Optimization for CVaR with Stochastic Gradient Descent

Duy Anh Le, Tat Dat Le, Gia Bao Phung

January 31, 2024

Abstract

We provide a way to find out what proportion to invest in different pools given an available environment using Stochastic Gradient Descent to min conditional value-at-risk (CVaR). Because it is practically impossible to clearly determine the profit function when investing, we used Monte Carlo Simulation combined with a few mathematical methods to solve the problem. By testing and examining the results of our algorithm with a few different inputs, we can conclude that it works quite well with what we have.

1 Introduction

Decentralized finance (DeFi) has emerged as a novel paradigm for financial services built on blockchain technology and smart contracts. By eliminating centralized intermediaries, DeFi protocols allow for transparent and accessible financial instruments. However, the complexity and volatility of DeFi markets present challenges in designing robust trading strategies. This is especially true for liquidity providers supplying assets into automated market maker (AMM) pools in exchange for liquidity tokens.

Optimizing rewards and risks is crucial for liquidity providers across multiple AMM pools. However, the complex dynamics between pooled assets and liquidity token returns make this a difficult portfolio optimization problem. Traditional metrics like variance are insufficient to capture tail risks and large losses. Furthermore, AMM pool simulations involve stochastic processes that render the overall optimization non-convex with multiple local optima.

Value at risk (VaR) and conditional value at risk (CVaR) have emerged as principled risk metrics to quantify downside losses. However, optimizing CVaR presents computational challenges due to reliance on Monte Carlo methods. This paper explores a simulation-based approach for optimizing CVaR of liquidity provider returns across multiple AMM pools through stochastic gradient descent. By minimizing CVaR, we aim to improve returns while controlling for tail risks and large losses. The volatile and complex nature of DeFi highlights the importance of CVaR optimization for liquidity provider risk management. Our computational framework combining AMM pool simulations and gradient-based optimization provides a tool for addressing these real-world challenges in DeFi finance.

1.1 Related Work

As an illustration of coherent risk measurements Artzner et al. [1999] in portfolio optimization, CVaR was presented by Rockafellar et al. [2000]. Since then, machine learning has effectively incorporated CVaR. SVM algorithms were examined by Gotoh and Takeda [2016] from the perspective of CVaR optimization. In reinforcement learning, Chow and Ghavamzadeh [2014], Chow et al. [2015], used CVaR optimization. Numerous authors investigated submodular maximization more generally and CVaR optimization in influence maximization. Ohsaka and Yoshida (2017), Maehara (2015), and Wilder (2018).

From the above research papers and the most recent paper with what we used for this problem is on Statistical Learning with Conditional Value at Risk by Tasuku Soma [2020]. With what has been developed from the paper, we have combined it with Monte Carlo Simulator to be able to estimate things that are difficult or almost impossible to accurately determine in reality to get closer to an algorithm that can apply to reality.

2 Methods

2.1 Monte Carlo Simulation

Conditional value at risk (CVaR) optimization relies heavily on Monte Carlo simulation methods. Monte Carlo simulation involves using random sampling and statistical modeling to numerically evaluate complex systems that have stochastic dynamics. For this paper, we employ Monte Carlo simulation to model the complex processes governing asset reserves and prices within automated market maker (AMM) pools. By simulating many sample paths for how AMM pools evolve under different scenarios, we can estimate the distribution of liquidity provider returns. This return distribution is then used to numerically compute CVaR as a risk metric that quantifies extreme losses. The gradient of the estimated CVaR concerning liquidity allocation parameters is also computed via finite differences using the simulations. Thus, Monte Carlo methods allow us to integrate the stochastic AMM processes into an overall CVaR optimization framework. However, generating sufficiently many sample paths is computationally intensive. There is also variance in CVaR estimates and gradients. Our methodology employs variance reduction techniques including control variates and common random numbers across sample paths. In summary, coupling Monte Carlo simulation with stochastic gradient descent provides a mechanism for CVaR optimization within complex, stochastic environments like decentralized finance.

2.2 Finite differences

Optimizing conditional value at risk (CVaR) requires computing gradients for decision variables like liquidity allocation parameters. However, the dependence of CVaR on these parameters is complex and not available in closed form. Therefore, we employ finite differences to numerically estimate the gradients. Finite differences approximate derivatives by evaluating the objective function at slightly perturbed parameter values. In particular, we use symmetric finite differences which perturb each parameter both above and below its current value. The gradient for a parameter is then approximated by the centered difference quotient. While simple to implement, a key challenge with finite differences is selecting the perturbation size. Too small and the differences are swamped by noise. Too large and approximation error dominates. We select the perturbation magnitude by balancing these considerations. The finite difference gradients are then used within the stochastic gradient descent algorithm to update the parameters. However, the gradient estimates can be noisy, hampering convergence. Regularization, local averaging, and iteration-dependent perturbations are useful techniques to help address this challenge. In summary, finite differences provide a mechanism to approximate gradients for CVaR optimization when analytical derivatives are unavailable, as is the complex case, of simulation-based AMM modeling.

2.3 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is an optimization algorithm commonly used to minimize objective functions in machine learning models. Unlike batch gradient descent, which computes the gradient of the loss function over the entire training dataset, SGD computes the gradient on a single training example or small batch of examples at a time. This allows SGD to start making progress right away, rather than having to process the full dataset on each iteration.

The algorithm proceeds as follows:

Algorithm Smoothed Stochastic Gradient Descent for CVaR

Require x_0 and the number of iterates T

1: Initialize θ and $\sum \theta = 1$
2: Set $\eta \leftarrow 0.05$
3: **for** t = 1,...,T do
4: $g_t \in \partial_\theta \tilde{f}_\varepsilon(\theta)$
5: Update $x_{t+1} = proj_K(x_t - _t)$
6: **return** $\theta = \frac{1}{T}\sum_{t=1}^{T} \theta_t$

By sampling batches randomly at each iteration, SGD ends up oscillating toward the global minimum over many iterations. The randomness helps SGD escape local minima and plateaus in the loss landscape. The learning rate η determines the size of the steps taken in the direction of the steepest

descent. We started finding the optimal η by randomly selecting an η range and running Stochastic Gradient Descent at both ends of the eta range. Once we have the results, based on the CVaR we found at both ends, we will gradually narrow the gap until we find the optimal learning rate to help CVaR converge in the shortest number of steps. After each time we find a θ with appropriate weights, we will project that θ into the feasible set to ensure the ratio of the θ always remains equal to 1.

In our experiments, we utilize SGD to optimize the parameters of our model. We sample batches of size T = 1000 from the training set. The learning rate is initially set to $\eta = 0.08$. We track the loss on a held-out validation set to detect convergence and avoid overfitting. Our results show SGD is effective at minimizing the training loss and improving validation accuracy over hundreds of epochs. The randomness and noise of SGD act as a regularizer to prevent overfitting. In future work, we plan to explore adaptive variants of SGD like AdaGrad and Adam.

2.4 Penalty Function

With the condition that minimizing CVaR is the probability that r_T is greater than $\zeta = 0.05$ is greater than q = 0.8, we used the penalty function to turn the original problem from a constrained optimization problem into an unconstrained optimization problem.
penalty function: $p(x) = \sum_{i=1}^{m}(max(0, g_i(x))^2$ where $g_i(x)$ is the constraint from the original problem. After we have defined the penalty function for the problem, we need to add $p(x)$ to the objective function because we are trying to minimize CVaR, the the next step will start charging a penalty for violating them.

3 Conclusion

In this work, we have presented a method for optimizing portfolio allocation across multiple automated market maker (AMM) pools using stochastic gradient descent. Our approach involves estimating the gradient of the expected log return concerning the asset allocation parameters via Monte Carlo simulation. This allows us to incrementally update the portfolio weights to maximize returns while minimizing risk.

Specifically, we modeled an environment with multiple Constant Product AMM pools, each with different liquidity conditions. We derived an upper bound on the Lipschitz constant of the expected return function, which allowed us to set an appropriate learning rate for SGD. At each iteration, we estimated the gradient by randomly perturbing the asset allocation and measuring the difference in simulated log returns. The gradient estimate was then used to update the portfolio weights.

Our experiments demonstrate that SGD can effectively optimize portfolio allocation in this setting, converging to weights that maximize expected log returns while reducing risk as measured by CVaR. The learned asset allocation significantly outperforms a uniform allocation baseline. The stochastic nature of the gradient estimation acts as a regularizer to prevent overfitting.

This work has several limitations that provide avenues for future investigation. First, we assumed a simple model for the dynamics of the AMM pools with fixed liquidity. Extending the approach to more complex environments with fluctuating liquidity pools would be worthwhile. Second, other stochastic optimization algorithms like Adam or natural gradient descent may provide faster convergence. Finally, explicit regularization of the portfolio weights could further reduce risk.

Overall, we have shown SGD combined with Monte Carlo simulation to be an effective methodology for optimal portfolio allocation across decentralized AMM exchanges. This work contributes to the growing literature on DeFi portfolio optimization and provides a starting point for further research at the intersection of machine learning and blockchain technologies. The methods developed here could ultimately be applied to real-world trading and yield farming strategies.

References

[1] Tasuku Soma, Yuichi Yoshida. Statistical Learning with Conditional Value at Risk, 2020

[2] Robert M.Freund. Penalty and Barrier Methods for Constrained Optimization, 2004

[3] R Tyrell Rockafellar, Stanislav Uryasev, et al. Optimization of conditional value-at-risk. Journal of Risk, 2000