Zhong, S., Yang, H., and Walid, A. Practical deep reinforcement learning approach for stock trading, 2018.