A Machine Learning Approach for Asset Allocation: Put-write and Trend Following

by

Bo Na

A research paper
presented to the University of Waterloo
in fulfillment of the
requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Supervisors: Prof. Peter Forsyth & Prof. Yuying Li

Waterloo, Ontario, Canada, 2019

I hereby declare that I am the sole author of this research paper. This is a true copy of the research paper, including any required final revisions, as accepted by my examiners.

I understand that my research paper may be made electronically available to the public.

Abstract

Put-write strategy and trend following strategy are both popular investment strategies in finance industry. In this research paper, we investigate an asset allocation problem in a portfolio comprised of the put-write and trend following indices. The ultimate goal on this combination of assets is to reduce the loss side risk of the put-write strategy, while generating a similar level of return. We define the loss side risk as the risk of generating negative returns and we look at the entire loss side of the distribution of return, instead of just focusing on the left tail.

We formulate the multi-period asset allocation strategy as a stochastic optimization problem. We use the target based objective function and the neural network framework introduced in [31, 28]. The one-sided quadratic function is chosen to penalize the expected shortfall, which focuses on minimizing the loss side risk. The neural network optimization satisfies constraints of no shorting and no leverage automatically, converting the constrained optimization problem to an unconstrained one. We consider three optimal control strategies based on optimizing the objective function: (i) an optimal constant weight control, (ii) an optimal deterministic control which is a function of the time only, and (iii) an optimal stochastic control which is a function of the time and the current wealth. The benchmark portfolio is a constant weight strategy that is based on optimizing the Sharpe ratio. We constrain the median of our three strategies to be the same as the benchmark portfolio. In general, the optimal constant and optimal deterministic controls are very similar to the benchmark strategy, and the optimal stochastic control outperforms the benchmark strategy by obtaining a smaller loss side risk.

Acknowledgements

I would like to express my gratitude to my supervisors, Prof. Yuying Li and Prof. Peter Forsyth, who have been supporting me with instructions and encouragement. I would like to thank Prof. Justin Wan for reading my paper. I also want to thank Greg Wang for editing this paper and Mingyu Yang for reading the paper and giving advice. I appreciate the efforts of Prof. Jeff Orchard, Prof. Henry Wolkowicz, Cherisse Mike, Amanda Guderian and the rest of Computational Mathematics staff in creating an incredible learning environment. I would also like to thank my wonderful colleagues for being supportive and growing together with me.

Dedication

This is dedicated to my family, for their support and encouragement during my master's degree.

Table of Contents

List of Tables viii

List of Figures ix

1 Introduction 1

2 Background 4

2.1 Put-write Strategy 4

2.1.1 Put Option 4

2.1.2 Collateralized Put-write Strategy 5

2.2 Trend Following Strategy 6

2.3 Problem Description 8

3 Formulation for Dynamic Asset Allocation 9

3.1 Multi-period Asset Allocation 9

3.2 Objective Function 10

3.3 Three Optimal Controls and the Benchmark 11

3.4 Represent Controls by a Neural Network 12

4 Optimization Technique 15

4.1 Trust Region Method 16

4.2 Gradient Descent 18

4.3 Mini-batch Gradient Descent 20
4.4 ADAM 23

5 Data **27**

6 Bootstrap Resampling **31**

7 Optimal Allocation Analysis **36**
7.1 Training Results 37
7.2 Testing on Bootstrap Resample Data 42
7.3 Testing on the Historical Data 45

8 Conclusion **48**

References **50**

List of Tables

5.1 Statistics for annualized log returns and maximum drawdown 29
5.2 Skewness and kurtosis of daily log returns 29

6.1 Optimal expected blocksize 32
6.2 Skewness of bootstrap resample data sets 34
6.3 Kurtosis of bootstrap resample data sets 34
6.4 95%-CVaR of bootstrap resample data sets 35

7.1 Training Results: Expected Blocksize = 5.8 days 37
7.2 Testing Results on bootstrap resample data, expected blocksize = 20 days 43
7.3 Testing Results on bootstrap resample data, expected blocksize = 10 days 44
7.4 Testing Results on bootstrap resample data, expected blocksize = 1 day . . 44
7.5 Testing results on the actual historical data paths (Annualized) 45

List of Figures

2.1 Payoff and profit of put options 5

3.1 A NN representing the stochastic control function 13

4.1 Trust region method: objective function value at each iteration (10,000 sample paths); computation time is 19.5 mins; terminal objective value is 0.000748 18

4.2 Gradient descent method: objective function value at each iteration 20

4.3 Mini-batch gradient descent method: objective function value at each iteration, $B = 10000$ 22

4.4 Mini-batch gradient descent method: objective function value at each iteration, $\eta = 100$ 22

4.5 ADAM method: objective function value at each iteration 25

4.6 ADAM method: objective function value at each iteration, $\eta = 0.001$, computation time is 45.5 mins, terminal objective function value is 0.000746 . . 26

5.1 Historical Performance of the PUT Index and SG CTA Index 28

5.2 Cumulative distribution function (cdf) comparison of daily log returns . . . 30

6.1 Probability density (pdf) comparison of three-month log return 33

6.2 Probability density comparison of three-month log returns (loss side) . . . 34

7.1 Deterministic control 38

7.2 Training: density histogram 39

7.3 Training: cdf comparison 40

7.4 Stochastic control heatmap 41
7.5 Stochastic control: distribution of allocation 41
7.6 Testing Results on bootstrap resample data, expected blocksize = 20 days 43
7.7 Testing Results on bootstrap resample data, expected blocksize = 1 day . . 44
7.8 Testing results on the actual historical data paths 46
7.9 Testing results on the actual historical data paths (2008) 47

Chapter 1

Introduction

In the investment world, one significant behavioral bias is risk aversion. Given investments with the same expected return, investors prefer strategies with less volatility. It is common to see professionals in portfolio allocation sacrifice some returns for more stable income streams and downside protection. In finance, hedging refers to the process of eliminating or mitigating the undesirable risk of adverse price movements in an asset. One of the most common hedging instruments to transfer downside risk is the put option. The fact that market participants naturally have long equity positions creates a high demand for put options to reduce the downside risk. This brings the put-write strategy to many investment professionals' attention.

Put-write is a strategy that writes put options and collects premiums. When writing put options, the writer accepts a future contingent liability that is less than or equal to the option's strike price. Meanwhile, the put option holder transfers the downside risk to the writer. Therefore, the put-write strategy is subject to huge losses during market downturns. It is of great importance to mitigate these losses in order to achieve superior performance in the long term. In this research paper, we seek for a solution to reduce the loss side risk of the put-write strategy. We define the loss side risk as the risk of generating negative returns and we look at the entire loss side of the distribution of return, instead of just focusing on the extreme left tail.

A traditional approach to mitigating risk is through delta hedging. However, while delta hedging removes the downside risk, it also eliminates the opportunities for gains. Alternatively, we look for an investment strategy or instrument to protect the put-write strategy when the market goes down, but does not eliminate the possibilities of gains when the market goes up. An investment strategy that generates positive returns in bear markets

would be optimal to hedge against the downside risk of the put-write strategy. Ideally, the strategy should be able to generate positive return as well when the market goes up. One strategy with such property would be a trend following strategy. A trend following strategy is based on the commonly held belief that prices of financial assets tend to follow their trends. By taking a long position in an up trend and a short position in a down trend, a trend following strategy is expected to generate possible returns as long as the market is showing a trend. Specifically, most bear markets historically occurred gradually over several months, rather than abruptly over a few days, which allows a trend following strategy to be in a short position after an initial drop and profit from sustained market declines. It has been shown in [25] that a time series momentum strategy, one of the most naive trend following strategies, experienced positive returns during eight out of ten of major US bear markets. Significant positive returns are observed during a number of these events. From historical evidence, we believe a trend following strategy can potentially protect the put-write strategy in a market downturn. Of course, such a strategy can never perfectly predict trends, so the best we can do is to reduce the losses of the put-write strategy in market downturns. The two investment strategies will be discussed in more detail in Chapter 2.

We consider an asset allocation problem in a portfolio comprised of the put-write and trend following indices. Asset allocation refers to the process where investors decide how their funds are allocated among different assets. The mean–variance criterion of Markowitz in [30] has been widely recognized as the foundation of the asset allocation problem. Under mean-variance analysis, investors attempt to achieve the greatest expected return at a given level of risk or the least risk at a given level of return. However, Markowitz's mean-variance framework is limited to the single-period asset allocation problem. Dynamic programming has been introduced in [4] for solving multi-period allocation problem by breaking it into single-period problems in a recursive manner. We note that solving an optimal control problem via a Hamilton Jacobi Bellman (HJB) Partial Differential Equation (PDE) has been applied to continuous time mean-variance asset allocation in [40, 41]. Numerical solutions of the HJB PDE which account for discrete rebalancing and practical assumptions, such as no shorting and no leverage, are developed in [15, 12]. However, these methods are only limited to a small (≤ 3) number of assets and we use a framework which in theory can accommodate asset allocation with multiple assets.

We use the target-based objective function introduced in [31]. The one-sided quadratic function is chosen to penalize the expected shortfall, which focuses on reducing the left tail. We remark that an alternative method of optimizing Conditional Value at Risk (CVaR) (see [14]) focuses on minimizing the risk in the left tail. In this research paper, we choose the one-sided quadratic function in order to generate a greater region of stochastic

dominance. The concept of stochastic dominance is explained in detail in Chapter 7. We consider three optimal control strategies based on optimizing the objective function: (i) an optimal constant weight control, (ii) an optimal deterministic control which is a function of the time only, and (iii) an optimal stochastic control which is a function of the time and the current wealth. The benchmark portfolio is a constant weight strategy that is based on optimizing the Sharpe ratio.

We use a data-driven machine learning optimization framework proposed in [28] for the asset allocation problem. We compute the optimal controls represented by a Neural Network(NN). The data-driven NN approach was originally applied to solve an asset allocation problem for defined contribution pension plans and has been shown to generate comparable results to a provably optimal strategy in [28]. Here, we extend the method to an investment portfolio consisting of the put-write and trend following indices. The neural network optimization satisfies the constraints of no shorting and no leverage automatically, converting the constrained optimization problem to an unconstrained one. We remark that machine learning approaches have been recently suggested for investment problems. In [7], NN approaches are used for global asset allocation. In [1, 29], machine learning approaches have been considered to predict asset returns.

We constrain the median of our three strategies to be the same as the benchmark portfolio. The three control strategies are directly learned from market return sample paths generated by bootstrap resampling. We show that the optimal constant weight and optimal deterministic controls are very similar to the benchmark strategy, and the stochastic control outperforms the benchmark strategy by obtaining a smaller loss side risk. To illustrate the robustness of our approach, we test the optimal controls on bootstrap resampling data sets with different blocksizes, as well as actual historical paths.

In this research paper, we also investigate the impact of the expected blocksize on the distribution of the overall return generated from bootstrap resampling. In addition, we discuss four optimization techniques, including (i) the trust region method, (ii) the gradient descent method, (iii) the mini-batch gradient descent method, and (iv) the adaptive moment estimate (ADAM) method. We compare the performance of these methods by considering the computation time, the terminal objective function value and the "smoothness" of the objective function value trajectory.

Chapter 2

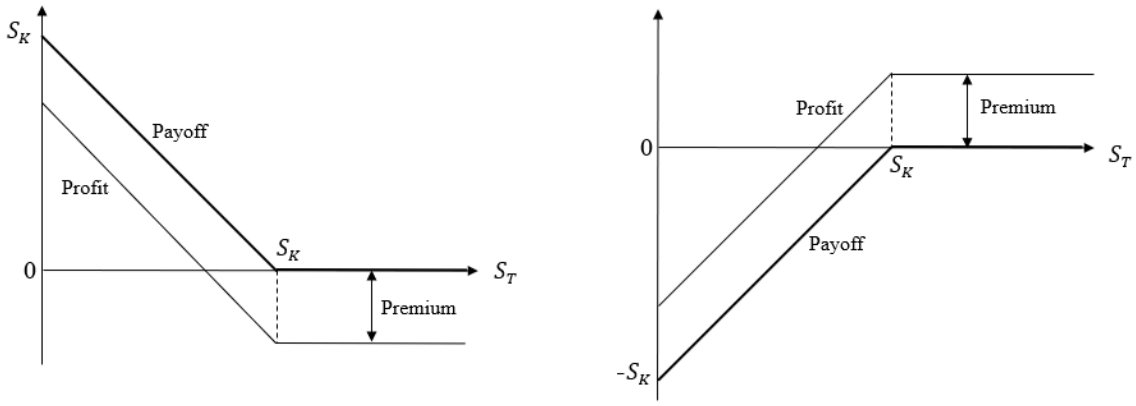
Background

2.1 Put-write Strategy

2.1.1 Put Option

A financial derivative is a contract whose value is derived from an underlying asset or a basket of assets. A put option is a common type of financial derivatives. A European put option provides the holder the right to sell an underlying asset at a strike price S_K at the maturity time T. If the price of the underlying asset is lower than the strike price S_K at time T, the put option holder can buy the asset from the market at a lower price and sell it at the strike price S_K. In this case, the put option expires "in-the-money" and the put option holder has a positive payoff. On the other hand, if the price of the underlying asset is higher than the strike price S_K at the maturity time T, a rational person will not exercise the option to sell the underlying asset at the lower strike price S_K. This is commonly referred to as expiring "out-of-the-money" and the payoff of the put option will be zero. The profit (or loss) for the option holder is just the payoff minus the premium of the put. Figure 2.1a illustrates the payoff and profit of a long put option. In industry, a put option is usually considered as an insurance to provide downside protection.

When financial institutions write put options, they take downside risk for the premiums. If a put option expires "out-of-the-money", the payoff will be zero. Then, put writers can walk away with the premiums as their profit. Statistical analysis has shown in [38] that put options on the S&P 500 Index expiring worthless outnumber those expiring "in-the-money". However, if the market crashes, put options will be deeply "in-the-money" and unhedged put writers will suffer huge losses from short puts. Therefore, the put-write strategy is



(a) Long put option

(b) Short put option

Figure 2.1: Payoff and profit of put options

profitable most of the time, but it has a large loss side risk due to the uncertainty of occasional market crashes. Figure 2.1b presents the payoff and profit of a short put option. An unhedged put-write position is also known as a "naked put-write".

2.1.2 Collateralized Put-write Strategy

Put writers are usually required to post collateral so that the collateral can cover the maximum potential loss from the short puts. Consequently, they typically create a collateral portfolio within the put-write strategy. When short-term U.S. Treasuries are held in the collateral portfolio, such as one-month and three-months Treasury Bills (T-bills), we call it a cash-collateralized put-write strategy. In general, there are three benefits of the cash collateral portfolio. In addition to creating an extra income stream from the short-term T-bills to enhance the long-term performance, the collateral portfolio has a smaller overall volatility. Previous studies (see [2, 39]) have shown that a cash collateralized put-write strategy on the S&P 500 Index is able to achieve smaller volatility and generate better risk-adjusted return, as measured by the Sharpe ratio. Another important advantage it gives over a "naked put-write" strategy is that it decreases the maximum drawdown. During periods of market stress, yields tend to decline, which leads to the increase in the value of the collateral portfolio. This mechanism can offset a portion of the losses from writing puts. For simplicity, we refer to the collateralized put-write strategy as the put-write strategy in this research paper.

In addition to the collateral income, the put-write strategy also generates return from

put option premiums. There are two principal risk components accounting for put option premiums, volatility risk premium and equity risk premium. The volatility risk premium is measured by the difference between implied volatility and realized volatility of the same underlying asset over time. Implied volatility is the volatility at which the market price of an option equals the price suggested by the Black-Scholes model. It is calculated using the market price together with the remaining parameters in the pricing model. Implied volatility is usually interpreted as the market's forecast of the volatility of the return of the underlying asset. Realized volatility, on the other hand, is the volatility based on the actual return of the underlying asset.

Intuitively, the volatility risk premium refers to the phenomenon that investors tend to over-estimate volatility. It has been shown in [26] that out-of-the-money put options are systematically overpriced. It is also found in [39] that the put-write strategy outperforms the covered-call-write strategy over the period 1986 - 2008. These findings are explained by the higher volatility risk premium in the left tail in [37] and high demand for downside protection in [17]. Such occurrence originates from investors' behavioral bias towards risk aversion and investors' bias towards overestimating downside risk, as described in [19].

The other important risk component is the equity risk premium, which comes from the long exposure to the market in a short put option. Market risk can be also understood as delta risk or beta risk (see [21]). Delta, calculated as the partial derivative of the option value with respect to the underlying asset price, is a measure of the sensitivity of the option value to the underlying asset. A short put option has a positive delta, which translates into a positive exposure to the market. Therefore, the value of a short put option increases if the market goes up. Another way to understand this is to isolate the volatility risk premium. Assume that the volatility risk premium is zero and options are fairly priced. Then the option premium only compensates for taking the downside risk. Put writers, who assume the downside risk, expect to be rewarded by the equity risk premium the same way as owning the underlying asset.

2.2 Trend Following Strategy

Trend following strategies have existed for a very long time. In the early 19th century, the legendary economist David Ricardo's quote "Cut short your losses; let your profits run on" marks an early exposition of the trend following strategies (see [20]). Trend following exploits the phenomenon that financial assets tend to exhibit return momentum. Investors in trend following strategies expect price movements to continue under certain market

conditions. Specifically, they take a long position in the market when it is in an up trend and they take a short position in the market when it is in a down trend.

The most basic trend following strategy would be based on market price momentum, which uses recent market returns to identify trends. Positive recent returns may signal an up trend and negative recent returns may imply a down trend. [32] provides the evidence of existence of trends in futures markets. It is also shown in [18] that time series momentum strategy generates consistent returns across all conventional asset classes from 1969 to 2015. Similar findings are reported when the backtesting period is extended to 135 years (see [25]). Another classic example of a trend following strategy uses moving average crossover, which is to long the market if 50-day moving average rises above 200-day moving average, and to short the market if 50-day moving average drops below 200-day moving average. [11] investigates the optimality of a trend following trading strategy based on moving average crossover. Previous study has also shown that trend following strategies are effective in reducing volatility and drawdowns (see [8]). Generally speaking, trend following is a popular strategy that is based on identifying trends and it appears to have worked well in the past.

The efficient Market Hypothesis (EMH) is a theory that financial asset prices reflect all publicly available information. Under EMH, securities are rationally priced and are always traded at their fair values. The existence of trends in financial asset prices clearly challenges EMH, and it allows sophisticated investors to generate profits by taking advantage of the trends without assuming extra risk. In general, under-reaction to market information is the universally accepted cause for trends to exist (see [8]). Under-reaction refers to the phenomenon that investors under-react or slowly react to new market information, causing stocks to slowly moving to their intrinsic value. On the contrary, over-reaction, which causes either overselling or overbuying, can lead to another price anomaly, mean reversion. Empirical research (see [3]) has shown security prices under-react to market information over horizons of one to twelve months and shows evidence of over-reaction over long horizons of three to five years.

Previous literature discusses a number of behavioral biases that can explain why investors under-react to market information. [3] relates under-reaction to conservativeness that causes slow updating of models in the face of new evidence. It is also discussed in [24] that the slow diffusion of news can explain why stock prices appear to drift after major corporate news announcements. The disposition effect, which is to sell winners too early and hold on to losers too long is found to contribute to the under-reaction effect, which further explains the price trends (see [16]).

2.3 Problem Description

Considering the attractive risk-adjusted return of the put-write strategy, we are interested in creating a trading strategy that achieves a similar return of the put-write strategy with an improved loss side risk. We suggest using another financial instrument or trading strategy to hedge the downside risk of put-write strategy. We consider the trend following strategy in this research paper.

We analyze how the two strategies behave under different market scenarios. Recall that the put-write strategy is subject to huge losses only when the market crashes. On the other hand, trend following strategy promises to be profitable as long as there is a trend in the market. Suppose that the market is trending down, then a trend following strategy can potentially protect the put-write strategy. Therefore, we expect that, by using these strategies together, some downside risk can be hedged.

Chapter 3

Formulation for Dynamic Asset Allocation

3.1 Multi-period Asset Allocation

We formulate the investment problem as a multi-period asset allocation problem. The wealth of the portfolio is allocated into two assets: the put-write and trend following indices, which represent the return of the put-write and trend following strategies. We assume the initial cash injection is \$1 and there is no other cash injection thereafter. The investment lasts N periods and portfolio is rebalanced at the beginning of each period. We further place a constraint that neither shorting nor leverage of the indices is allowed.

Let the initial time $t_0 = 0$ and the investment horizon $t_N = T$. Consider a set of rebalancing times

$$\tau \equiv \{t_0 = 0 < t_1 < ... < t_{N-1}\}.$$

Then the fraction of the total wealth allocated to each asset is adjusted at time t_n, where $n = 0, ..., N-1$. Let $\vec{\rho}_n$ be the allocation control vector at t_n. The allocation is held constant in period (t_n, t_{n+1}). We assume $\vec{R}(t_n)$ is the vector of returns on assets in period

(t_n, t_{n+1}). Then the terminal wealth of the portfolio $W(T)$ is determined as follows,

$$
\begin{aligned}
&W(t_0) = 1 \\
&\text{for } n = 0, 1, ..., N-1 \\
&\qquad W(t_{n+1}) = W(t_n)\vec{\rho}_n^{\intercal} e^{\vec{R}(t_n)} \\
&\text{end} \\
&\text{subject to} \quad \mathbf{0} \leq \vec{\rho}_n^{\intercal} \leq \mathbf{1},\ \mathbf{1}^{\intercal}\vec{\rho}_n = 1,\ \forall\, n = 0, ..., N-1,
\end{aligned}
\tag{3.1}
$$

where we have two constraints on the control vector. The inequality $\mathbf{0} \leq \vec{\rho}_n^{\intercal} \leq \mathbf{1}$ translates into the constraint that no shorting or leverage of either index is allowed. And the other constraint $\mathbf{1}^{\intercal}\vec{\rho}_n = 1$ means that we invest 100% of the wealth into these two indices and we do not hold cash in our portfolio.

3.2 Objective Function

We consider using the quadratic based objective function advocated in [31] as follows,

$$
\min_{\vec{\rho}_0, ..., \vec{\rho}_{N-1}} E\Big[\big(W^* - W(T)\big)^2\Big], \tag{3.2}
$$

where W^* is a pre-defined target value. The objective function penalizes for missing or exceeding the target at terminal. The objection function in (3.2) has been shown to generate pre-commitment multi-period mean variance optimal strategy in [42].

In this research paper, we choose a one-sided quadratic function used in [28] where it is used to generate optimal control for defined contribution pension plan. The objective function is as follows,

$$
\min_{\vec{\rho}_0, ..., \vec{\rho}_{N-1}} E\Big[\max(W^* - W(T), 0)^2 + \lambda W(T)\Big], \tag{3.3}
$$

where λ is a negative constant with a small absolute value. This objective function in (3.3) penalizes only the shortfall if terminal wealth does not meet target wealth W^*. In other words, the objective function is a performance measure focusing on the loss side risk. Here we add a small weight λ to push for higher terminal wealth when $W(T) \gg W^*$. We choose $\lambda = -10^{-6}$ in our numerical experiments. By minimizing the objective function in (3.3),

we expect the loss side risk of the portfolio to be reduced. Adding constraints in (3.1), our problem becomes a constrainted optimization problem, as follows,

$$\begin{aligned} &\min_{\vec{\rho}_0,\dots,\vec{\rho}_{N-1}} E\Big[\max(W^* - W(T), 0)^2 + \lambda W(T)\Big] \\ &\text{s.t.} \quad \mathbf{0} \le \vec{\rho_n} \le \mathbf{1},\ \mathbf{1}^\intercal \vec{\rho_n} = 1,\ \forall\, n = 0, ..., N-1. \end{aligned} \tag{3.4}$$

3.3 Three Optimal Controls and the Benchmark

We consider three optimal control strategies: (i) an optimal constant weight control, (ii) an optimal deterministic control, and (iii) an optimal stochastic control. All three optimal controls minimize the objective function, but they have different feature variables. The optimal constant weight control is the optimal constant holding strategy through the entire investment horizon. The optimal deterministic control is the optimal control that is only a function of the time. In the optimal stochastic control, the feature variables include the time t_n and a stochastic state variable $W(t_n)$, i.e. the portfolio wealth at time t_n. In other words, the optimal stochastic control is the optimal control which is a function of the time and current wealth. For simplicity, we omit the word "optimal" and refer to these strategies as the constant weight, deterministic and stochastic controls from now on. We only have two features for the stochastic control in this research paper, but we can consider adding additional features in the future. Additional features may include implied volatility and technical indicators which identify the strength of a trend in the market.

For the benchmark strategy, we consider a constant weight strategy, which is optimized for the Sharpe ratio. The Sharpe ratio is a measurement of the excess return per unit of risk, typically estimated by the standard deviation of returns. The optimal weight for the Sharpe ratio is determined as follows,

$$\begin{aligned} &\underset{\rho_{put}}{\text{argmax}} \left(\frac{\rho_{put} \cdot \hat{\mu}(R_p) + (1-\rho_{put}) \cdot \hat{\mu}(R_t) - r_f}{\sqrt{\rho_{put}^2 \hat{\sigma}^2(R_p) + (1-\rho_{put})^2 \hat{\sigma}^2(R_t) + 2(1-\rho_{put})\rho_{put} \hat{Cov}(R_p, R_t)}} \right) \\ &\text{s.t } 0 \le \rho_{put} \le 1. \end{aligned} \tag{3.5}$$

Here, ρ_{put} is the proportion of wealth invested in the put-write index. $\hat{\mu}(R_p)$ and $\hat{\mu}(R_t)$ are the annualized mean returns of the put-write index and trend following index. $\hat{\sigma}(R_p)$ and $\hat{\sigma}(R_t)$ are the annualized volatility of the return of the two indices. $\hat{Cov}(R_p, R_t)$ is the annualized covariance of the return of the two indices. We first calculate $\hat{\mu}(R_p)$, $\hat{\mu}(R_t)$, $\hat{\sigma}(R_p)$, $\hat{\sigma}(R_t)$, and $\hat{Cov}(R_p, R_t)$ using the daily return, then we scale them to the annualized level. r_f is the risk-free rate.

We use historical data for the above calculation, where $\hat{\mu}(R_p)$, $\hat{\mu}(R_t)$, $\hat{\sigma}(R_p)$, and $\hat{\sigma}(R_t)$ are reported in Table 5.1. The expected returns $\hat{\mu}(R_p)$ and $\hat{\mu}(R_t)$ are calculated using log returns and they are geometric mean returns. The risk-free rate r_f is 2.5%. The optimized weight in the put-write strategy is 50.7%. We also calculate the optimized weight using arithmetic mean returns, where simple returns are used to calculate the expected returns. The optimized weight in the put-write strategy is 50.5%. Since the two weights are very close, we choose the constant strategy with 50.7% in the put-write strategy as the benchmark strategy and we refer to 50.7% as the Sharpe ratio weight in the following context. In the numerical experiment section, we will compare the three proposed control strategies with the benchmark strategy, in terms of loss side risk.

3.4 Represent Controls by a Neural Network

Now we consider solving for the optimal controls in the constrained optimization problem in (3.4). Assume that we want to use L sample paths for training and the investment lasts N periods. Let M be the number of assets in the portfolio. For a general multi-asset allocation problem, the optimization problem in (3.4) has $\mathcal{O}(MNL)$ variables with $\mathcal{O}(MNL)$ constraints, as it solves for the controls at each rebalancing time along each path. Since L is typically a large number, the problem becomes computationally difficult to solve. In addition, it does not provide a solution at different paths.

Here, we compute the optimal controls using a simple Neural Network (NN) with one hidden layer. Taking the stochastic control for an example, the control vector at time t_n, $\vec{\rho}_n$, is a function of the feature vector $F(t_n)$, which includes time and current wealth, i.e., $F(t_n) = [t_n, W(t_n)]$. In other words, input into the NN is the feature vector $F(t_n)$ and output from the NN is the control vector $\vec{\rho}_n$. In this study, instead of using a deep neural network, we use a simple neural network with one hidden layer, shown in Figure 5.1, to represent the control vector. We note that "shallow learning" is found to outperform "deep learning" in an asset pricing problem, which is likely due to low signal-to-noise ratio in asset pricing problems (see [22]). Good results are also obtained in an variable annuities portfolio valuation problem with a NN with only one hidden layer (see [23]).

As shown in Figure 3.1, there are two nodes in the input layer, three nodes in the hidden layer, and two nodes in the output layer. In the input layer, the two nodes represent two features of the stochastic control, namely time and the current wealth. In the hidden layer, we use the sigmoid function as the activation function,

$$\sigma(u) = \frac{1}{1+e^u}. \tag{3.6}$$

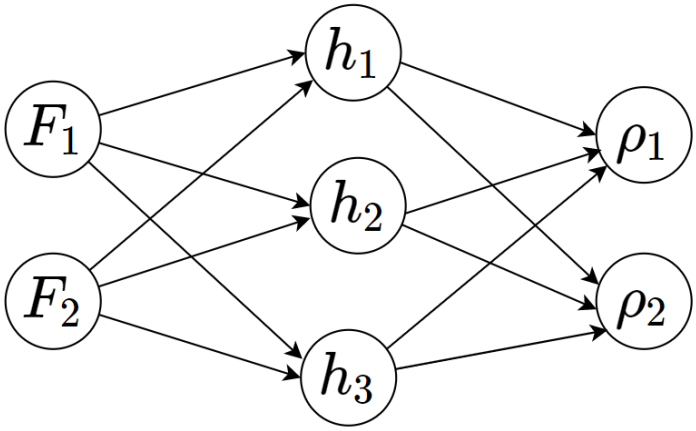


Figure 3.1: A NN representing the stochastic control function

Let $h \in R^3$ be the output of the hidden layer. Assume the matrix $z \in R^{2\times 3}$ is the weight matrix from the inputs layer $F(t_n) \in R^2$ to the hidden layer $h \in R^3$. Vector $z^\intercal F(t_n)$ corresponds to the values stored in the three nodes in the output layer. Assume $(z^\intercal F(t_n))_j$ is the j-th element in the vector. The output of the hidden layer h becomes

$$h_j = \sigma\big((z^\intercal F(t_n))_j\big) \quad \text{for} \quad j = 1, 2, 3. \tag{3.7}$$

In the output layer, we use the logistic sigmoid function as the activation function. Assume the matrix $x \in R^{3\times 2}$ is the weight matrix from the hidden layer $h \in R^3$ to the output layer $\vec{\rho}_n \in R^2$. Vector $x^\intercal h$ corresponds to the values stored in the two nodes in the output layer and let $(x^\intercal h)_m$ is the m-th element in the vector. Then the output layer ρ_n becomes

$$(\vec{\rho}_n)_m = \frac{e^{(x^\intercal h)_m}}{\sum_{i=1}^{2} e^{(x^\intercal h)_i}} \quad \text{for} \quad m = 1, 2. \tag{3.8}$$

One benefit of using this activation function is that controls automatically satisfy the constraints,

$$\mathbf{0} \le \vec{\rho}_n \le \mathbf{1}, \ \mathbf{1}^\intercal \vec{\rho}_n = 1.$$

These constraints are originally introduced in the formulation of the problem (3.1). Then for the stochastic control, the constrained optimization problem (3.4) becomes an unconstrained optimization problem with 12 variables which correspond to the weight matrices

z and x. The problem is as follows,

$$\begin{aligned} &\min_{z\in R^{2\times 3}, x\in R^{3\times 2}} E\Big[\max(W^* - W(T), 0)^2 + \lambda W(T)\Big] \\ &\text{s.t } h_j = \sigma\big((z^\intercal F(t_n))_j\big) \quad \text{for} \quad j = 1, 2, 3 \\ &(\rho_n)_m = \frac{e^{(x^\intercal h)_m}}{\sum_{i=1}^{2} e^{(x^\intercal h)_i}} \quad \text{for} \quad m = 1, 2. \end{aligned} \tag{3.9}$$

We can also compute the constant weight and deterministic controls using the same NN framework by allowing the feature to be a constant or a function of time.

Chapter 4

Optimization Technique

It is well known that numerical optimization is the cornerstone of training a neural network. Generally speaking, there are two major types of optimization techniques, first-order and second-order methods. First-order methods, also known as gradient based methods, use the first-order derivative gradient to construct each optimization iteration, whereas the second-order methods use the second-order derivative, Hessian matrix, to decrease the objective function. In this section, we consider four optimization methods, including (i) the trust region method, (ii) the gradient descent method, (iii) the mini-batch gradient descent method, and (iv) the adaptive moment estimate (ADAM) method.

We will use the bootstrap resample data set with the expected blocksize of 5.8 days for training. We will explain the bootstrap resampling algorithm in detail in Chapter 6. The data set contains $100,000$ sample paths for the two indices and each path lasts 63 days (three business months). We refer to this data set as the training set in this chapter. We use the above four optimization methods to train the stochastic control by solving the unconstrained optimization problem (3.9). Then we compare the computation times and the "smoothness" of the objective function value trajectory. The experiments are performed using MATLAB 2017 on a computer with an Intel Core i5-8600k CPU. Then we investigate how the choice of the learning rate impacts the convergence of gradient-based method. We also study how the batch size impacts the optimization of mini-batch gradient descent. Finally, we show why we choose ADAM over the other three methods.

4.1 Trust Region Method

The trust region method is one of the most popular second-order optimization methods. A trust region is the subset of the region in which the objective function is closely approximated by a quadratic function $\tilde{f}$. The method is an iterative method. Assume x_{k-1} is given. The quadratic approximation can be written as follows,

$$f(x) \approx \tilde{f}(x) = f(x_{k-1}) + \nabla f(x_{k-1})^\intercal (x - x_{k-1}) + \frac{1}{2}(x - x_{k-1})^\intercal H(x_{k-1})(x - x_{k-1}),$$

where $\nabla f(x_{k-1})^\intercal$ is the first-order derivative of f and $H(x_{k-1})$ is the Hessian matrix, the second-order derivative of f at x_{k-1}. Then we solve for the simpler sub-problem

$$x^* = \underset{||x - x_{k-1}||_2 \leq \delta}{\operatorname{argmin}} \tilde{f}(x),$$

where δ is the size of the trust region. The next step is to use the following formula to examine if the solution to the sub-problem leads to a sufficient reduction of the objective function.

$$\alpha_k = [f(x_k) - f(x_{k-1})]/[\tilde{f}(x_k) - f(x_{k-1})].$$

The numerator gives the actual reduction of f and the denominator represents the estimated reduction of f. If α_k is greater than a small threshold $0 < \gamma < 1$, the iteration is said to be successful. We accept x_k and increase the trust region by enlarging δ. Otherwise, the iteration is unsuccessful. We set $x_k = x_{k-1}$ and decrease the trust region by reducing δ.

For computational comparison in this chapter we use the trust region method introduced in [10]. The first-order derivative is calculated explicitly, while the Hessian matrix is approximately using the finite difference method. The pseudo-code is shown in Algorithm 1. The details to update δ is not presented here and we refer the interested readers to [10]. One of the major advantages of second-order methods over gradient based methods is their faster convergence rate. The trust region method can be shown to have a quadratic convergence rate. However, in practice, computing the Hessian matrix is usually computationally expensive and overshadows its fast convergence rate.

In our experiment, we find the computation time for the trust region method is significantly longer than the gradient based methods. We use the trust region method to solve the optimization problem (3.9) on a smaller data set with only $10,000$ paths. This data set is sampled randomly from the training set that has $100,000$ paths. The input parameters of the trust region method are defined as follows:

$$K = 100,\ \epsilon = 10^{-7},\ \gamma = 0.1,\ \nu = 10^{-12}.$$

Here, K is the maximum number of iterations. ϵ is the stopping criterion for gradient (minimum L_∞ norm for $\nabla f(x_{k-1})$) and ν is the stopping criterion for trust region size (minimum δ).

Figure 4.1 shows the plot of objective function value at each iteration. We remark that the optimization takes 19.5 minutes on the smaller data set with $10,000$ paths. Using the trust region method on the training set will take hours. Due to its long computation time, we need a faster method to optimize the objective function on the training set.

Algorithm 1 Trust Region Method

Input: Objective function $f(x)$; Maximum number of iterations K ; Stopping criterion for gradient (minimum L_∞ norm) ϵ ; Stopping criterion for trust region size (minimum δ) ν; Threshold for a successful iteration $0 < \gamma < 1$;

Initialize: Size of the trust region δ_0; Starting point x_0; Number of iterations $k = 0$

Compute the first order derivative vector $\nabla f(x_0)$
and the second order derivative Hessian matrix $H(x_0)$ at x_0
while $k < K$ and $||\nabla f(x_n)||_\infty \geq \epsilon$ and $\delta \geq \nu$ **do**
  $k = k + 1$
  Let $\tilde{f}(x) = f(x_{k-1}) + \nabla f(x_{k-1})^\intercal (x - x_{k-1}) + \frac{1}{2}(x - x_{k-1})^\intercal H(x_{k-1})(x - x_{k-1})$
  Solve the subproblem $x_k = \text{argmin } \tilde{f}(x)$ s.t. $||x - x_{k-1}||_2 \leq \delta$
  if $[f(x_k) - f(x_{k-1})]/[\tilde{f}(x_k) - f(x_{k-1})] \leq \gamma$ **then**
    $x_k = x_{k-1}$ (Unsuccessful Iteration)
  end if
  Compute the first order derivative vector $\nabla f(x_n)$
  and the second order derivative Hessian matrix $H(x_n)$ at x_n
  Adjust δ_k
end while
return x_k

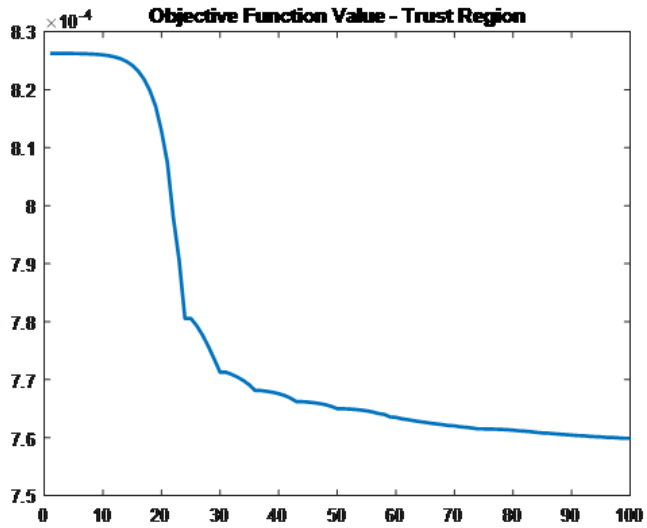


Figure 4.1: Trust region method: objective function value at each iteration (10,000 sample paths); computation time is 19.5 mins; terminal objective value is 0.000748

4.2 Gradient Descent

The gradient descent method is one of the most popular optimization algorithms and it is commonly used for training neural networks. The gradient descent method can be understood as iteratively moving in the opposite direction of the gradient in order to minimize the objective function, which can be described by the formula below,

$$x_k = x_{k-1} - \eta \nabla f(x_{k-1}),$$

where $\eta > 0$ is the learning rate. The process is repeated until the maximum number of iterations is reached or the magnitude of the gradient is small enough. In the later case, it corresponds to converging to a stationary point. There is a trade-off in selecting an appropriate learning rate. With a large learning rate, the objective function is often quickly reduced, but the risk of overshooting the lowest point is high. If a small learning rate is chosen, it is less likely to overshoot, but it takes a painfully long time to get to the solution. We show such findings later in Figure 4.2. The pseudo-code of the gradient descent method is shown in Algorithm 2. Compared to the trust region method, the gradient descent method has a much lower convergence rate. For convex and differential

objective function f with Lipschitz continuous gradient with a constant $L > 0$, i.e.,

$$||\nabla f(x) - \nabla f(y)|| \leq L||x - y|| \text{ for any } x, y,$$

the gradient descent method with $\eta = \frac{1}{L}$ has a convergence rate $\mathcal{O}(1/I)$, where I is the number of iterations (see [6]). We refer to it as sublinear convergence rate. In practice, the objective function may not have such properties, so it can be difficult to determine the convergence rate of the gradient descent method.

Algorithm 2 Gradient Descent Method

Input: Objective function $f(x)$; Maximum number of iterations K ; Stopping criterion (minimum L_∞ norm) for gradient ϵ ; Learning rate η

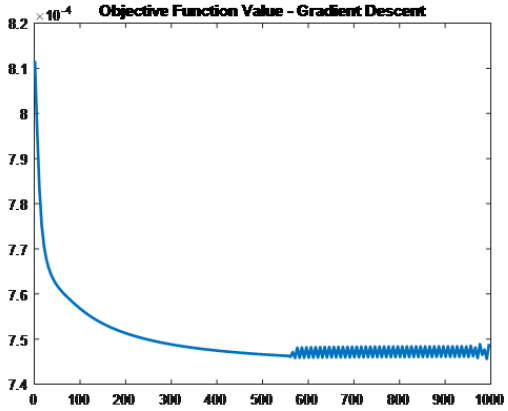
Initialize: Starting point x_0; Number of iterations $k = 0$

1: Compute the first order derivative vector $\nabla f(x_0)$ at x_0
2: **while** $k < K$ and $||\nabla f(x_n)||_\infty \geq \epsilon$ **do**
3: $k = k + 1$
4: $x_k = x_{k-1} - \eta \nabla f(x_{k-1})$
5: Compute the first order derivative vector $\nabla f(x_n)$ at x_n
6: **end while**
7: **return** x_k

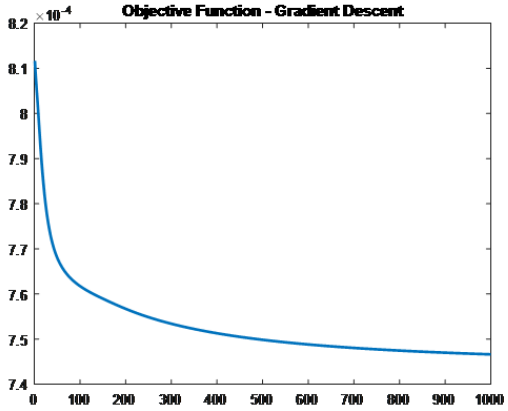
We use the gradient descent method to solve the optimization problem (3.9) on the training set. We use the following input parameters:

$$K = 1000, \ \epsilon = 10^{-7},$$

where K is the maximum number of iterations and ϵ is the stopping criterion for gradient (minimum L_∞ norm for $\nabla f(x_{k-1})$). Figure 4.2 shows the plot of the objective function with two different learning rates at each iteration. We note that the objective function oscillates near the minimum for the larger learning rate $\eta = 200$. With the smaller learning rate $\eta = 100$, it appears that the algorithm smoothly converges to a minimum. We notice that the gradient descent method is much faster than the trust region method, but spending an hour to solve one optimization problem is still not efficient enough, as we seek to fix the median of the three control strategies to the same as benchmark strategy by a bisection method.



(a) $\eta = 200$, computation time is 64.8 mins, terminal objective function value is 0.000749



(b) $\eta = 100$, computation time is 64.4 mins, terminal objective function value is 0.000747

Figure 4.2: Gradient descent method: objective function value at each iteration

4.3 Mini-batch Gradient Descent

The gradient descent method takes all training data into consideration in every iteration. We use the mean of the gradients of all the training examples to update x_k. The method is computationally heavy for a large data set. Here, we consider using the mini-batch gradient descent method to overcome this challenge.

A mini-batch refers to a subset of training examples which is usually a lot smaller than the entire data set. At each iteration, we randomly sample from the data set and use that a smaller sample set for the gradient evaluation. When the sample size is equal to one, the method is called stochastic gradient descent. This family of methods offer convergence faster than the gradient descent, because each iteration takes less computation time. Since the sample used to evaluate the gradient is randomly selected at each iteration, the objective function tends to fluctuate around a minimum. Algorithm 3 describes how the method is implemented.

Algorithm 3 Mini-batch Gradient Descent Method

Input: Objective function $f(x)$; Maximum number of iterations K ; Stopping criterion (minimum L_∞ norm) for gradient ϵ ; Learning rate η ; Batchsize B

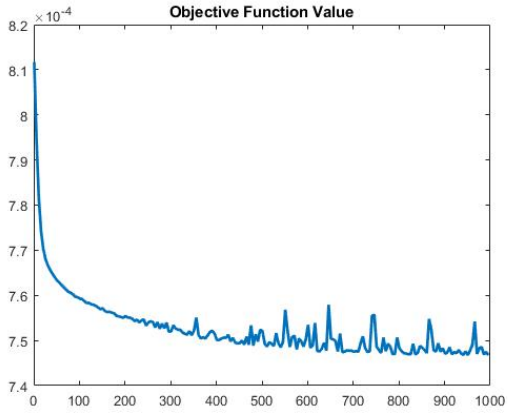
Initialize: Starting point x_0; Number of iterations $k = 0$

1: Random select a subset S of size B from the entire data set
2: Compute the first order derivative vector $\nabla f(x_0)$ at x_0 on S
3: **while** $k < K$ and $||\nabla f(x_n)||_\infty \geq \epsilon$ **do**
4: $k = k + 1$
5: $x_k = x_{k-1} - \eta \nabla f(x_{k-1})$
6: Random select a sample S of size B from the entire data set
7: Compute the first order derivative vector $\nabla f(x_n)$ at x_n on S
8: **end while**
9: **return** x_k

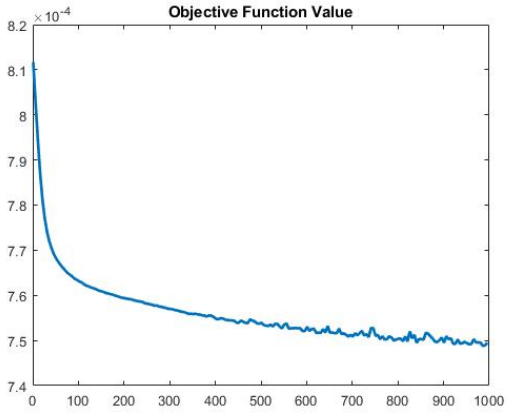
We use the mini-batch gradient descent method to solve the optimization problem (3.9) on the training set. We use the following input parameters:

$$K = 1000, \epsilon = 10^{-7},$$

where K is the maximum number of iterations and ϵ is the stopping criterion for gradient (minimum L_∞ norm for $\nabla f(x_{k-1})$). We adjust the learning rate and batch size to study their impact on optimization. Figure 4.3 shows the plot of objective function with two different learning rates. We note that the objective function value fluctuates more for the larger learning rate $\eta = 200$ and it declines more smoothly with the smaller learning rate $\eta = 100$, which is in line with our result for the gradient descent method in Figure 4.2. Figure 4.4 shows the plot of objective function with two different batch sizes. We notice that as batch size gets smaller, computation time is shorter, but the objective function value declines in a more volatile manner. Using a batch size of one, namely stochastic gradient descent, will cause the optimization to be very volatile. Here, we believe that batch size equal to 10000, which is 10% of the entire training set, gives a good balance between computation time and the "smoothness" of the objective function value trajectory.

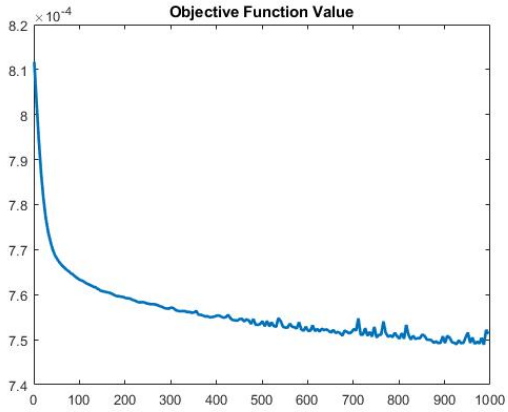


(a) $\eta = 200$, computation time is 22.1 mins, terminal objective function value is 0.000747

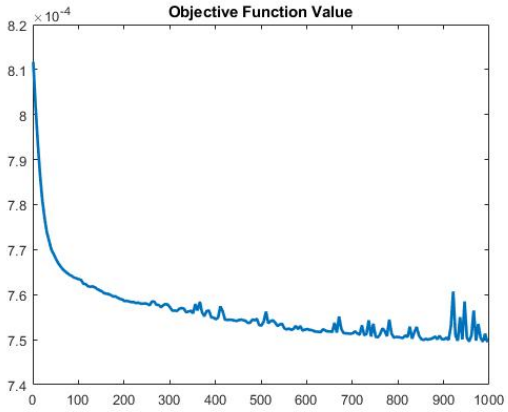


(b) $\eta = 100$, computation time is 21.7 mins, terminal objective function value is 0.000750

Figure 4.3: Mini-batch gradient descent method: objective function value at each iteration, $B = 10000$



(a) $B = 5000$, computation time is 19.7 mins, terminal objective function value is 0.000752



(b) $B = 1000$, computation time is 18.4 mins, terminal objective function value is 0.000750

Figure 4.4: Mini-batch gradient descent method: objective function value at each iteration, $\eta = 100$

4.4 ADAM

One common problem from the gradient descent and mini-batch gradient descent methods is that choosing a proper learning rate is difficult. As we can see from the results above, a very small learning rate leads to painfully slow convergence, whereas a very large learning rate can hinder convergence and causes the objective function to fluctuate around the minimum or even to diverge. One way to tackle this problem is to pre-define a learning rate schedule. Nonetheless, such a schedule needs to be defined in advance and it is not able to adapt to a data set's own characteristics.

Another difficulty of finding a proper learning rate arises from the fact that we apply the same learning rate to all parameter updates in the gradient descent and mini-batch gradient descent methods. In the case where the features are on different scales, we might want to choose a bigger learning rate for some features. It is argued in [36] that a constant learning makes it difficult to escape from a saddle point, at which the gradient is zero in all directions, making it difficult for the gradient descent and mini-batch gradient descent methods to escape. The better alternative is to use a method that computes adaptive learning rates for each parameter.

Now we consider the ADAM method introduced in [27]. ADAM, short for adaptive moment estimate, uses the first and second moments of the gradient to calculate the adaptive learning rates for each parameter. The n-th moment of a random variable is defined as the expected value of the variable to the power of n, i.e.,

$$m_k = E[X^n] \ .$$

The ADAM method stores an exponentially decaying average of the past squared gradients, as well as an exponentially decaying average of the past gradients:

$$m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k$$

$$v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k^2 \ .$$

Here, m_k and v_k refer to the estimate of the first moment and second moment of the gradients, respectively. g_k^2 stands for the element-wise square $g_k \odot g_k$. All operations on vectors are element-wise in this method. Since m_k and v_k are initialized as vectors of zero, the estimates are biased towards zero during the initial steps. The biases are corrected as follows,

$$\hat{m}_k = m_k / (1 - \beta_1^t)$$

$$\hat{v}_k = v_k / (1 - \beta_2^t) \ .$$

Then the parameters are updated:

$$x_k = x_{k-1} - \frac{\eta}{\sqrt{\hat{v}_k} + \tau}\hat{m}_k \ ,$$

where $\eta > 0$ is the learning rate and τ is a constant to adjust the learning rate. We implement the ADAM method based on mini-batch gradient descent and the complete algorithm is shown in Algorithm 4.

Algorithm 4 ADAM Method

Notation: We use g_k to indicate the first-order derivative and g_k^2 stands for the element-wise square $g_k \odot g_k$. All operations on vectors are element-wise. β_1^t and β_2^t represent β_1 and β_2 to be power of t

Input: Objective function $f(x)$; Maximum number of iterations K ; Stopping criterion (minimum L_∞ norm) for gradient ϵ ; Learning rate η ; Batchsize B; Exponential decay rates for the moment estimates β_1 and β_2; A constant to adjust the learning rate τ

Initialize: Starting point x_0; Number of iterations $k = 0$; Estimate of the first moment $m_0 = \mathbf{0}$; Estimate of the second moment $v_0 = \mathbf{0}$

Random select a subset S of size B from entire data set
Compute the gradient g_0 at x_0 on S
while $n < N$ and $||g_k||_\infty \geq \epsilon$ **do**
  $k = k + 1$
  $m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k$
  $v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k^2$
  $\hat{m}_k = m_k / (1 - \beta_1^t)$
  $\hat{v}_k = v_k / (1 - \beta_2^t)$
  $x_k = x_{k-1} - \frac{\eta}{\sqrt{\hat{v}_k} + \tau}\hat{m}_k$
  Random select a subset S of size B from entire data set
  Compute the gradient g_k at x_n on S
end while
return x_k

We use the ADAM method to solve the optimization problem (3.9) on the training set. We use the following input parameters:

$$K = 1000, \ \epsilon = 10^{-7}, \ B = 10000,$$

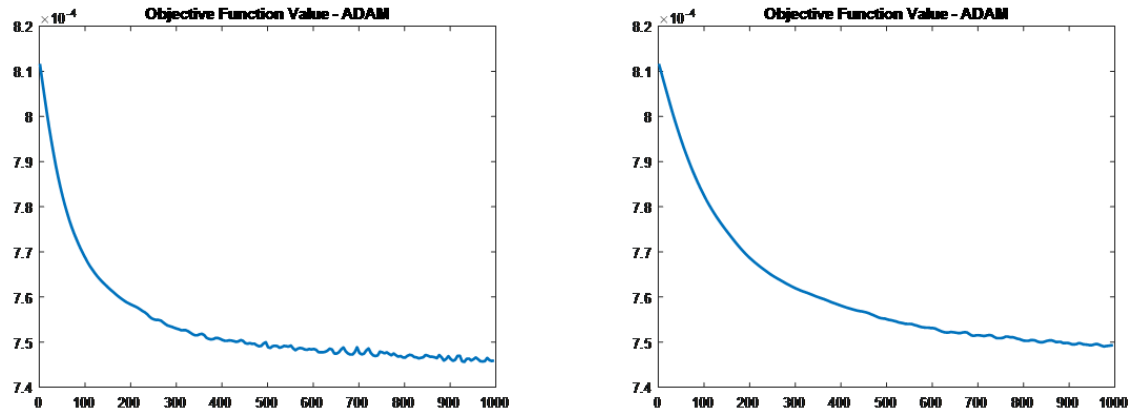
where K is the maximum number of iterations, B is the batchsize and ϵ is the stopping criterion for gradient (minimum L_∞ norm for $\nabla f(x_{k-1})$). We also choose the recommended

setting

$$\beta_1 = 0.9,\ \beta_2 = 0.999,\ \tau = 10^{-8},$$

from [27]. β_1 and β_2 are the exponential decay rates for the moment estimates and τ is a constant to adjust the learning rate.

Here, we show the optimization results of two small learning rates $\eta = 0.001$ or $\eta = 0.002$ in Figure 4.5. We remark that both learning rates generate a generally smooth reduction in objective function value. Even though both learning rates are significantly smaller than the ones used in the gradient descent and mini-batch gradient descent methods ($\eta = 100$ or $\eta = 200$), the ADAM method dynamically adjusts the learning rates, leading to less fluctuation and relatively good convergence. As shown in Figure 4.6, we choose learning rate $\eta = 0.001$ and increase the maximum number of iterations to 2000. We observe that objective function value continues declining after 1000 iterations, but the improvement in optimized objective function is negligible compared to using the bigger learning rate and less iteration, i.e., $\eta = 0.002$, $K = 1000$. In conclusion, the ADAM method with $\eta = 0.002$, $K = 1000$, $B = 10000$ gives the best balance between computation time among the four optimization methods, "smoothness" of the objective function value trajectory and terminal objective function value. Therefore, we use the ADAM method with these parameters to solve the stochastic optimization problems (3.9).



(a) $\eta = 0.002$, computation time is 21.2 mins, terminal objective function value is 0.000746

(b) $\eta = 0.001$, computation time is 22.3 mins, terminal objective function value is 0.000749

Figure 4.5: ADAM method: objective function value at each iteration

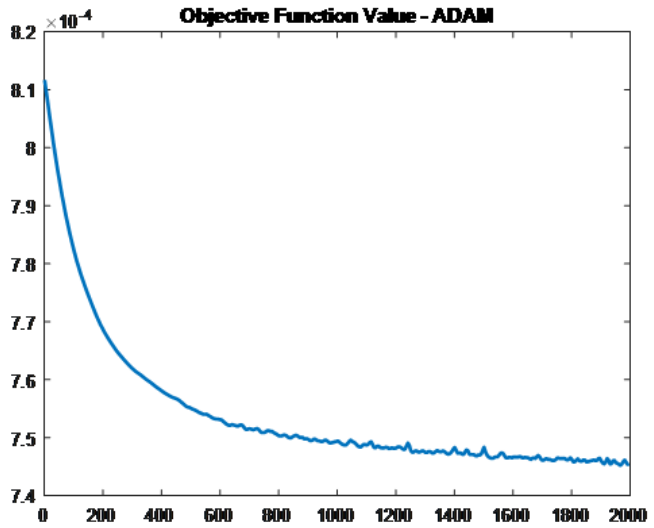


Figure 4.6: ADAM method: objective function value at each iteration, $\eta = 0.001$, computation time is 45.5 mins, terminal objective function value is 0.000746

Chapter 5

Data

We use the S&P 500 Put Write Index (PUT Index) as a proxy for the put-write strategy. The PUT Index measures the performance of a hypothetical strategy that sells a sequence of one-month, at-the-money, put options on the S&P 500 Index and then invests the premiums in the one-month and three-month Treasury Bills. The number of puts sold is limited to the amount such that Treasury Bills held in the collateral portfolio can finance the maximum potential loss from final settlement of the puts.

For the trend following strategy, the Societe Generale (SG) Commodity Trading Advisor (CTA) Index is selected as the proxy. The SG CTA Index is a pool of CTA funds selected from the larger managers that are open to new investment. A CTA fund uses trend following strategy as its primary trading strategy. It originates from commodity futures trading, but has now been extended to other asset classes. The SG CTA Index consists of CTA funds on equity, fixed income, currency and commodities.

Both time series range from January 2002 to April 2019. There are 4362 daily close prices for the PUT Index, while the SG CTA Index has 4522 daily close prices over the same time period. This is because the SG CTA Index consists of not only funds on the S&P 500, but also funds on the currency and commodity market. Therefore, the index is published on New York Stock Exchange (NYSE) non-trading holidays as well. We remove these extra data points, so that these two time series have the same amount of daily close prices. Then we calculate the daily log returns. By removing the closing price on a holiday, we combine the daily returns on the holiday and the next business day together. For example, suppose the close price of the SG CTA Index is P_h on a holiday and P_{h-1} and P_{h+1} are the business day before and after the holiday. Then we omit the close price on the holiday and let $r_{h+1} = log(\frac{P_{h+1}}{P_{h-1}})$. In this way, the total return over the entire time

horizon remains the same. Here, we make an assumption that no trading of either index is performed on the NYSE non-trading holidays.

Figure 5.1 shows the historical performance of the two indices. Over the 17 years, the PUT Index outperformed the SG CTA Index in terms of the cumulative return. However, the put-write strategy is the more risky strategy as the PUT Index is subject to substantial losses in every major bear markets. For example, big drawdowns are observed in 2002, 2008 and 2018. These years correspond to the tech bubble, the great financial crisis and the 2018 Q4 sell-off respectively. It is interesting to notice that the SG CTA Index shows strong performance while the PUT Index falls together with the market. The observation shows that combining the put-write strategy with the trend following strategy seems promising. We can also see from the plot that the trend following index yields less return, but it is the relatively safer strategy of the two.

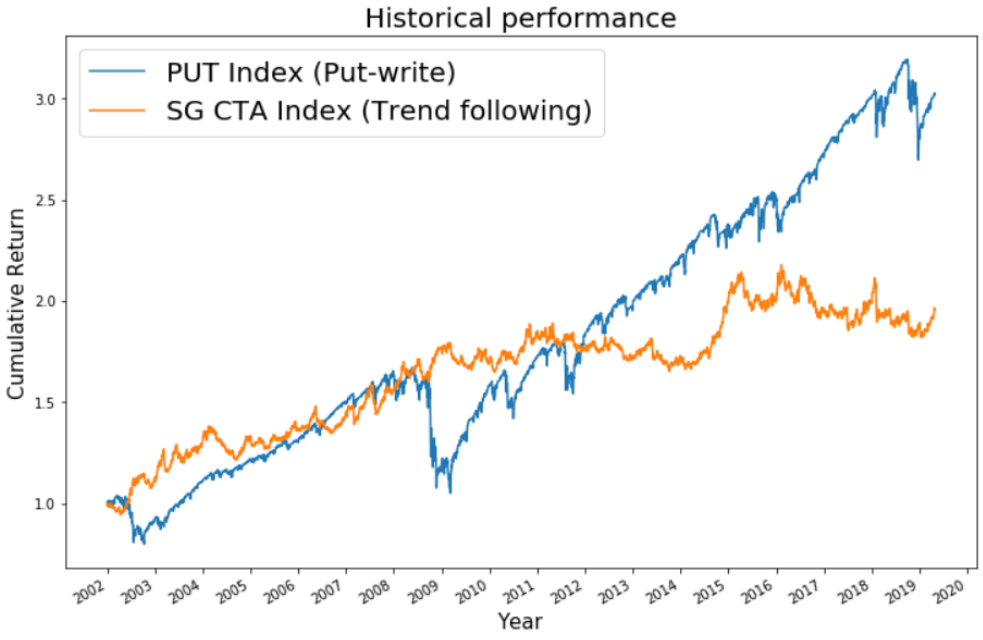


Figure 5.1: Historical Performance of the PUT Index and SG CTA Index

Table 5.1 shows the median, mean, volatility for annualized return, Sharpe ratio, and maximum drawdown for the two indices. After removing the extra data points, there are 4362 daily close prices for each index. Median, mean and volatility are calculated using the 4361 daily log returns, then scaled to annualized returns. Sharpe ratios are calculated

as follows,

$$\frac{E(r_p) - r_f}{\sigma_p},$$

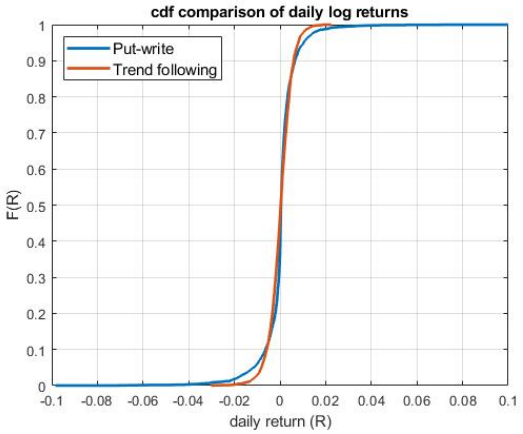
using the mean return and volatility reported in this table. We assume the risk-free rate is 2.5%. We also report the skewness and kurtosis calculated using the 4361 daily log returns in Table 5.2. We observe that the daily log returns of both strategies are negatively-skewed and the put-write strategy has a very long tail. Figure 5.2 shows the cumulative distribution function (cdf) comparison of the daily log returns of the two indices and we can see that the put-write strategy has a large loss side risk with many extreme values. We aim to develop a strategy that can significantly reduce the loss side risk of the put-write strategy in this study.

Strategy	**Median**	**Mean**	**Volatility**	**Sharpe ratio**	**Max drawdown**
Put-write	11.5%	6.4%	12.8%	0.30	-37.1%
Trend Following	7.3%	3.9%	8.0%	0.17	-16.5%

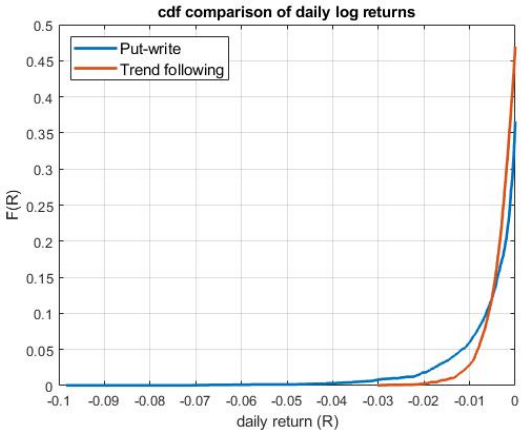
Table 5.1: Statistics for annualized log returns and maximum drawdown

Strategy	**Skewness**	**Kurtosis**
Put-write	-0.58	29.4
Trend Following	-0.46	5.4

Table 5.2: Skewness and kurtosis of daily log returns



(a) cdf comparison



(b) cdf comparison (loss side)

Figure 5.2: Cumulative distribution function (cdf) comparison of daily log returns

Chapter 6

Bootstrap Resampling

In order to use the proposed data driven approach, we need to generate more data than what is historically observed. We adopt the bootstrap resampling technique to generate the training and testing data sets. For each training and testing data set, 100,000 paths are sampled from the historical data and we construct each bootstrap resampled path as follows. If we divide the total investment horizon (T days) into h blocks of size b days, so that $T = hb$. Then h blocks of size b days are randomly selected with replacement from the historical data (from both the PUT Index and SG CTA Index). These blocks are then concatenated to form a single path. However, a randomly sampled block can exceed the range of time series. We use circular block bootstrap introduced in [34], which is wrapping around the historical data, to avoid end effects.

We sample in blocks in order to preserve possible serial correlation in the historical data. The choice of the blocksize is important and can have a large influence on the results (see [9]). We use the stationary bootstrap resampling technique introduced in [33] to reduce the impact of a fixed blocksize. Instead of using a fixed blocksize, we use a geometric distribution with an expected blocksize $\hat{b}$ to sample the blocksize. Then the blocksize b follows a geometric distribution and let $c = \frac{1}{\hat{b}}$, we have

$$P(b = k) = (1 - c)^{k-1}c \tag{6.1}$$

The pseudo-code to generate one sample path is described in Algorithm 5.

Algorithm 5 Use stationary bootstrap resampling to generate one sample path

Input: Vector of returns $X(s)$: $s = 1, \cdots, N_{total}$; Expected blocksize $\hat{b}$; Path length N

Initialize: current length $l = 0$; sampled path $P =$ zeros $(N, 1)$

while $l < N$ **do**
  Generate blocksize b from a geometric distribution with expected value $\hat{b}$, i.e. $b \sim \text{geo}(\hat{b})$ (Stationary bootstrap)
  Generate random starting index i from a uniform distribution from 1 to N_{total}, i.e. $i \sim \text{rand}(1, N_{total})$
  $b_r = \min(b, N - l)$
  for $m = 1, \cdots, b_r$ **do**
    $index =$mod$(i + m - 1, N)$ (Circular bootstrap)
    $P(l) = X(index)$
    $l = l + 1$
  end for
end while
return P

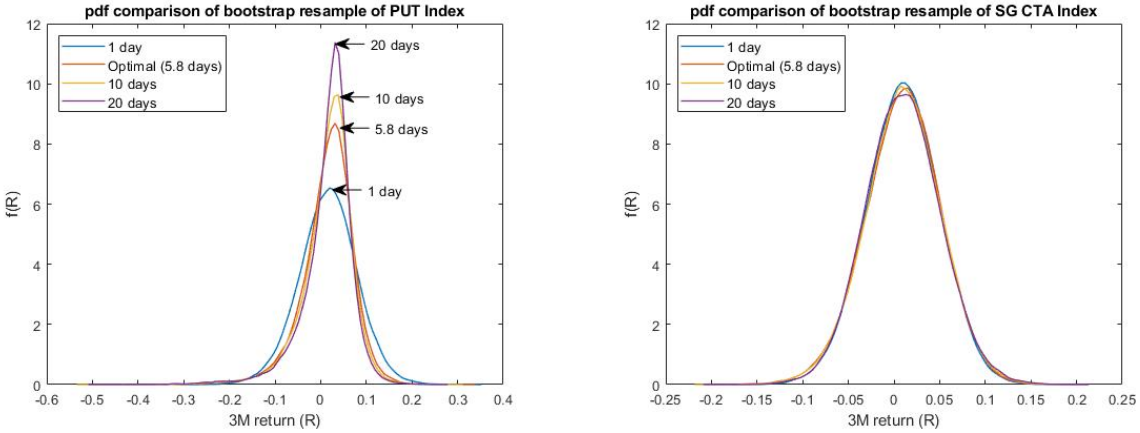
We use the algorithm described in [33] to determine the optimal expected blocksize $\hat{b}$. This approach has been previously used in other tests of portfolio allocation problems as well (see [13]). Table 6.1 shows the calculated optimal values for $\hat{b}$ for both indexes. When we perform bootstrap resampling, we need to simultaneously sample the same block from both historical time series (the PUT Index and SG CTA Index). It is unclear what blocksize is the best in our simultaneous resampling method. We use the average of the two values, 5.8 days, as the optimal expected blocksize for generating bootstrap resample data sets. We will test the control strategies on bootstrap resample data sets with different expect blocksizes, as well as the historical data paths.

Data Series	**Optimal expected blocksize $\hat{b}$ (days)**
PUT Index	7.3
SG CTA Index	4.4

Table 6.1: Optimal expected blocksize

In this research paper, we choose the investment horizon to be three months, which is equivalent as 63 business days. We use the stationary bootstrap resampling technique

to generate 100,000 sample paths from the historical data. As a result, we have one bootstrap resample data set, a matrix of dimension $100,000 \times 63$ for each index. We study the impact of the expected blocksize on the distribution of the overall return generated from bootstrap resampling. We generate bootstrap resample data sets with the expected blocksize $\hat{b} = 1, 5.8, 10, 20$ days and study the distribution of 100,000 three-month log return. Figure 6.1 show the probability density function (pdf) comparison of three-month log return. As we can see from Figure 6.1a, bootstrap resample data set for the PUT Index with a larger expected blocksize has higher degree of peakedness and a longer left tail. For the SG CTA Index, no major difference can be observed from Figure 6.1b.



(a) PUT Index (Put-write)

(b) SG CTA Index (Trend following)

Figure 6.1: Probability density (pdf) comparison of three-month log return

The skewness and kurtosis of each distribution are reported in Table 6.2 & 6.3. They are calculated using 100,000 three-month log returns. We observe that as the expected blocksize decreases from 20 to 1, skewness approaches 0 and kurtosis approaches 3 for the PUT Index. In other words, the three-month return distribution transforms into a normal distribution from a negatively-skewed long-tailed distribution. We remark that when the expected blocksize is 1 day, the actual blocksize is 1 day. Plug $c = \frac{1}{\hat{b}} = 1$ into equation 6.1 and we have $P(b = 1) = (1 - 1)^0 \times 1 = 1$. In other words, we are essentially sampling i.i.d. from historical data. By central limit theorem, the three-month return will tend to be normally distributed. The transformation is less obvious for SG CTA Index, as distributions of all expected blocksizes all resemble the normal distribution to some extent.

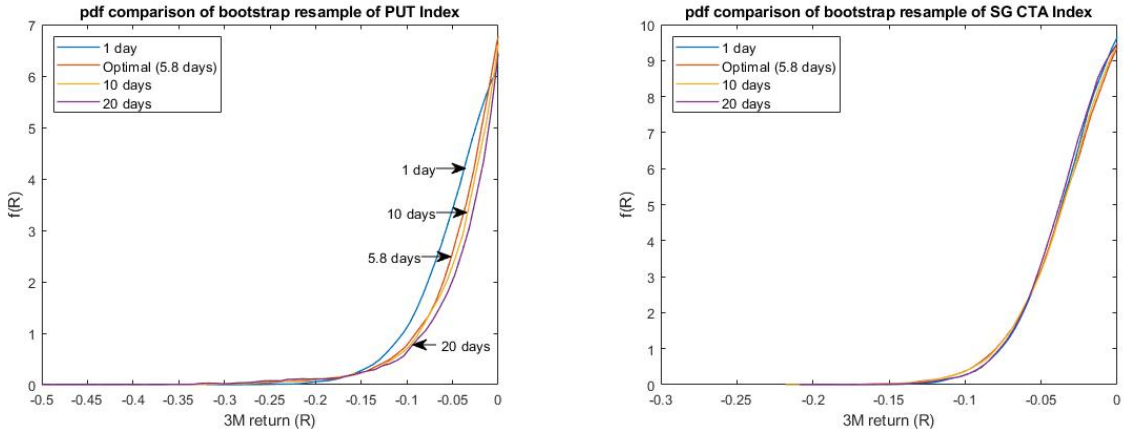
Expected Blocksize (days)	**1**	**5.8**	**10**	**20**
PUT Index	-0.09	-0.87	-1.14	-1.55
SG CTA Index	-0.06	-0.16	-0.10	0.05

Table 6.2: Skewness of bootstrap resample data sets

Expected Blocksize (days)	**1**	**5.8**	**10**	**20**
PUT Index	3.4	5.8	6.9	8.7
SG CTA Index	3.0	3.3	3.3	3.2

Table 6.3: Kurtosis of bootstrap resample data sets

Figure 6.2 compares the loss side of the four pdf's. Table 6.4 shows the 95%-CVaR of bootstrap resample data sets. For the PUT Index, we find that the bootstrap resample data sets with larger expected blocksizes have more and bigger extreme values. When the market crashes, stocks decline for several days or even weeks consecutively. With large blocksizes, significant market crashes can be captured and it is possible that multiple bear markets are concatenated into one path. It is less likely for a smaller blocksize to include the consecutive declines, as the bootstrap algorithm randomly selects blocks from historical data more frequently. The four pdf's of the SG CTA Index are alike. We believe this is due to the fact that the historical daily log return of this index has less and smaller extreme values (see Table 5.2 and Figure 5.2).



(a) PUT Index (Put-write) (b) SG CTA Index (Trend following)

Figure 6.2: Probability density comparison of three-month log returns (loss side)

Expected Blocksize (days)	**1**	**5.8**	**10**	**20**
PUT Index	-12.3%	-13.1%	-13.5%	-14.1%
SG CTA Index	-7.4%	-8.1%	-8.0%	-7.5%

Table 6.4: 95%-CVaR of bootstrap resample data sets

We conclude that bootstrap resample data sets with different expected blocksizes have different distributions. Later we present results where we train the Neural Network on the data set with the expected blocksize $\hat{b} = 5.8$ days and test it on the other three data sets.

Chapter 7

Optimal Allocation Analysis

In this chapter, we analyze and assess performance of optimal controls. Recall that we formulate the problem as a multi-period asset allocation problem. We have a portfolio of two assets, the PUT Index and the SG CTA Index. We place two constraints on the portfolio. The first constraint is that no shorting or leverage of any index is allowed. The other constraint is that 100% of the portfolio wealth is allocated into these two assets. The two constraints are automatically satisfied by adopting the Neural Network framework in Chapter 3.

We assume the initial wealth is \$1 and there is no subsequent cash injection. Transaction costs are not taken into consideration. The investment horizon is 3 months, which is equivalent as 63 trading days. The portfolio is rebalanced at the beginning of each day. We use the stationary bootstrap resampling technique introduced in Chapter 6 to generate 100,000 paths for training and testing. With each path spanning 63 days, each training and testing data set is of a matrix of dimension $100,000 \times 63$. The training data set is bootstrap resample data set with the optimal expected blocksize, 5.8 days. During training, we constrain the median return to be the same as that of the benchmark, by adjusting the target terminal wealth W^* in problem (3.9) using the bisection method. Then we compare the loss side risk of the three control strategies to the benchmark. We test the control strategies on bootstrap resample data sets with the expected blocksize = 1, 10 and 20 days. We also test the stochastic control on historical paths.

Three proposed control strategies are obtained by solving the unconstrained optimization problem (3.9). For neural networks, scaling input data is of great importance for efficient training. In practice, it is nearly always advantageous to apply pre-processing transformations to the features before it is presented to a neural network (see [5]). Since

the initial wealth is \$1 and there is no cash injection thereafter, the portfolio wealth at each timestep is approximately centered around \$1. We standardize the wealth with mean = 1 and standard deviation = 0.001. Then we normalize the other feature, i.e. time, on a scale of 0 to 1, i.e. $t_n = n/N$. We explicitly compute the objective function and its gradient. Then we solve the optimization problem using the ADAM method.

7.1 Training Results

Table 7.1 reports the training statistics for the three month returns of the three control strategies, as well as the benchmark strategy, the put-write strategy and the trend following strategy. Terminal wealth at the end of three months is first converted into three month log returns, i.e. $log\big(\frac{W(T)}{W(0)}\big)$. Median, mean and standard deviation are calculated using 100,000 three month log returns. The 95%-CVaR reflects the mean log return of the worst 5% terminal wealth. CVaR is a common measure of tail risk.

Strategy	**Median**	**Mean**	**Standard Deviation**	**95%-CVaR**
Constant Control	1.6%	1.3%	3.4%	-6.6%
Deterministic Control	1.6%	1.3%	3.4%	-6.6%
Stochastic Control	**1.6%**	**1.5%**	**3.3%**	**-6.2%**
Benchmark	**1.6%**	**1.4%**	**3.4%**	**-6.7%**
Put-write	2.2%	1.6%	5.7%	-13.1%
Trend Following	1.1%	1.0%	4.2%	-8.1%

Table 7.1: Training Results: Expected Blocksize = 5.8 days

We look at the constant weight and deterministic controls first. We observe that the two controls have exactly the same statistics for three month returns. This implies that the two controls might be similar. After fixing the median to be the same as the benchmark strategy, we observe that the mean return is slightly less than the benchmark and 95%-CVaR is marginally better, while the standard deviations are the same. When we compare the stochastic control to the benchmark strategy, we achieve a better mean return, a smaller standard deviation and a better 95%-CVaR, while maintaining the same median returns. In this table, we also report the statistics of the put-write and trend following strategies. The put-write strategy clearly gives the best mean return, but it is also the most volatile strategy and has the largest left tail risk. Compared to the put-write strategy, the trend following strategy is the safer strategy with a smaller standard deviation and a smaller loss side risk. The compromise is that the mean return of the stochastic control is much less

than the put-write strategy. Since the trend following strategy is inferior to the benchmark strategy in terms of smaller median and mean returns, a larger standard deviation and a larger 95%-CVaR, we will not focus on the comparison of the three control strategies to the trend following strategy.

Next, we analyze the constant weight control and the deterministic control. The constant weight control suggests allocating 50.2% of the asset in the put-write strategy. We note that the deterministic control is a function of time only. The deterministic control is centered around 50.1% in the put-write strategy. Figure 7.1 shows the deterministic control over 3 months and we observe negligible time dependence. The benchmark strategy allocates 50.7% of wealth in the put-write strategy. Hence, the constant weight and deterministic controls are not significantly different from the benchmark. This also explains why the statistics in Table 7.1 are so close for these three strategies.

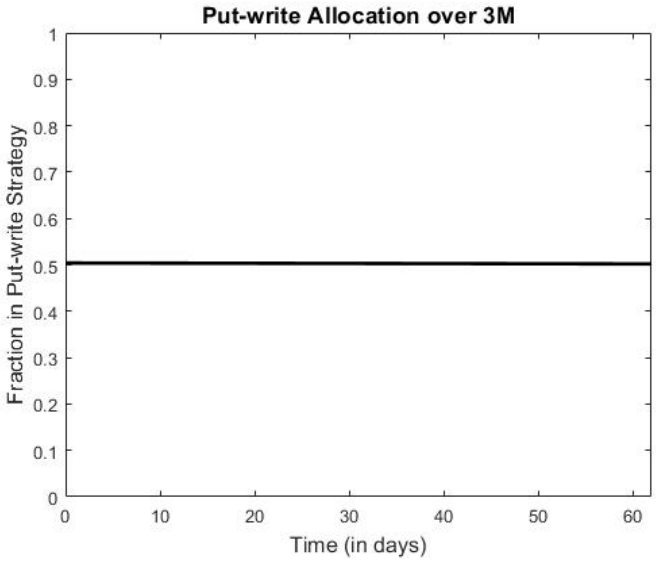


Figure 7.1: Deterministic control

We now focus on the comparison between the benchmark and the stochastic control. Figure 7.2 compares the probability density of the stochastic control and the benchmark. The vertical line at $W(T) = 1$ separates the gains and the losses. Since we are more interested in the loss scenarios, we will focus on the comparison of the loss side. The benchmark strategy obviously has a larger loss side than the stochastic control.

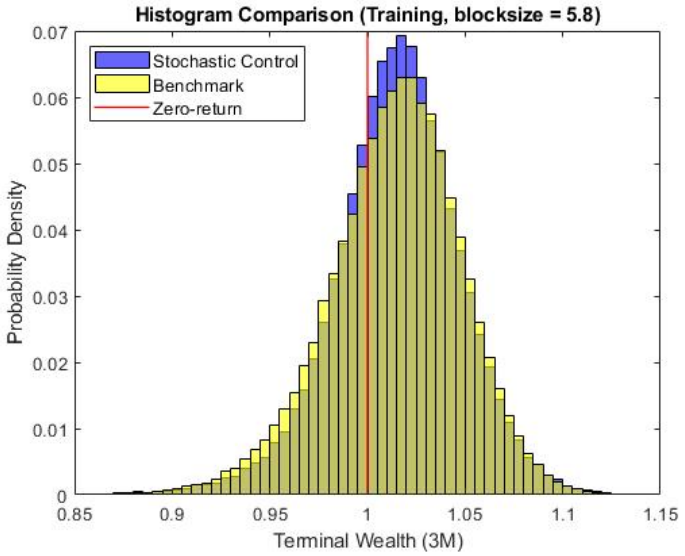
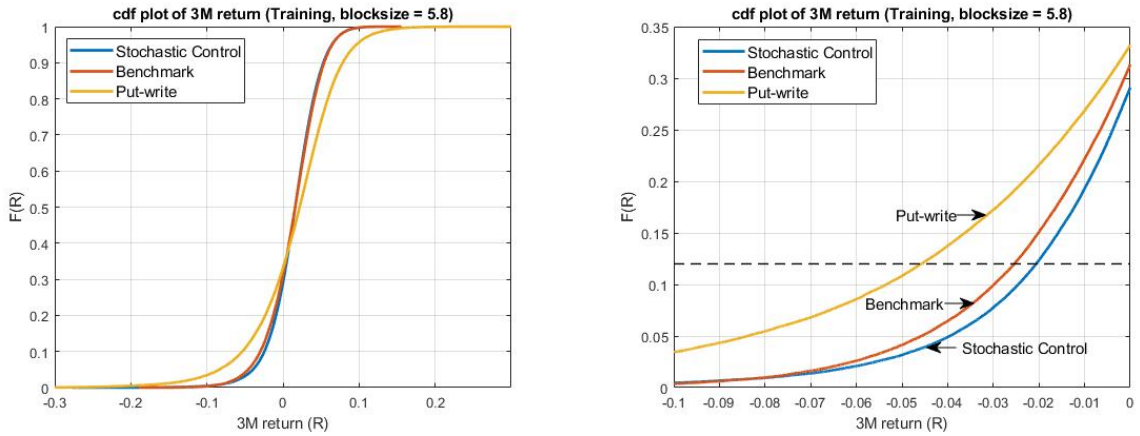


Figure 7.2: Training: density histogram

We also compare the cumulative distribution function (cdf) of the stochastic control, benchmark and put-write strategy in Figure 7.3a. From the cdf plot on the loss side, it can be observed that the stochastic control is relatively more similar to the benchmark strategy than the put-write strategy. We also note that the put-write strategy not only has the largest loss side, but the largest right side as well. When we incorporate the trend following strategy into the put-write strategy, we are giving up some positive expected return to reduce the downside risk. If we zoom into the loss side (7.3b), we remark that stochastic control has the smallest loss side, followed by the benchmark strategy, while the put-write strategy has a much larger loss side than the other two control strategies. The concept of the first-order stochastic dominance was first introduced in [35]. It is defined as, at a certain probability level, if strategy A has a better outcome than strategy B, then A is stochastically dominant over B. Further, if it is true on an interval of probability, it is called a region of stochastic dominance. The plot demonstrates that the stochastic control strategy has a region of stochastic dominance over the benchmark strategy and the put-write strategy over the entire loss side. Therefore, the goal to reduce the loss side risk has been achieved.



(a) Whole distribution

(b) Loss side

Figure 7.3: Training: cdf comparison

Figure 7.4 shows the allocation heatmap of the stochastic control. We observe that the greater the current wealth is, the less the stochastic control invests in the put-write strategy. Intuitively, when the portfolio is losing money, we should invest more wealth in the more aggressive put-write strategy in order to reach the target terminal wealth. On the other hand, if the current wealth is high, we can derisk the portfolio and invest more in the safer trend following strategy. Even though the feature time seems to have a smaller impact on the allocation decision, it is also observed that the closer it is to the terminal time T, the more wealth is invested in the put-write strategy. This can be understood instinctively that it is generally easier to reach the target terminal wealth when the time remaining is long. When there is not much time left to hit the target wealth, the stochastic control will be slightly more aggressive by allocating more wealth into the put-write strategy.

Figure 7.5 shows the stochastic control's distribution of allocation in the put-write strategy. We remark that the majority of allocation in the put-write strategy falls in the range of 15% to 70%. We also note that, on average, median of holding in put-write is less than 40%. Recall that the Sharpe ratio weight of the benchmark strategy is 50.7%. This implies that the stochastic control is quite distinct from the benchmark. The difference explains why the stochastic control is able to achieve a smaller loss side risk to some extent.

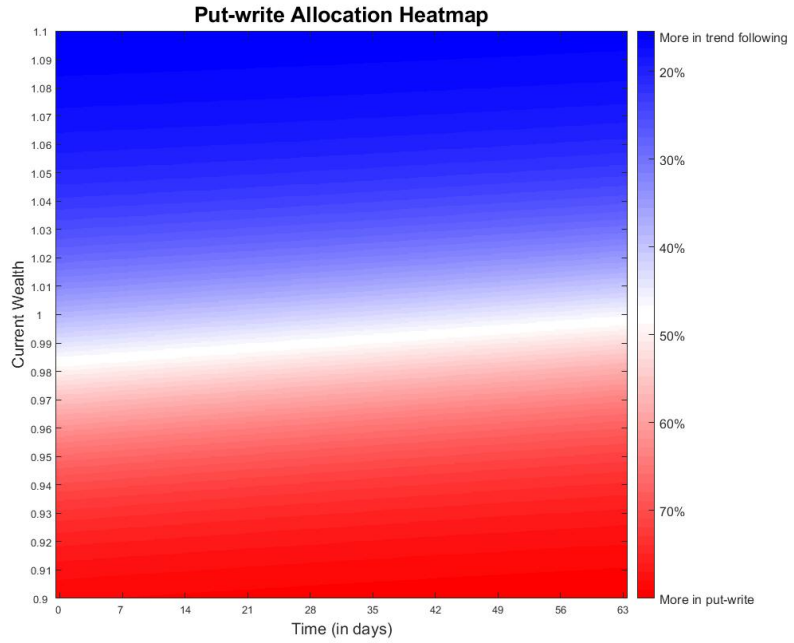


Figure 7.4: Stochastic control heatmap

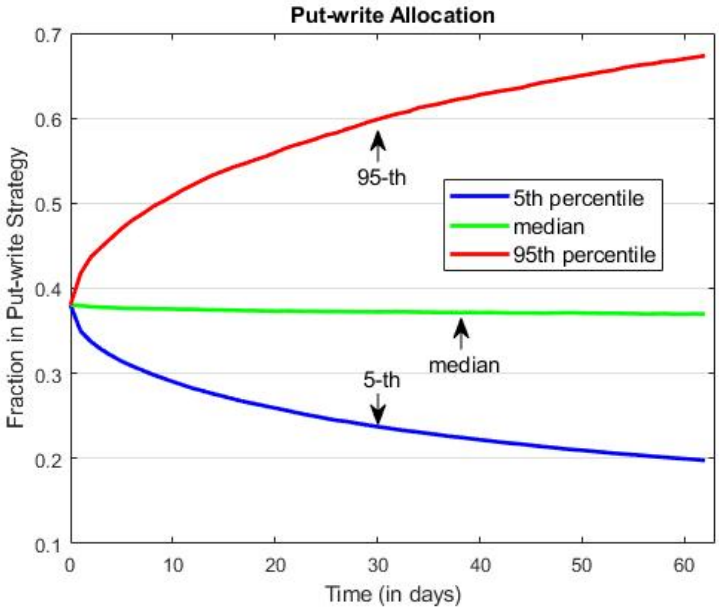


Figure 7.5: Stochastic control: distribution of allocation

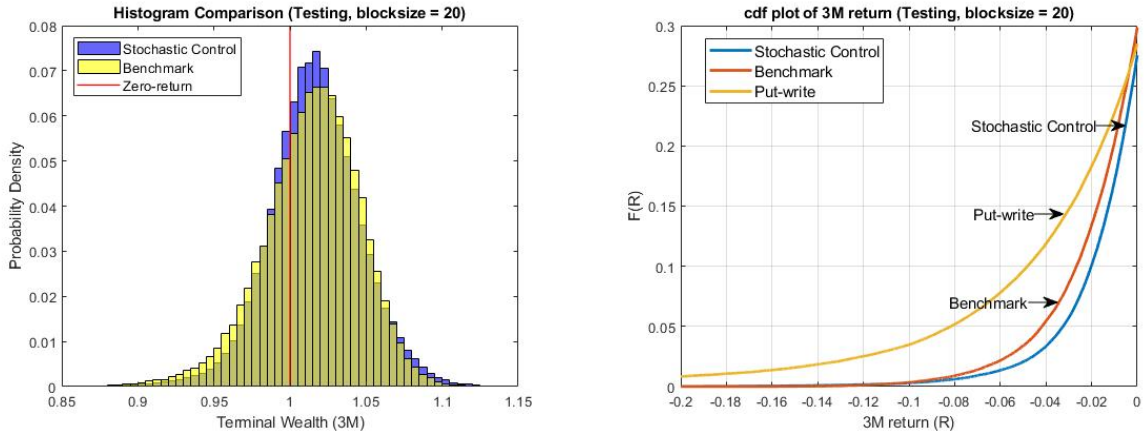
7.2 Testing on Bootstrap Resample Data

We validate the robustness of control strategies on bootstrap resample data sets with the expected blocksize = 1, 10 and 20 days. Recall that as shown in Chapter 6, changing the expected blocksize changes the distribution of returns. Table 7.2 presents the statistics for three month returns of the six strategies on bootstrap resample data set with the expected blocksize = 20 days. We have similar findings with the training results (see Table 7.1). The constant weight and deterministic controls are all on par with the benchmark strategy. While the stochastic control demonstrates an edge over the benchmark strategy in training, the advantage seems more substantial on this test set in terms of larger improvements on mean, standard deviation and 95%-CVaR. The put-write strategy still yields the best median return with the largest loss side risk. Figure 7.6 shows the probability density and cdf comparison on this test set. From the probability density histogram, we remark that the loss side of the stochastic control is covered by the benchmark, suggesting the stochastic control's smaller loss side risk. Then we look at the cdf comparison on the loss side. It can be observed that the stochastic control has a region of stochastic dominance over the put-write strategy and the benchmark. Table 7.3 shows the testing results on the bootstrap resample data set with the expected blocksize equals to 10 days. The test results are very similar to the training results (see Table 7.1).

Then we test the controls on the bootstrap resample data set with the expected blocksize equal to one day. The testing results are presented in Table 7.4. In general, the results are less ideal. Comparing to the benchmark, the stochastic control is less volatile, but has a smaller mean return and a larger 95%-CVaR. It is hard to tell which strategy has a smaller loss side risk from these statistics. Figure 7.7 presents the probability density and cdf comparison on this test set. From the density histogram, we observe that the stochastic control has a smaller probability density than the benchmark strategy on the majority of the loss side. However, the stochastic control has more extreme values on the loss side, which causes the CVaR of the stochastic control to be less than the benchmark. In the cdf comparison, we remark that the benchmark has stochastic dominance over stochastic control on $(-\infty, -6\%)$. On interval $(-6\%, 0\%)$, the stochastic control is stochastically dominant over the benchmark, but the region of stochastic dominance is relatively small compared to the training results and the other two test cases.

We believe the distortion of the serial correlation explains this observation. As discussed earlier in Chapter 6, we sample i.i.d. from historical return when the expected blocksize equal to one day. Consider a case where the stochastic control overweights the put-write strategy, because stochastic control expects it to outperform in the next several days. Nevertheless, the subsequent returns are sampled i.i.d. and it is possible that the

subsequent path is selected when the put-write strategy underperforms the trend following strategy. We expect that the trend following index incorporates some path-dependent market information into the index itself. However, the put-write index does not reserve path-dependent market information into its index construction. We do not believe that the serial correlation in the time series can be fully reflected by daily returns. With a larger expected blocksize, more serial correlation is preserved and these data sets reflects more realistic scenarios whereas the serial correlation is completely distorted when the expected blocksize equals to one day. From this point of view, choosing the expected blocksize equal to one day does not give a realisitc replication of the market return scenarios.



(a) Density comparison

(b) cdf comparison

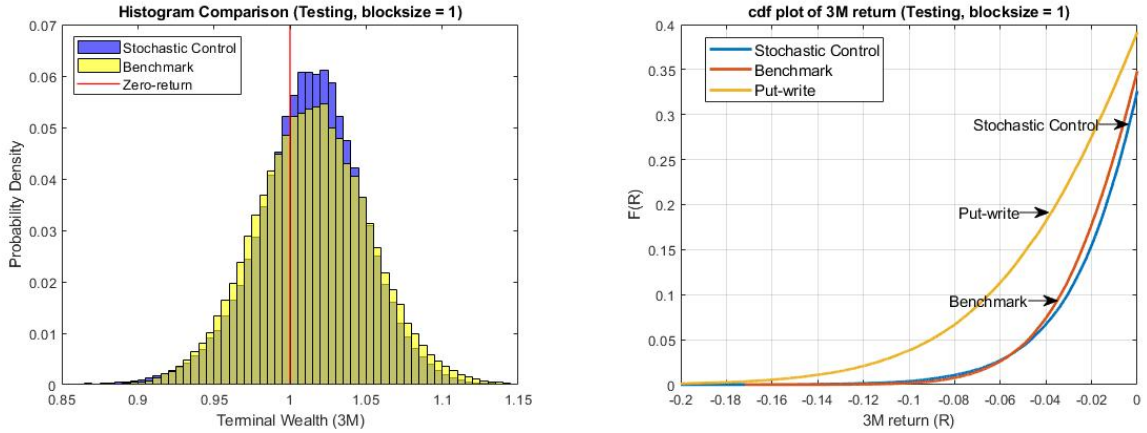
Figure 7.6: Testing Results on bootstrap resample data, expected blocksize = 20 days

Strategy	**Median**	**Mean**	**Standard Deviation**	**95%-CVaR**
Constant Control	1.6%	1.4%	3.2%	-6.3%
Deterministic Control	1.6%	1.4%	3.1%	-6.2%
Stochastic Control	**1.6%**	**1.6%**	**3.0%**	**-5.2%**
Benchmark	**1.6%**	**1.4%**	**3.2%**	**-6.3%**
Put-write	2.5%	1.6%	5.5%	-14.1%
Trend Following	1.0%	1.0%	4.1%	-7.5%

Table 7.2: Testing Results on bootstrap resample data, expected blocksize = 20 days

Strategy	Median	Mean	Standard Deviation	95%-CVaR
Constant Control	1.6%	1.4%	3.3%	-6.5%
Deterministic Control	1.6%	1.4%	3.3%	-6.4%
Stochastic Control	**1.6%**	**1.5%**	**3.2%**	**-5.8%**
Benchmark	**1.6%**	**1.4%**	**3.3%**	**-6.5%**
Put-write	2.4%	1.6%	5.6%	-13.5%
Trend Following	1.0%	1.0%	4.2%	-8.0%

Table 7.3: Testing Results on bootstrap resample data, expected blocksize = 10 days



(a) Density comparison

(b) cdf comparison

Figure 7.7: Testing Results on bootstrap resample data, expected blocksize = 1 day

Strategy	Median	Mean	Standard Deviation	95%-CVaR
Constant Control	1.4%	1.4%	3.7%	-6.5%
Deterministic Control	1.4%	1.4%	3.7%	-6.4%
Stochastic Control	**1.4%**	**1.3%**	**3.4%**	**-6.6%**
Benchmark	**1.4%**	**1.4%**	**3.7%**	**-6.5%**
Put-write	1.7%	1.6%	6.4%	-12.2%
Trend Following	1.0%	1.0%	4.0%	-7.4%

Table 7.4: Testing Results on bootstrap resample data, expected blocksize = 1 day

7.3 Testing on the Historical Data

We also test the stochastic control on the actual historical data paths. We note that the training data is on the bootstrapped data and the actual historic path is not in the training data set. We take the first 4347 daily returns ranging from January 2, 2002 to April 9, 2019. We divide 4347 data points into 69 investment periods, and each period is 63 days long, or three months equivalently. In other words, the test set consists of 69 historical paths, with each path lasting 63 days. We test the control strategies on these paths and then we stack the results of all investment periods to see the historical performance.

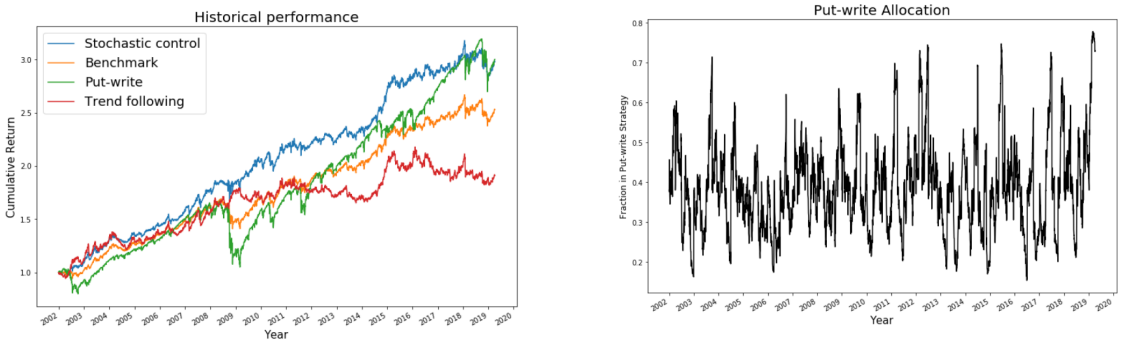
Table 7.5 shows the statistics for annualized returns as well as the maximum drawdown, which is a common method to measure downside risk in historical backtesting. Median, mean, and volatility are calculated using 4347 daily log returns, then scaled to annualized returns. Mean return and volatility reported in this table are used to calculate Sharpe ratios. The risk-free rate is assumed to be 2.5%. We do not report results for constant weight and deterministic controls due to their similarity to the benchmark. From the table, we note that the stochastic control has the best risk-adjusted return, measured by Sharpe ratio. Compared to the benchmark, the stochastic control has a similar level of volatility and maximum drawdown, but its mean return has outperformed the benchmark strategy by 0.9%. The put-write strategy gives the same level of mean return as the stochastic control, but comes with much a larger volatility and maximum drawdown, leading to a less attractive risk-adjusted return. The trend following strategy is the safer strategy with a smaller volatility and maximum drawdown, compared to put-write strategy. Nevertheless, the mean return is so low that trend following strategy has the lowest Sharpe ratio.

Strategy	**Median**	**Mean**	**Volatility**	**Sharpe ratio**	**Max. drawdown**
Stochastic Control	**10.2%**	**6.3%**	**7.9%**	**0.48**	**-13.7%**
Benchmark	**11.5%**	**5.4%**	**7.4%**	**0.39**	**-11%**
Put-write	11.5%	6.3%	12.8%	0.30	-37.1%
Trend Following	7.2%	3.8%	8.0%	0.16	-16.5%

Table 7.5: Testing results on the actual historical data paths (Annualized)

Figure 7.8a shows the historical performance of the four control strategies. We see that the stochastic control shows a smoother path while achieving the same level of return as the put-write strategy. The put-write strategy suffers substantial losses from major market crashes, while the stochastic control seems to be able to avoid the large drawdowns. The benchmark strategy also gives a stable historical performance, but is outperformed by the stochastic control. Figure 7.8b shows the stochastic control's allocation over the 18 years

period. Allocation in put-write ranges from 15% to 80%. Since we are interested in how stochastic control behaves in a bear market, we choose the period from September 2008 to December 2008 to further analyze the allocation decision made by the stochastic control and its relative performance.

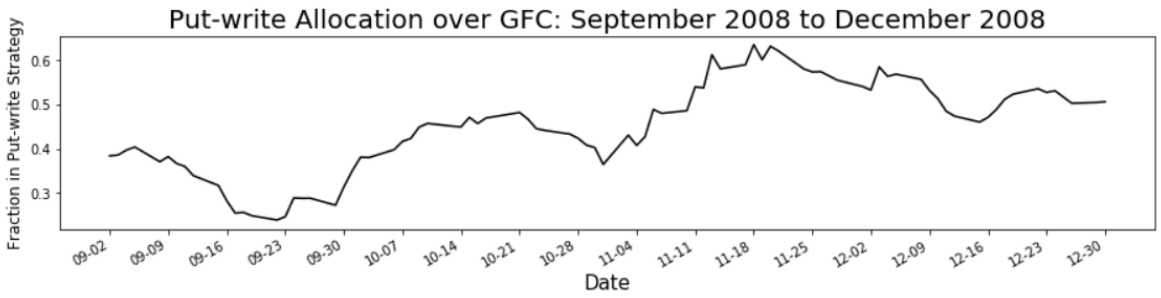


(a) Historical Performance

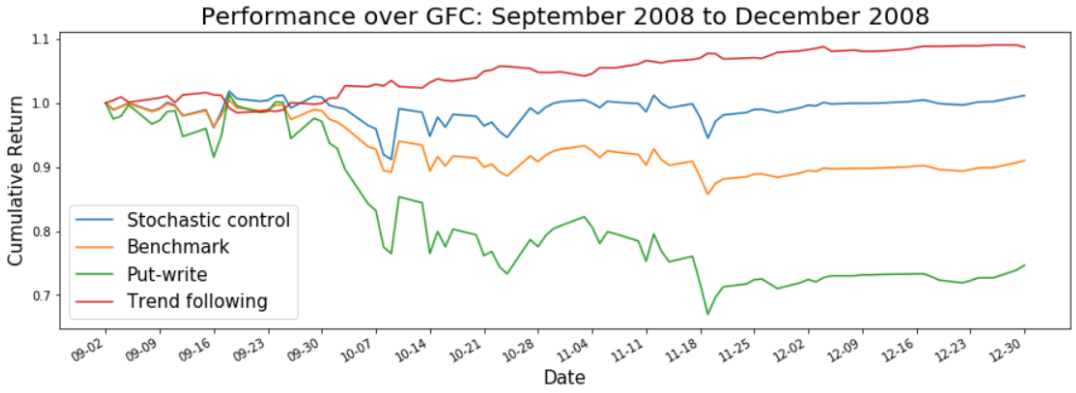
(b) Stochastic control allocation over time

Figure 7.8: Testing results on the actual historical data paths

Figure 7.9 presents the stochastic control's allocation decision and the relative performance of the four strategies in that period. From September to the mid November, the trend following strategy has a steady gain, while the put-write strategy loses more than 30%. We expect the stochastic control to underweight the put-write strategy in order to outperform the benchmark in this period. The put-write index drops since the mid September, so portfolio wealth has been shrinking ever since and more wealth is allocated into the put-write strategy. The stochastic control expects the put-write index to bounce back. We observe that the stochastic control increases its holding in the put-write strategy until it reaches its peak in the mid November 2008. In the mid November, the put-write strategy starts to climb back, and the stochastic control still overweights in the put-write strategy since the portfolio wealth remains low. In summary, when the put-write strategy underperforms the trend following strategy from September to mid November, the stochastic control underweights in the put-write strategy for majority of the time. And when the put-write strategy outperforms the trend following strategy from the mid November to the end of December, the stochastic control overweights in the put-write strategy. Such allocation decision allows the stochastic control to outperform the benchmark in a period when the put-write strategy loses more than 30%.



(a) Stochastic control allocation



(b) Relative performance

Figure 7.9: Testing results on the actual historical data paths (2008)

Chapter 8

Conclusion

In this research paper, we formulate the put-write and trend following investment problem as a multi-period asset allocation problem. We propose using a one-sided quadratic function to penalize the expected shortfall, if the terminal wealth fails to reach a target. We convert the constrained optimization problem (3.4) into an unconstrained optimization problem (3.9) with less variables by the use of a neural network. The neural network optimization framework makes it computationally efficient to solve the multi-period allocation problem with many sample paths and a multi-period investment horizon.

We consider three control strategies based on solving the unconstrained optimization problem (3.9): (i) a constant weight control, (ii) a deterministic control which is a function of time only, and (iii) a stochastic control which is a function of time and the current wealth. The benchmark portfolio is a constant weight strategy that is based on optimizing the Sharpe ratio. We constrain the median of the three strategies to be the same as the benchmark portfolio.

The constant weight and deterministic controls make similar allocation decisions with the benchmark strategy, so their performances are close to the benchmark strategy. The stochastic control is able to achieve a smaller loss side risk during training. We observe similarly promising test results on the bootstrap resample data sets with larger expected blocksizes. Test results are less ideal for the data set with a smaller expected blocksize, as we observe more extreme values at the left tail and smaller region of stochastic dominance.

We believe that this can be explained by the distortion of the serial correlation when the expected blocksize $\hat{b}$ equals one day. We also test the stochastic control on the historical data paths. Historical performance of the stochastic control achieves a similar cumulative return with the put-write strategy, and it gives a much smoother ride with smaller volatility

and drawdown. Overall, the performance of stochastic control is superior to the benchmark strategy, achieving a higher mean return, a smaller volatility and a smaller left tail risk.

We also investigate the impact of the expected blocksize on the distribution of the overall return generated from bootstrap resampling. We observe that as the expected blocksize decreases to one day, the three-month return distribution for the PUT Index changes from a negatively-skewed, long-tailed distribution to a normal distribution. We also note that resample data sets with larger expected blocksizes have more and bigger extreme values.

In addition, we compare four optimization techniques on this allocation problem, including (i) the trust region method, (ii) the gradient descent method, (iii) the mini-batch gradient descent method, and (iv) the ADAM method. We find that the trust region method is the slowest algorithm among the four and it is impractical when data size becomes large due to its long computation time. Both the gradient descent and mini-batch gradient descent methods face the dilemma where small learning rates lead to slow convergence and large learning rates cause the objective to oscillate around the minimum. Therefore, we use ADAM, an optimization method with adaptive learning rates to solve the optimization problem. Using this method, the objective function converges smoothly to a minimum within a short period of time.

References

[1] M. Abe and H. Nakayama. Deep learning for forecasting stock returns in the cross-section. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2018.

[2] G. Bakshi and N. Kapadia. Delta-hedged gains and the negative market volatility risk premium. *The Review of Financial Studies*, 16(2):527–566, 2003.

[3] N. Barberis, A. Shleifer, and R. Vishny. A model of investor sentiment. *Journal of Financial Economics*, 49(3):307–343, 1998.

[4] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[5] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 1995.

[6] S. Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[7] G. Chakravorty, A. Awasthi, B. Silva, and M. Singhal. Deep learning for global tactical asset allocation. *SSRN Electronic Journal*, 3242432, 2018.

[8] A. Clare, J. Seaton, P. Smith, and S. Thomas. The trend is our friend: Risk parity, momentum and trend following in global asset allocation. *Journal of Behavioral and Experimental Finance*, 9:63–80, 2016.

[9] P. Cogneau and V. Zakamouline. Block bootstrap methods and the choice of stocks for the long run. *Quantitative Finance*, 13(9):1443–1457, 2013.

[10] T. Coleman and Y. Li. An interior trust region approach for nonlinear minimization subject to bounds. *SIAM Journal on Optimization*, 6(2):418–445, 1996.

[11] M. Dai, Z. Yang, Q. Zhang, and Q. Zhu. Optimal trend following trading rules. *Mathematics of Operations Research*, 41(2):626–642, 2016.

[12] D.-M. Dang and P. Forsyth. Continuous time mean-variance optimal portfolio allocation under jump diffusion: An numerical impulse control approach. *Numerical Methods for Partial Differential Equations*, 30(2):664–698, 2014.

[13] H. Dichtl, W. Drobetz, and M. Wambach. Testing rebalancing strategies for stock-bond portfolios across different asset allocations. *Applied Economics*, 48(9):772–788, 2016.

[14] P. Forsyth. Multi-period mean CVAR asset allocation: Is it advantageous to be time consistent? *SSRN Electronic Journal*, 3340194, 2019.

[15] P. Forsyth and G. Labahn. ε-monotone Fourier methods for optimal stochastic control in finance. *Journal of Computational Finance*, 22(4), 2019.

[16] A. Frazzini. The disposition effect and underreaction to news. *The Journal of Finance*, 61(4):2017–2046, 2006.

[17] N. Garleanu, H. Pedersen, and A. Poteshman. Demand-based option pricing. *The Review of Financial Studies*, 22(10):4259–4299, 2008.

[18] A. Georgopoulou and J. Wang. The trend is your friend: Time-series momentum strategies across equity and commodity markets. *SSRN Electronic Journal*, 2618243, 2016.

[19] W. Goetzmann, D. Kim, and R. Shiller. Crash beliefs from investor surveys. *SSRN Electronic Journal*, 2750638, 2016.

[20] J. Grant. *The Great Metropolis*. Philadelphia, E.L. Carey & A. Hart, 1838.

[21] C. Green and S. Figlewski. Market risk and model risk for a financial institution writing options. *The Journal of Finance*, 54(4):1465–1499, 1999.

[22] S. Gu, B. Kelly, and D. Xiu. Empirical asset pricing via machine learning. *SSRN Electronic Journal*, 3159577, 2018.

[23] S. A. Hejazi and K. Jackson. A neural network approach to efficient valuation of large portfolios of variable annuities. *Insurance: Mathematics and Economics*, 70, 169-181, 2016.

[24] H. Hong and J. Stein. A unified theory of underreaction, momentum trading, and overreaction in asset markets. *The Journal of Finance*, 54(6):2143–2184, 1999.

[25] B. Hurst, Y. H. Ooi, and L. Pedersen. A century of evidence on trend-following investing. *The Journal of Portfolio Management*, 44 (1) 15-29, 2017.

[26] C. Jackwerth. Recovering risk aversion from option prices and realized returns. *The Review of Financial Studies*, 13(2):433–451, 2000.

[27] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

[28] Y. Li and P. Forsyth. A data-driven neural network approach to optimal asset allocation for target based defined contribution pension plans. *Insurance: Mathematics and Economics*, 86:189–204, 2019.

[29] J. Liew and B. Mayster. Forecasting ETFs with machine learning algorithms. *The Journal of Alternative Investments*, 20(3):58–78, 2017.

[30] H. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[31] F. Menoncin and E. Vigna. Mean–variance target-based optimisation for defined contribution pension. *Insurance: Mathematics and Economics*, 76, 172-184, 2017.

[32] T. Moskowitz, Y. H. Ooi, and H. Pedersen. Time series momentum. *Journal of Financial Economics*, 104(2):228–250, 2012.

[33] A. Patton, D. Politis, and H. White. Correction to "automatic block-length selection for the dependent bootstrap". *Econometric Reviews*, 28(4):372–375, 2009.

[34] D. Politis and H. White. Automatic block-length selection for the dependent bootstrap. *Econometric Reviews*, 23(1):53–70, 2004.

[35] J. Quirk and R. Saposnik. Admissibility and measurable utility functions. *The Review of Economic Studies*, 29(2):140–146, 1962.

[36] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[37] A. Saha, B. Malkiel, and A. Rinaudo. Option writing: Using VIX to improve returns. *Journal of Derivatives*, 26:38–49, 12 2018.

[38] J. Summa. Options sellers vs. buyers who wins? *Futures*, 32(4):52–52, 2003.

[39] J. Ungar and M. Moran. The cash-secured putwrite strategy and performance of related benchmark indexes. *The Journal of Alternative Investments*, 11(4):43–56, 2009.

[40] E. Vigna. On efficiency of mean–variance based portfolio selection in defined contribution pension schemes. *Quantitative Finance*, 14(2):237–258, 2014.

[41] H. Yao, Z. Yang, and P. Chen. Markowitz's mean–variance defined contribution pension fund management under inflation: A continuous-time model. *Insurance: Mathematics and Economics*, 53(3):851–863, 2013.

[42] X. Y. Zhou and D. Li. Continuous-time mean-variance portfolio selection: A stochastic LQ framework. *Applied Mathematics and Optimization*, 42:19–33, 2000.