STAT 4830
Slides Draft 2 (Week 6)

Portfolio Refinement Through Iterative Sequential Modeling (PRISM)

Team PRISM:
Dhruv Gupta, Aiden Lee, Frank Ma,
Kelly Wang, Didrik Wiig-Andersen

Wharton
University of Pennsylvania

Our Problem

Goal:
Optimize a daily portfolio of assets to achieve high risk-adjusted returns while respecting certain constraints on . . .

leverage, *drawdown*, and *volatility*.

Specifically, our project focuses on maximizing a combination of financial risk metrics and ratios while limiting maximum drawdown over a given historical period.

Why This Matters

- Traditional mean-variance optimization oversimplifies risk by only using variance.
- Real-world portfolios must manage multiple risk dimensions (drawdown, volatility) and practical constraints (leverage, short selling) at the same time.
- Variability in the real-world and extreme events (like COVID) don't follow a nice and familiar distribution, so due to their unpredictability, traditional trading strategies have different tolerance levels to these unexpected events.
- By addressing these complexities, we aim to create a more realistic and robust decision-making strategy for creating a dynamic portfolio.

Success Metric

What the fitness score consist of

- Sortino Ratio: $\frac{\mathbb{E}[R_p - R_f]}{\sigma_p}$
  - Where R_p = Return of the portfolio
  - R_f = Risk-free rate
  - σ_d = Standard deviation of negative asset returns (i.e., downside deviation where returns below a minimum acceptable return or MAR).
- Risk Metrics - Maximum Drawdown reduction, controlled volatility
- Factor Exposure - this refers to the sensitivity of an investment, portfolio, or asset to specific risk factors or systematic drivers of returns
- Portfolio Constraints - we ask the model to minimize transaction costs, minimize the change in our stock positions between days, and try to maintain stable returns over time

Constraints

- **Leverage**: May exceed 1 but within a specified maximum (e.g., 1.5–2.0)
- **Drawdown**: Must remain below a specified percentage (e.g., 20% max drawdown).
- **Data**: Historical daily/weekly returns for selected assets (10–30).

Data Requirements

- Daily price data from YFinance.
- Sufficient history to handle training and validation. The goal is to be able to test our model against extreme events like the 2008 Financial Crisis and COVID.

Potential Pitfalls

- Overfitting to historical data (backtest bias).
- Incorrect handling of missing data or survivorship bias.
- High computational costs if too many assets or constraints are added.

Technical Approach

Mathematical Formulation:
Let w_i denote the weight of asset i in the portfolio. We define the portfolio return as

$$R_P = \sum_i w_i \cdot R_i$$

The portfolio volatility σ_p is computed as the annualized standard deviation of the daily portfolio returns, and the maximum drawdown of the portfolio, hereby abbreviated to $MPP(p)$, is computed from the cumulative returns. Given this, our objective function is

$$\max \quad \alpha \cdot \frac{\mathbb{E}[R_P - R_f]}{\sigma_d} + \beta \cdot (-MDD(p)) - \lambda \sum_i |w_i - w_i^{\text{prev}}|$$

Definitions:

- R_f is the risk-free rate, so that $\frac{\mathbb{E}[R_p - R_f]}{\sigma_d}$ represents the Sortino ratio.
- α scales the impact of the Sharpe ratio.
- β scales the impact of the drawdown term.
- λ is the transaction cost penalty, which penalizes large changes in portfolio weights between periods.
- w_i^{prev} denotes the weight of asset i in the previous period.

Algorithm & PyTorch Strategy

- Represents weights $\mathbf{w}$ as a PyTorch tensor.
- Compute portfolio returns and risk measures (volatility, drawdown) within the computational graph.
- Use gradient-based methods (e.g., Adam, LBFGS) to optimize-objective (because PyTorch minimizes by default).

Validation Methods

- **In-Sample Optimization:**
  Train on a subset of historical data.
- **Out-of-Sample Backtest:**
  Test on later data (walk-forward or simple split).
- **Validation:**
  Compare results to a baseline (e.g., equal weights).

Optimization Strategy

PyTorch-Based Approach:

- Represent weights as tensors.
- Compute portfolio returns, volatility, and drawdown within PyTorch.
- Use gradient-based methods (Adam, LBFGS) for optimization.

Initial Results

Evidence of Working Implementation

- Basic Test: A small 7-asset dataset was loaded into our PyTorch pipeline.
  Companies: Tesla, Google, Microsoft, Amazon, Apple, Meta, NVIDIA
- Hyperparameters: We experimented with random values and chose the set that resulted in the best model specifications.

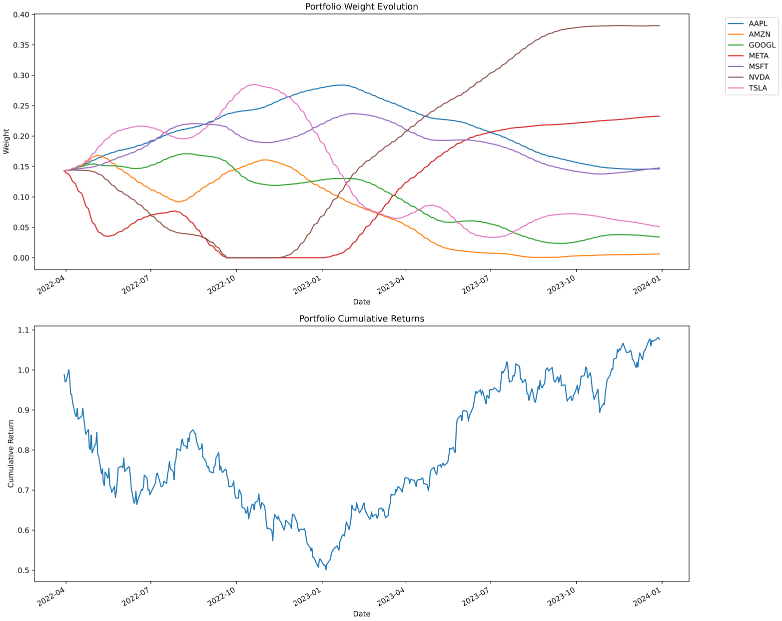

Figure 1: Our model weights evolution along with their cumulative returns.

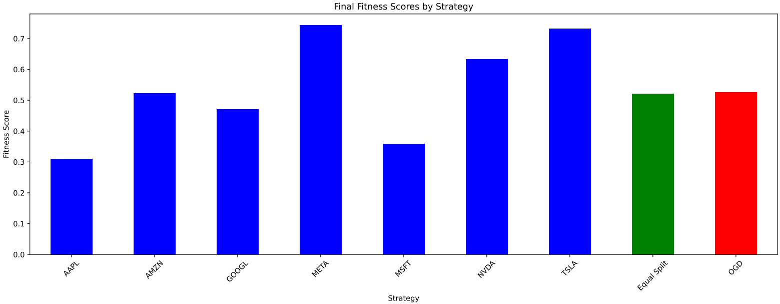

Figure 2: Fitness scores for the original OGD algorithm.

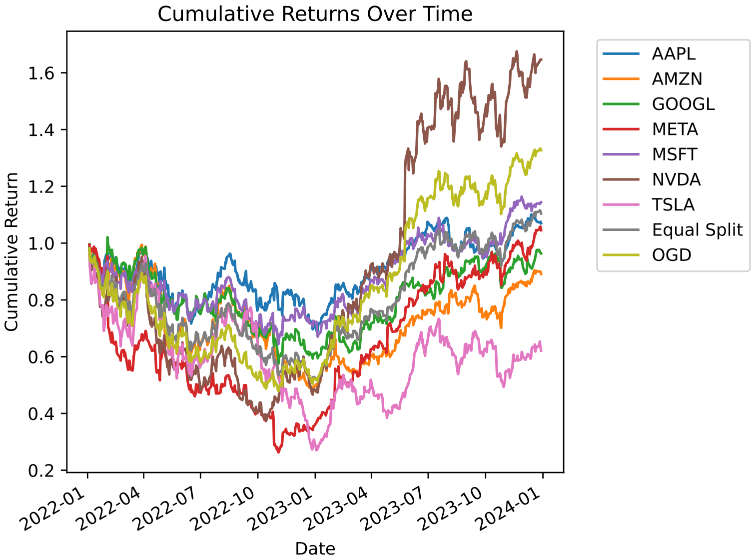

Figure 3: Compare the cumulative returns of our strategy to the other portfolios.

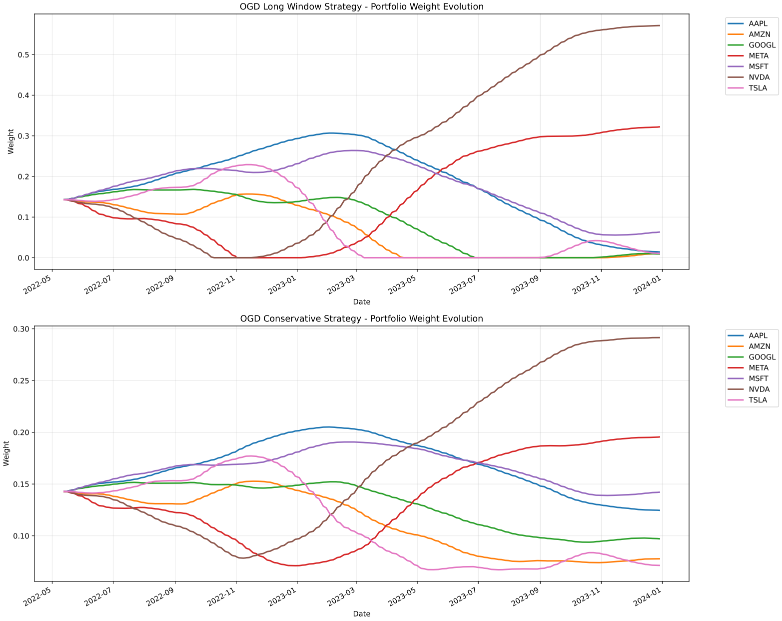

Figure 4: Comparing the weight evolution for different strategies.

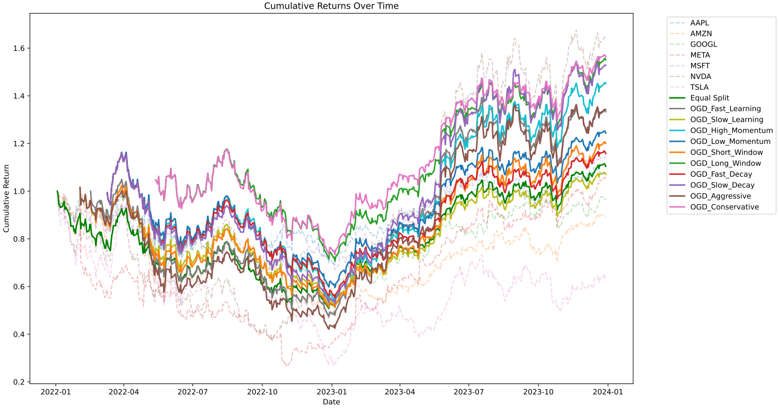

Figure 5: Cumulative returns for all variations of strategy using OGD.

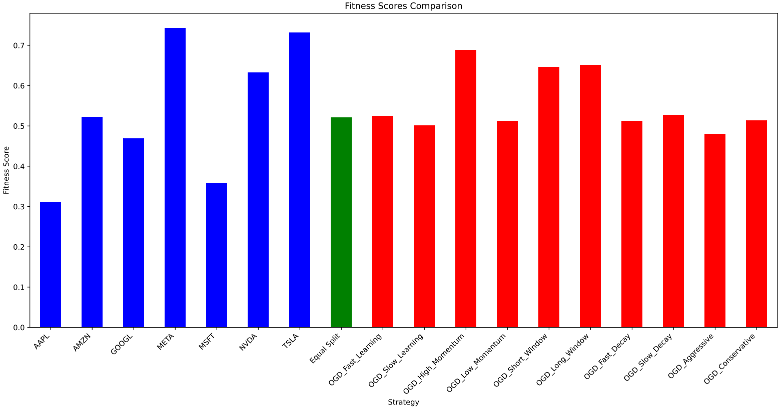

Figure 6: Comparing the fitness scores across different strategies.

Methodology

Online Gradient Descent

- Our testing only considers the stock market in 2022-2024 and produced weights that resulted in a portfolio that was only outperformed by a portfolio that only contained META and only contained NVDA.
- With our updated fitness function using a working OGD algorithm, the score now outperforms all single investment stocks except for NVDA.

Next Steps

Planned Improvements:

1. Expand Data Universe
   - Increase the number of assets considered in our portfolio to the complete S&P500, ensuring robust coverage of different sectors.
   - Acquire a longer historical window and consider testing our model against time periods where there where sudden shocks to the market due to extreme events.
   - We will also consider the question: How long of historical window matters?
2. Refine Constraints
   - Enforce leverage up to 1.5, short selling up to 30% of portfolio.
   - Evaluate how these constraints interact with drawdown penalty
   - Integrate more advanced risk measures like conditional value-at-risk
   - Review the literature on alternative risk measures we should consider incorporating.

Next Steps (cont.)

3. Rolling Optimization
   - Implement a time-series approach to re-balance daily/monthly/quarterly.
4. Short Selling
   - Allow the ability, up to a certain threshold, to bet against certain assets.
5. Transaction Costs
   - Add a penalty for changing weights significantly between re-balances.
6. Advanced Validation
   - Perform a walk-forward validation to reduce over-fitting risk.
   - Compare with multiple baselines (index funds, risk-parity strategy).
   - Test our model with real-time data implementation

Next Steps (cont.)

7. Other Methods to Consider
   - We need to account for cases when the weights we put on our stocks are too sparse. There should be a mechancism to add a penalty for the sum of the weights w_i. We need more flexibility in our objective function to improve the way we account for the risk in the stock market.
   - Online Multiplicative Weights - look into using multiplicative experts.
   - Online Mirror Descent - think of stocks at the sector level. Then, the goal is to maximize entropy $\sum_i w_i \cdot \log(w_i)$. Look into the Bregman divergence.

Self-Critique

Strengths:

- Effective implementation of multi-objective optimization.
- Demonstrated initial backtesting results.

Areas for Improvement:

- Optimize objective function weights.
- Diversify asset selection.
- Test impact of different window sizes.
- We are influencing our portfolio with hindsight bias due to the fact that we chose the 7 firms that it can invest in. The choices we made depend on our knowledge of the past, so it is necessary to remove our influence on the model.

Conclusion

Key Takeaways:

- Multi-objective portfolio optimization is complex but achievable.
- PyTorch provides flexibility but requires careful constraint handling.
- Future work will refine constraints, expand datasets, and validate models.