Risk-Adjusted Reward Functions for Reinforcement Learning Stock Trading

G006 (s2123972, s1443481, s1714840)

Abstract

Stock market trading is a natural application for Deep Reinforcement Learning, but it remains a relatively unexplored field. If agents are rewarded by non-risk adjusted returns, they may take risky trades in pursuit of maximising profit. There has been a lack of application of risk-adjusted reward measures from finance applied to Deep RL, the Differential Sharpe Ratio (DSR) being the only exception.

Using sum of total returns and DSR as baselines, we propose novel risk-adjusted reward measures inspired by Modern Portfolio Theory - Variance, Drawdown and CVaR. Contrary to the Sharpe Ratio, these risk measures are not the same for long and short trades. Markets are more volatile in the downward direction, hence such risk measures may perform better.

We apply these measures to two trading agents using Deep Reinforcement Learning - one a value-based agent using Deep Q-Networks, and the other a policy-based agent, using Deep Direct Reinforcement Learning. We test with Dow Jones stocks in 1 minute intervals from Feb 2020 to Feb 2021. We find the risk-adjusted measures perform better than non risk-adjusted rewards for the value-based, but not in policy-based agents. This is attributed to risk being calculable using past data.

1. Introduction

The stock market is an exchange where investors (or traders) buy and sell stocks. Traders can take long positions (buying), which is a positive position on the stock or short positions (short selling), which is a negative position on a stock. If the stock price P moves by ΔP, then the trader makes ΔP for long positions and $-\Delta P$ for short.

Recent advances have motivated researchers to apply AI techniques to stock market trading, and Reinforcement Learning (RL) methods are a natural fit for this purpose. In RL, an agent is dispatched into some environment and takes actions. Based on the actions, the agent receives some reward from the environment. Through appropriate algorithms, we can help the agent learn good strategies to maximise the reward. Researchers have successfully built trading agents that can outperform the market (Yang et al., 2020; Zhang et al., 2019). A common approach is to build agents that maximise the returns of the stocks.

The reward the agent receives affects the agent's decisions. If we reward the agent based on the returns of its trades, the agent will learn to maximise returns even if that comes at the expense of taking risky trades. To this end, we adjust the reward of our agent based on the risk the agent took. We propose three novel rewards for the trading agents. The rewards discussed in this paper are inspired by risk measures used in Modern Portfolio Theory. To the best of our knowledge this approach is novel.

Another novel aspect of our work is that the risk of a trade is calculated using data from after the trade. The agent makes a decision at time t based on data up to time t. However, we estimate the risk the agent took using data after t. We believe this is more appropriate, since the agent should not have access to the data used to calculate its reward.

In RL stock market trading literature, there exists a mix of both value- (Lee et al., 2020; Patel, 2018; Park et al., 2020) and policy- (Yang et al., 2020; Deng et al., 2016; Aboussalah & Lee, 2020) based RL algorithms. We examine if the risk-adjusted rewards improve performance of either of the two techniques. We train one agent using Q-Learning (Mnih et al., 2013) and the other using Deep Direct Reinforcement Learning (Deng et al., 2017), which is the stock market version of the REINFORCE algorithm (Williams, 1992) with an RNN. DDRL expands on the (Moody et al., 1998)'s work (DRL) by adding a fuzzy layer (Ching Teng Lin, 1991) and a deep network.

We examine the risk-adjusted rewards for both approaches used in literature. Our objective is to find if our risk measures are able to outperform non risk-adjusted rewards or other risk-adjusted rewards used in literature.

2. Related work

Reinforcement Learning has been prevalent in automated trading for at least two decades. Currently, Deep Q-Networks/Q-Learning, Policy-Gradient methods, and even Actor-Critic (Yang et al., 2020) algorithms can all be found implemented across AI trading algorithms. However, some of the first (and possibly pioneering) reinforcement trading algorithms, proposed by Moody *et al.*, (Moody & Saffell, 1998a; 2001; 1998b) have been direct, recurrent Policy-Gradient methods. Moody *et al.* use the terms "direct reinforcement" and "recurrent reinforcement" - these can be referred to collectively as DRRL (Direct Recurrent Reinforcement Learning). Direct means the methodology is centered on finding a policy function directly from financial

data (such as returns); recurrent means DRRL adapts over time as parameters are updated on incoming inputs and the feedback (reward) from its actions given this data. In our methodology section, we detail a development of DRRL for trading which we refer to as DDRL (Deep Direct Reinforcement Learning) or Policy-Gradient algorithm as in (Deng et al., 2017).

A novel advancement of Moody *et al.*'s work is the use of common risk measures such as the Sharpe Ratio and Drawdown risk measures for online trading. Portfolio managers and traders often maximise risk measures rather than a sum of returns (Moody & Saffell, 2001). For this end, they expand and approximate the Sharpe Ratio, deducing a recursive equation for the Differential Sharpe Ratio.

We define R_t to be the return at time t. Then, for some update constant η, define

$$A_t = A_{t-1} + \eta(R_t - A_{t-1})$$
$$B_t = B_{t-1} + \eta(R_t^2 - B_{t-1})$$

The Differential Sharpe Ratio at time t, which may be used as a reward, is

$$D_t = \frac{B_{t-1}\Delta A_t - 0.5A_{t-1}\Delta B_t}{(B_{t-1} - A_{t-1}^2)^{\frac{3}{2}}}$$

Where $\Delta A_t = R_t - A_{t-1}$ and $\Delta B_t = R_t^2 - B_{t-1}$

The approach is simply to change the utility function in order to maximise performance evaluated by a risk measure. The differential form of the risk measure considers the influence on the risk measure; therefore it is especially convenient for live trading, which only needs to update recursively from the previous step and has no need to keep track of financial history.

Inspired by the Sterling Ratio, they propose another novel approach - to maximise returns without penalising positive returns (Moody & Saffell, 2001; 1998b). Concerning Drawdown risk measures, Moody *et al.* (Moody & Saffell, 2001) (Section II E) note the standard deviation of returns is symmetrical and does not differentiate between bullish and bearish movements - this would cause the agent to penalise upward and downward movements. The Downside Deviation Ratio measures the proportion of negative returns. This is more meaningful; a highly positive average return compared to negative returns better outlines the performance. The differential form of the Downside Deviation Ratio helps the agent to learn a policy function that receives better returns from trades with less downside. As far as we know, the Differential Downside Deviation utility function has not been used in Deep Reinforcement Learning.

3. Data set and task

3.1. Data

We use stock price data over one minute intervals from 6 February 2020 to 25 February 2021 downloaded from Google Finance API. This API was selected as it provides reliable data free of charge. We use the data of 29 stocks in the Dow Jones Industrial Average included in the Appendix. We omit data from outside market hours, giving us prices from 09:30 to 15:59 on working days. The training set is 6 Feb - 15 Sept 2020, validation set from 16 Sept - 15 Dec 2020, while the rest is test set.

For each minute t and each stock, the dataset contains four values: **Open** is the starting price of the stock at t. **Close** is the price right before we transition to $t + 1$. **High** is the highest price in $[t, t + 1)$ and **Low** is the lowest price in $[t, t + 1)$. We have adjusted the data for stock splits.

Figure 1. Example from the dataset - closing prices of KO in over 1 minute intervals

As stocks come at different price ranges, we transform the data and use logarithmic returns, so the network can generalise easier. If c_t is the closing price at time t and o_t, l_t, and h_t are open, low and high prices at t respectively, we instead use $log(c_t/c_{t-1})$, $log(o_t/c_{t-1})$, $log(l_t/c_{t-1})$, $log(h_t/c_{t-1})$. We also apply fuzzy layers. Given some data points x_i, let μ be the mean and σ the standard deviation. The data is then transformed as,

$$x \leftarrow e^{-\frac{x-\mu}{\sigma}}$$

In Table 1 we present some statistics of our dataset. The close prices have a slightly positive mean. More importantly, we observe the absolute value of the average log low is higher than the absolute value of the average high. This means that stocks fall more dramatically compared to when they move up. This movement also happens with more variance, which means more volatility. We also give a sample distribution of the log closing returns of Coca-Cola (KO) in Figure 2.

A time series statistical analysis shows that the data has the

DATA	MEAN	VARIANCE
LOG CLOSE	0.003	0.137
LOG OPEN	0.000	0.005
LOG HIGH	0.174	0.078
LOG LOW	-0.179	0.099

Table 1. Mean and Variance of Log Close, Log Open, Log High and Low Low (scaled by 100) averaged across all stocks

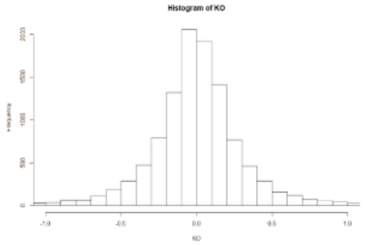

Figure 2. Histogram of Logarithmic Returns (scaled by 100) of KO (Coca-Cola)

same ACF[1] as that of white noise (Figure 3). This means that the returns are not correlated. This is consistent with the Efficient Marekt Hypothesis (Malkiel, 1989).

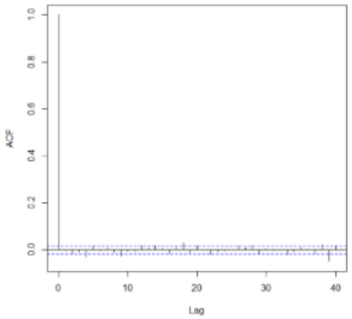

Figure 3. ACF plot of KO data set

3.2. Task

The agent is only allowed to trade every 10 minutes, so if the agent made a trade at minute t, it would be allowed to trade again at time $t+10$. We use $t+1, t+2, \ldots, t+9$ to approximate the risk the agent took. Let $\Re(t)$ be the risk of some trade at time t. Let c_t be the closing price of a stock at minute t, and define similarly for open (o_t), low (l_t), high (h_t) and returns (r_t). We propose the following ways to approximate $\Re(t)$, inspired by standard ways of measuring risk in finance.

(1) ***Variance***: Markowitz proposed to measure risk using variance (Markowitz, 1959). This measure is widely used today. We estimate the risk as $\Re(t) = 1 - var(r_{t+1}, ..., r_{t+9})$.

(2) ***Drawdown***: This is the worst return during a trade. If the agent took a long trade, then the agent suffered the worst return at its lowest point. Similarly, if the agent took a short trade, the agent suffered its worst loss at the highest point. So, for long trades,

$$\Re^{+}(t) = \frac{min(l_{t+1}, ..., l_{t+9})}{c_t}$$

and, for short trades,

$$\Re^{-}(t) = \frac{c_t}{max(h_{t+1}, ..., h_{t+9})}.$$

(3) ***CVaR***: Conditional Value at Risk (Acerbi & Tasche, 2002) is considered the state-of-the-art risk measure as it satisfies the mathematical properties of coherence. It is the average value of the $\delta\%$ worst cases. We have no access to tick data, so we instead approximate this measure using the following equations: for long trades, let z_t be the average of $(min(c_i, o_i) + 2 * l_i)/3$ for i in $t+1, ..., t+9$. Then,

$$\Re^{+}(t) = min(\frac{z_t}{c_t}, 1)$$

For short trades, we define z_t as the average of $(max(c_i, o_i) + 2 * h_i)/3$ for i in $t+1, .., t+9$. Then,

$$\Re^{-}(t) = min(\frac{c_t}{z_t}, 1)$$

We have purposely, constructed the risk measures so that $\Re(t)$ is always between 0 and 1 [2]. In addition, a risk measure of 1 symbolizes a trade with the lowest possible risk, and a risk measure of 0 is a trade with the highest possible risk.

In practise, our risk measures end up in the range [0.998, 1]. For a better spread we raise them to the power of 100. We leave as future work examining the use of different powers for this purpose. For numerical stability and to avoid overfitting, we take the maximum of the result and 0.4.

If an agent makes a trade at time t with associated risk $\Re$ and earns a return R, we reward the agent by

$$R \times \Re^{|R|/R} + c$$

Regardless if $R > 0$ or $R < 0$, the agent will receive a reward less than R which depends on the risk the agent took at t for its trade. c is a constant that gives the reward a zero mean.

We evaluate the results with three metrics: (1) With Returns (2) With the Sharpe Ratio, which is a risk-adjusted evaluation metric. If agent has made n trades with returns r_i, then the Sharpe Ratio is

[1] Autocorrelation Function

[2] Theoretically, the variance could be outside that range, but in practice that does not happen.

$$\frac{sum(r_i)/n - r_f}{var(r_i)}$$

where r_f is the risk-free interest. We assume this is 0, since interest rates are 0 for the period of our data set. The Sharpe Ratio calculates risk using variance, which is symmetric for long and short trades. (3) We also want a metric to capture downside risk. If our agents buy and sell again in less than 3 hours we will use the the worst trade. Otherwise we will use Downside Deviation. We differentiate, because the worst trade is more appropriate for short-term trades while downside deviation is more appropriate for longer term trades.

4. Methodology

We define the following Markov Decision Process (MDP):

- **State Space**: The agent has access to the past 50 data points. This includes the log low, log high, log open and log close. Furthermore, the agent knows the previous action it took.
- **Actions**: The are three possible actions: short sell, neutral (hold), or long (buy) which we denote -1, 0, and 1 respectively. If the agent chooses action $i \in [-1, 0, 1]$, then i holds for the next 10 minute interval.
- **Reward**: The reward of an action depends on the risk measure we use. If the agent gets reward r, the environment has transaction cost h, the last action was i_{t-1}, and new action is i_t, then the final reward will be:

$$r + |i_{t-1} - i_t| * h$$

In training, the transaction cost is treated as a hyper-parameter. A lower transaction cost leads the agent to over-trade and hence overfit the data. Bigger transaction costs lead the agent to take only one action. At validation and test time we keep the transaction cost fixed at 0.1%.

4.1. DQN

We train a Deep Q-Learning agent, following the architecture of (Mnih et al., 2013). The Q-Learning agent is controlled by a neural net, known as Q-Network, that takes as inputs some features that describe the state of the environment (here, the stock prices) and outputs the expected reward for each action the agent may take. The agent does not differentiate between stocks as done in (Lee et al., 2020).

We use a target network, Q', that gets updated by a copy of the main Q-Network every C steps as in (Mnih et al., 2013). If $Q(x, a)$ is the output of action a for input x, and the agent receives reward r and ends up at state x' after action a, we update using

$$Q(x, a) \leftarrow Q(x, a) + \alpha \cdot (r + \gamma \cdot max\, Q'(x', a') - Q(x, a))$$

where $\gamma \in [0, 1]$ is the discount rate (a constant) and α is the learning rate. We also use a memory that stores the state, action, reward, next state transitions. We use the standard learning algorithm for Q-Learning again following (Mnih et al., 2013).

In order to train all our stocks with the same agent, we follow the below routine,

Algorithm 1 Training one agent with multiple stocks

Initialize *list* (FIFO list of length 15)
repeat
  From *stocks* not in *list*, randomly select stock x
  Play stock x, while updating agent
  Put x in *list*
until stopping condition

4.2. Deep Direct Reinforcement Learning

We use the framework of (Deng et al., 2017) which, as far as we are aware, is one of the first algorithms to incorporate deep neural networks with a recurrent Policy-Gradient algorithm such as the ones proposed by Moody *et al.* (Moody & Saffell, 1998a; 2001). Deng *et al.*. train only on one asset at a time, so we only train on one stock (Apple - AAPL). In essence the algorithm adopts the same approach to these works by maximising the rewards over all time steps which ultimately learns the policy directly. However, Deng *et al.* (Deng et al., 2017) propose to further extract financial information by recurrently passing the data through a deep neural network. These inputs to the deep neural network is a novel element from Deng. They use a single fuzzy layer which, during training, learns to group the input data into the representations of interest (buy, hold and sell).

Below, we define our features to be represented from the past 50 close prices:

$$\mathbf{f_t} = [z_{t-m+1}, ..., z_t]$$
$$\mathbf{F_t} = g_d(\mathbf{f_t})$$

where $\mathbf{f_t}$ is the feature of vectors obtained from finding the differences between the close prices, and $\mathbf{F_t}$ is the vector of features transformed through the fuzzy layer and the deep neural network. The number of ticks in the window (50) is m.

Next, we define the policy function to be learned:

$$\delta_t = tanh(< \mathbf{F_t}, \mathbf{w} > + b + u * \delta_{t-1})$$

Here, w are the weights of the deep neural network (including the weights of the fuzzy layer), b is our bias term as the output is a linear regression, and $u * \delta_{t-1}$ is used to penalise a different trading decision to the trading decision one time step ago. This helps prevent too much trading, which adds up costs.

After applying this, we get a number $x \in (-1, 1)$. If

$x \in (-1, -1/3)$ the agent takes the short action, else if $x \in [-1/3, 1/3)$, the agent takes the neutral action, and otherwise it takes the long action.

We next define the returns, and hence the utility function the needs to be maximised:

$$R_t = \delta_{t-1} z_t - c \,|\, \delta_t - \delta_{t-1}|$$
$$max\ U_T[R_1, R_2, ...R_T|\Theta]\ w.r.t\ \Theta$$

Here, R_t are the returns up to time t, which evaluates the rewards from the previous trade and the current trade, c is a term accounting for transaction costs. The utility function U_T is to be maximised for each reward up to a period T (here the periods are t=1, t=2, etc.) with respect to all the parameters of the system (Θ).

The fuzzy input layer is also an integral part of the overall system, and Deng *et al.* follow pioneering work to incorporate this into their system (Ching Teng Lin, 1991). The motivation is that the agent will learn to filter and collect the data appropriately into 3 membership groups - buy, hold and sell. The methodology followed to implement a fuzzy layer is given in (Deng et al., 2017) Section IV Part A. Before sending data into the autoencoder, we cluster a single batch of 50 returns with K = 3 using a K-Means clustering algorithm. Our skeleton code then batch normalises using each group's mean and standard deviation to yield a Gaussian transformation for each group. The three groups are then concatenated and forwarded into the autoencoder.

Algorithm 2 Deep Direct Reinforcement Agent

Calculate returns from historic close prices between each point.
repeat
  Initialise any arrays or parameters.
  repeat
    Forward pass batch through the deep recurrent neural network.
    Calculate the utility function and collect the current reward.
    Optimize the utility function and update all parameters Θ
  until All batches have been processed
  Sum the collected rewards
until 50 Epochs

During training, we may begin to encounter the vanishing gradient problem as the DDRL agent becomes deeper. The authors use task-aware back-propagation (see Figure 3 in (Deng et al., 2017)) which provides a residual link between the gradient information concerning the current task U_t and all previous actions $\delta_{t-1}, \delta_{t-2}$... which in themselves are neural networks. We found batch normalising between the auto encoder layers was simple to implement and maintained functionality.

5. Experiments

5.1. DQN experiments

We train our Deep Q agent with the following condition: if the agent chooses a random action at timestep t, the agent is forced to take the same action for the next x 10 minute intervals. Normally, the agent is allowed to trade every 10 minutes. We expected the agent would be able to find long term rewards more easily, and used this as a way of encouraging exploration. However, in practice, it ended up making the agent less flexible and hence the agent was not able to get better results in the training set (Figure 4).

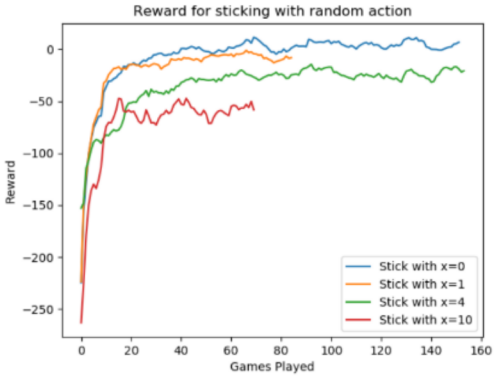

Figure 4. Agent is less flexible during training. Training set results get worse as we restrict the agent more

The agent learns to trade, but 83% of the time it exits the trade as soon as possible, meaning after 10 minutes. This is not necessary bad, however (1) traders may be uncomfortable with such short-term trades and (2) a supervised approach may be more applicable for such predictions.

To encourage longer trades, we give the agent a small reward [3] for sticking with the previous action. Furthermore, we pass the state of the agent through a dense four layer network, and then concatenate the result with the one-hot encoded previous action and pass that through another dense two layers network, shown in figure 5. This helps the network to have more immediate access to the previous action.

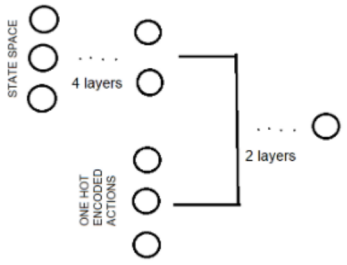

Figure 5. DQN Neural Network Architecture

We do hyper-parameter tuning in two steps. We treat learning rate, the rate C of how often we update the target net-

[3] Approximately 0.05 percent of the average absolute returns.

Risk Measure	Returns	Sharpe Ratio	Worst Trade
No Risk	-4.00	-0.14	-13.46
Diffe. Sharpe	-3.91	-0.15	-13.14
Variance	-5.12	-0.21	-11.13
Drawdown	**−0.84**	**−0.08**	**−3.14**
CVaR	-3.95	-0.14	-8.49

Table 2. Test set results for longer trades. Returns are averaged across all stocks. Sharpe Ratio and worst trade is across all trades of all stocks. Returns and Worst trade are in percentage.

work, and the exploration rate as a means of making the agent learn the training data. We do a grid search over weight decay and discount factor to find out which combination performs the best on the validation set. We use a batch size of 12 and memory size of 400K. For reference, each stock has around 6500 training steps.

To understand how the learning rate and C affect learning we run different combinations and present the results in Figure 6. During our experiments we found that an exploration rate of 3% worked consistently well across different hyper-parameters.

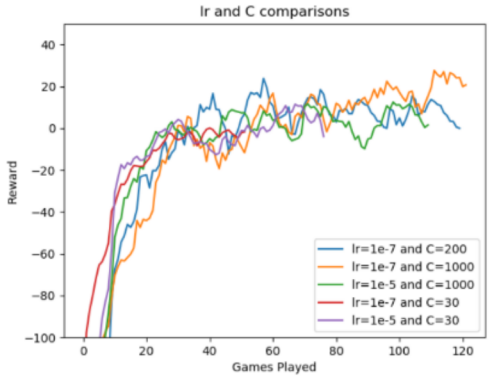

Figure 6. Comparison of different learning rates and C

Using a learning rate of $1e-7$ and $C = 1000$ works the best after 80 games are played. The explanation is that data is noisy. Hence, we need a small learning rate to take small steps. Also, we need a high C because frequent updates to the target network are not reliable due to the noise of the data. The target network needs to be updated less often to have a more reliable estimate.

Using our new understanding of the optimal learning rate and C we do a grid search over both discount factor and weight decay to optimise them. A good weight decay is around $1e-20$. For the discount factor ranges (between 0.7 and 0.95), we generally found that a small discount factor worked better, probably because the next estimate is not very reliable due to the noise of the data. We present the test scores of each risk measure in Table 2.

5.2. Policy Gradient Experiments

Our baseline experiment (using no risk measure) uses a typical utility function which usually defines the methodology for Policy-Gradient reinforcement trading e.g. (Moody & Saffell, 2001). This is simply the objective of maximising the sum of total returns for every trade.

We train the algorithm in batches of 5 windows from the start of the stock prices (at t = 0). For example, in the first 5 batches, the first batch contains the returns from t-start=0 to t-finish=49, the next batch will contain returns from t-start=1 to t-finish=50, etc., and the final window in the last batch will have returns from t-start=T-100 to t-finish= T - 50, where T is the index of the last closing price.

For all experiments, we incorporate Differential Sharpe Ratio as outlined in Section 2 for the DDRL agent and CVaR (Section 3). The other risk-adustments could not be easily implemented. We used a learning rate of 0.001 using a standard gradient descent optimiser from TensorFlow to learn the deep network parameters during the trading windows. (Deng et al., 2017) use a decaying learning rate of 0.97, but we found we had best performance on the baseline validation set by using 0.001 as learning rate; smaller values hampered the total returns and larger values do not maximise performance. We keep our autoencoder consistent with Deng *et al.*, in which they use 3 hidden layers of 128 units and an output layer of 20 units as their default setting. Concerning the fuzzy layer, which takes K clustered groups of the input batch by K-Means clustering, has K set to 3 for all experiments (as in (Deng et al., 2017)). Intuitively, we need to group the data into 3 membership functions (upwards, stagnant and downwards), which the agent should learn during training. Deng *et al.* train for 100 epochs over the returns, however, we encountered some difficulty since the RL agent became unreliable and would sometimes stop learning on both training and validation sets. Since Deng *et al.* implement "Task Aware Back Propagation" to tackle the gradient vanishing problem, we also have reason to believe that the agent stops training for the same reason. We reduce to 50 epochs which produces consistent results with no sign of learning problems, we also observe convergence over the epochs (see Figure). Additionally, we use leaky ReLus in our autoencoder as well as batch normalisation.

In this context, overfitting means to train the policy function δ so that it poorly generalises to the next or future windows of price returns. We do not have reason to believe that this is a significant problem in our experiments since Deng *et al.* mention that training for only 100 epochs is not a cause for overfitting concern. We train for 50 epochs. Furthermore, we have treated our experiments in an ex ante manner, which means that predictions are continuously unsupervised and the agent is exposed to unseen data before each trade. However, we take an approach to make our algorithm generalize through a validation set through which we found the above hyper-parameters. We simply split our data into training, validation and testing and discard the first 100 points of the latter two sets. This allows a large

Risk Measure	Final Returns	Sharpe Ratio	Downside Dev.
No Risk	**28.91**	**0.12**	**0.23**
Diffe. Sharpe	-11.52	-0.043	-0.06
CVaR	23.39	0.089	0.15

Table 3. Experiment test results for Policy-Gradient- CVaR

enough gap for the Apple stock conditions to change. 60% of the data was used for training. 15% of the data gave us a chance to optimise any hyper-parameters and we use 25% of the data as a test set. We wanted to maximise our use of the test set to observe performance; we found that Moody *et al.* (Moody & Saffell, 2001; 1998a) test their agent over a period of 25 years.

The best results for the Policy-Gradient algorithm are produced by the non risk-adjusted reward. Between total sum and CVaR experiments, the risk evaluations are comparable though neither are spectacular. Both make profit but have poor risk-adjusted returns despite minimizing risk. The worst result came from using Drawdown reward. We could not produce sensible results even on the training set, so opted to not pursue this experiment further. Implementing the Differential Sharpe Ratio did not produce higher returns on the training set; in fact we observed a loss in profit on training and test sets. We were forced to change the hyper-parameters for this experiment. The hyper-parameters for the other experiments gave us consistent results and stable behaviour but in the case of using a Differential Sharpe Ratio, we had to change the learning rate to 0.01. A learning rate of 0.1 or 0.001 created instability in the agent and to produce good results at least on the training set required 100 epochs. Our value for η was set to $8e-4$ which produced consistent ,stable results. The second best result came from using CVaR risk-adjusted returns on learning rate 0.001 and 50 epochs.

We summarise our test results in Table 3 having evaluated the returns using the Sharpe Ratio and Downside Deviation (Moody & Saffell, 2001) (Section II E, Equation 20). We display CVaR test returns in Figure 7 for what we believe to be a more noteworthy result; as far as we are aware using CVaR is a novel application of maximising risk-adjusted return for a Policy-Gradient algorithm.

The Downside Deviation Ratio is given as:

$$\frac{Average(R_t)}{(\frac{1}{T}\sum_{t=1}^{T} min(R_t,0)^2)^{\frac{1}{2}}}$$

for which we reviewed its differential form in Section 2. This is the average returns over average negative returns. If this value is high, then it means that the agent trades well even when accounting its negative returns.

6. Discussion

For the DQN experiment, we observe that Drawdown is able to outperform, across all metrics, all the other risk

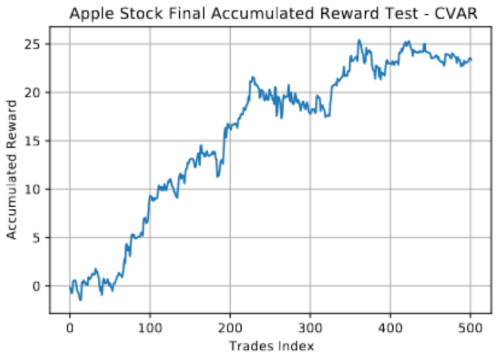

Figure 7. CVaR test set, profit accumulation over trades

measures. Here we explain why this happens. First note that Drawdown is the most extreme version of risk measure. Drawdown uses as its risk measure the worst possible case, whereas Variance and Differential Sharpe Ratio just use the variance, and CVaR calculates the average of the $\delta\%$ worst cases. In finance, short trades are known to be more volatile than long trades. Therefore Drawdown would be trading long more often than short when compared to the other measures.

Now consider the timeline of our dataset shown in Figure 8. During the test set, the markets were going up. Hence, strategies that favored long positions had an advantage. Meanwhile, in our training set, there was a big dip in Feb - March 2020. Hence, agents that did not punish risky trades as hard, were tempted to short sell.

Figure 8. Timeline of our dataset. Shows the market index of S&P 500.

In short, the Drawdown risk measure punishes risky trades the most. Hence, it was more cautious with short trades and as a result took more long trades, which benefited it in the test set. As a further investigation we examine the results for short trades and long trades independently in Table 4. We see Drawdown and CVaR outperform the No Risk agent. An interesting idea for further study would be to use CVaR for long trades and Drawdown for short trades.

Our risk measures were not able to outperform the baseline in Direct Reinforcement Learning. Predicting the value of the risk-adjusted reward (Q-Learning) seems from the experiments to be more appropriate than trying to derive

Risk Measure	Short Return	Long Return
No Risk	-6.86	-1.13
Differential Sharpe	-5.87	-1.21
Variance	-7.21	-1.81
Drawdown	×	-0.84
CVaR	-4.40	**−0.53**

Table 4. Returns of short and long trades averaged across all stocks in percentage

the policy that maximises the risk-adjusted reward (DRL). Training across multiple stocks for DRL is required to confirm this, however we give the following explanation. Calculating the actual risk is more natural than trying to follow the least risk. It may be hard for the agent to understand when risk will happen given past data. However, the agent may be able to calculate the current risk easier. This would be consistent with literature, for example GARCH models (Engle, 1982) which are widely used. The philosophy of GARCH models is that you can estimate the future risk (variance) of the stock based on past data, using time series models. Similarly, the Q-Network is able to calculate the current risk, unlike the policy agent.

Our Q-Learning agent also fails to make money. Q-Learning for stock market trading has faced criticism from researchers (Deng et al., 2017; Moody & Saffell, 2001), despite the success mentioned in the Introduction. Critics note that predicting exact value in the stock market is impossible due to the volatile nature of markets. Instead, learning actions directly is more appropriate. If a stock falls it may be hard to calculate the exact reward the agent will receive, however it may be significantly easier to learn that the agent should short the stock or go neutral.

When simply maximising profit the Policy-Gradient algorithm outperforms Deep Q-Learning but it failed to penalise risky rewards especially when using the Differential Sharpe Ratio. Using past values of CVaR at least produced profit on the test set, but no more than using no risk-adjustment. It is worth mentioning that the DDRL (Deep Direct Reinforcement Learning) algorithm has to use the output of the previous policy function (to calculate returns) which in itself is a deep neural network. Over time, this enlarges the neural network since the returns are calculated recursively. This could require more than just batch normalisation to produce better results. Deng *et al.* use Task Aware Back Propagation (see figure 3 in (Deng et al., 2017)) to deal with this problem.

We also found running the Policy-Gradient algorithm was sometimes difficult to keep stable; we blame the vanishing gradient problem since we occasionally observed fully stagnant returns. When dealing with the Differential Sharpe Ratio, it does require initialisation due to its recursive nature; we set A_{t-1} and B_{t-1} to values comparable to a typical return and also make sure the first calculations do not lead to indeterminate values. However, we think a better approach to create a stable initialisation would be switching to the Differential Sharpe Ratio once the agent is warmed up on the sum of returns utility function, at least this way initialisation will be more accurate. We evaluate the results using the Sharpe Ratio and the Downside Deviation Ratio (DDR) and find that typically DDR is sometimes twice as much relative to the Sharpe Ratio. All methods of evaluation show possible overfitting since the test results across experiments show worse performance and generalisation capability. A good value for the Sharpe Ratio is considered to be at least 0.7. Our test set, however, is shorter than the training set, so it is worth considering that the agent will take more time to adjust to the new market conditions. The Apple stock smoothly increases during training but becomes more volatile during the test set. Still, we at least observe profit and provide evidence of adjusting capability for the sum of returns utility function and CVaR risk-adjusted returns.

7. Conclusions

The risk-adjusted reward of CVaR and Drawdown convincingly outperform the non risk-adjusted reward for Q-Learning, for both short and long trades. Calculating the exact value of future returns is not an easy task, but calculating the exact risk is possible. For this reason, the agents perform better when they maximise risk-adjusted reward, instead of maximizing the returns.

Volatile trades that made high returns in the past are not likely to repeat in the future. Hence when our agent adopts a more risk-aware approach, it is able to ignore such trades and choose the safer trades that are more likely to repeat in the future. As a result, the risk-aware Q-Network performs better for both short and long trades.

The opposite happens in Policy-Based. The agent finds a policy that is profitable using both the non-risk and risk-adjusted reward (except DSR), but the policy that maximises a risk measure does not necessarily lead to better results. Perhaps not surprisingly the policy that gives the best returns is that which maximises returns. There are signs of overfitting since the returns are far less than returns when training.

One potential problem with our proposed risk measure is that it adds a lot of noise to the dataset. As future work, we suggest clipping the risk measure in three different categories, 'Low', 'Medium' and 'High' Risk. The new associated risk of our agent $\Re$ will be for example $\Re_{med}$. We also suggest experimenting with longer-term trades - these are more volatile so our risk measures will be more informative; for example, Moody *et al.* test over 25 year periods.

We tried implementing the Differential Downside Deviation utility function as implemented in (Moody & Saffell, 2001) (Section II E), but faced severe overfitting problems. As future work, we propose using dropout layers, and Task Aware Backpropagation (Deng et al., 2017) with batch normalisation.

Thanks to AshengLin from where we found Policy-Gradient skeleton code to work from on GitHub.

References

Aboussalah, Amine Mohamed and Lee, Chi-Guhn. Continuous control with stacked deep dynamic recurrent reinforcement learning for portfolio optimization. *Expert Systems with Applications*, 140:112891, 2020.

Acerbi, Carlo and Tasche, Dirk. On the coherence of expected shortfall. *Journal of Banking & Finance*, 26(7): 1487–1503, 2002.

Ching Teng Lin, C.S. George Lee. Neural network based fuzzy logic control and decision system. *IEEE Transactions on Computers*, 40(12):1320–1336, 1991.

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, 2017. doi: 10.1109/TNNLS.2016.2522401.

Deng, Yue, Bao, Feng, Kong, Youyong, Ren, Zhiquan, and Dai, Qionghai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3):653–664, 2016.

Engle, Robert F. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica: Journal of the econometric society*, pp. 987–1007, 1982.

Lee, Jinho, Kim, Raehyun, Yi, Seok-Won, and Kang, Jaewoo. Maps: Multi-agent reinforcement learning-based portfolio management system. *arXiv preprint arXiv:2007.05402*, 2020.

Malkiel, Burton G. Efficient market hypothesis. In *Finance*, pp. 127–134. Springer, 1989.

Markowitz, Harry. Portfolio selection, 1959.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Moody, John and Saffell, Matthew. Reinforcement learning for trading. NIPS'98, pp. 917–923, Cambridge, MA, USA, 1998a. MIT Press.

Moody, John, Wu, Lizhong, Liao, Yuansong, and Saffell, Matthew. Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting*, 17(5-6):441–470, 1998.

Moody, John E. and Saffell, Matthew. Reinforcement learning for trading systems and portfolios. In Agrawal, Rakesh, Stolorz, Paul E., and Piatetsky-Shapiro, Gregory (eds.), *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), New York City, New York, USA, August 27-31, 1998*, pp. 279–283. AAAI Press, 1998b. URL http://www.aaai.org/Library/KDD/1998/kdd98-049.php.

Moody, John E. and Saffell, Matthew. Learning to trade via direct reinforcement. *IEEE Trans. Neural Networks*, 12(4):875–889, 2001. doi: 10.1109/72.935097. URL https://doi.org/10.1109/72.935097.

Park, Hyungjun, Sim, Min Kyu, and Choi, Dong Gu. An intelligent financial portfolio trading strategy using deep q-learning. *Expert Systems with Applications*, 158:113573, 2020.

Patel, Yagna. Optimizing market making using multi-agent reinforcement learning. *arXiv preprint arXiv:1812.10252*, 2018.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Yang, Hongyang, Liu, Xiao-Yang, Zhong, Shan, and Walid, Anwar. Deep reinforcement learning for automated stock trading: An ensemble strategy. 2020.

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

Appendices

The stocks in the Dow Jones at the time of writing,

Stocks
Apple
Amgen
American Express
British Airways
Caterpillar
Salesforce
Cisco
Chevron
Walt Disney
Goldman Sachs
Home Depot
Honeywell
IBM
Intel
Johnson & Johnson
JPMorgan Chase
Coca-Cola
McDonald's
3M
Merck
Microsoft
Nike
P&G
Travelers
UnitedHealth
Visa
Verizon
Walgreens
Walmart

Figure 9. Timeline of Apple stock and buy/sell marks by the agent training set for DRL. Marks have been stripped for clarity

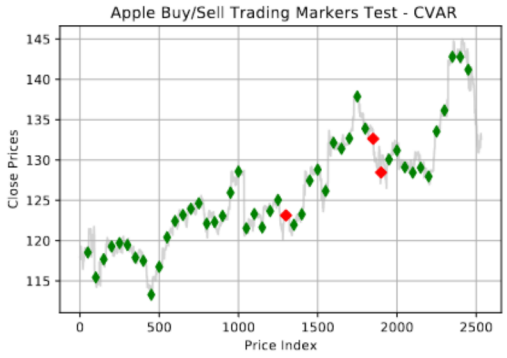

Figure 10. Timeline of Apple stock and buy/sell marks by the agent on test set for DRL. Marks have been stripped for clarity

Risk Measure	Final Returns	Sharpe Ratio	Downside Dev.
No Risk	**215.45**	**0.16**	**0.39**
Diffe. Sharpe	27.05	0.023	0.035
Variance	×	×	×
Drawdown	-268.88	-0.16	-0.19
CVaR	150.69	0.11	0.23

Table 5. Experiment Training Results for Policy - Gradient (DDRL)

Risk Measure	Final Returns	Sharpe Ratio	Downside Dev.
No Risk	**11.19**	**0.053**	**0.073**
Diffe. Sharpe	-1.043	-0.0049	0.062
CVaR	6.56	0.030	0.03818

Table 6. Experiment Validation Results For Policy - Gradient (DDRL)

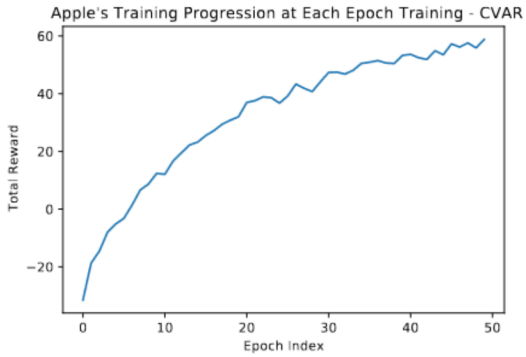

Figure 11. Total Reward convergence over epochs (DRL)