AMATH 482 Homework 5

Trevor Ruggeri

Due: March 20, 2025

This report discusses the development and implementation of a Fully Connected Neural Network (FCN) to optimize portfolio weights for a selected basket of 20 stocks, aiming to maximize the Sharpe ratio on a test dataset. The project combines the principles of portfolio optimization with machine learning techniques, leveraging advanced loss functions to achieve the objective. Key insights are drawn from computational experiments, where adjustments in hyperparameters—such as learning rate, batch size, and epochs—were critical in stabilizing the training process and enhancing the model's generalization performance.

1 Introduction and Overview

Portfolio optimization is a fundamental problem in finance, focusing on allocating weights to assets to maximize returns while minimizing risk. Among several performance metrics, the Sharpe ratio is widely used for its ability to quantify risk-adjusted returns. This project aimed to utilize an FCN trained on historical stock data, enhanced with technical indicators, to generate optimal portfolio weights that maximize the Sharpe ratio on the test set. The study investigates the behavior of the model under different hyperparameter configurations, analyzing their effects on both training stability and test performance.

2 Theoretical Background

2.1 Portfolio Optimization and the Sharpe Ratio

Portfolio optimization aims to allocate weights w_i to a set of n assets in order to balance expected returns and associated risk. One of the most widely used performance metrics for this purpose is the *Sharpe Ratio*, which measures risk-adjusted returns. It is defined as:

$$\text{Sharpe Ratio} = \frac{\mathbb{E}[R_p] - R_f}{\sigma_p},$$

where:

- $\mathbb{E}[R_p]$ is the *expected portfolio return*, calculated as $\mathbb{E}[R_p] = \sum_{i=1}^{n} w_i \mu_i$, where w_i are the portfolio weights, and μ_i are the expected returns of the individual assets.
- R_f is the *risk-free rate*, representing the return on a theoretically risk-free investment. Assumed to be 0.01.
- σ_p is the *portfolio standard deviation*, representing the risk of the portfolio, computed as $\sigma_p = \sqrt{w^T \Sigma w}$, where Σ is the covariance matrix of asset returns.

The goal of portfolio optimization is to maximize the Sharpe Ratio by finding the optimal set of weights w such that the portfolio achieves the highest possible risk-adjusted return.

2.2 Sharpe Ratio Loss in Machine Learning

The Sharpe Ratio presents a nonlinear optimization problem that aligns well with the capabilities of machine learning models. A Fully Connected Neural Network (FCN) was trained to predict portfolio weights w that maximize the Sharpe Ratio. To achieve this, a custom loss function was developed to dynamically compute the Sharpe Ratio for each batch of data during training:

$$\text{Loss} = -\frac{\mathbb{E}[R_p] - R_f}{\sigma_p}.$$

The custom loss function incorporates:

- **Portfolio Return**: Computed as $\mathbb{E}[R_p] = \sum_{i=1}^{n} w_i \mu_i$, where μ_i are the expected returns of the assets.
- **Portfolio Variance**: Computed as $\sigma_p^2 = w^T \Sigma w$, where Σ is the covariance matrix of asset returns.
- **Batch-Level Dynamics**: The loss function dynamically calculates expected returns and variance for each batch of data, ensuring adaptability to changes in the data.

By minimizing the negative Sharpe Ratio during training, the FCN learns to output weights w that maximize the Sharpe Ratio on the validation and test datasets.

3 Algorithm Implementation and Development

3.1 Fully Connected Neural Network (FCN) Architecture

To predict portfolio weights that maximize the Sharpe Ratio, a Fully Connected Neural Network (FCN) was constructed. The architecture consists of three primary layers:

- **Input Layer**: Accepts a feature matrix derived from technical indicators such as moving averages, RSI, Bollinger Bands, and others.
- **Hidden Layers**: Three hidden layers with 512, 246, and 128 neurons, respectively. Each layer applies dropout regularization to mitigate overfitting, followed by the ReLU activation function to introduce non-linearity.
- **Output Layer**: Outputs a vector representing the portfolio weights for all assets, normalized to sum to 1 using the Softmax activation function.

This design ensures the network can effectively model the relationship between input features and optimal portfolio weights, while maintaining numerical stability.

3.2 Sharpe Ratio Loss Function

A custom loss function was implemented to directly optimize the Sharpe Ratio:

- **Portfolio Return**: Calculated as the weighted sum of rolling expected returns for a given batch of assets.
- **Portfolio Risk**: Computed as the standard deviation of portfolio returns, derived from the quadratic form $w^T \Sigma w$, where Σ is the rolling covariance matrix.
- **Sharpe Ratio**: Defined as $\frac{\mathbb{E}[R_p] - R_f}{\sigma_p}$, where R_f is the risk-free rate. The negative Sharpe Ratio was minimized during training to indirectly maximize risk-adjusted returns.

By incorporating batch-specific rolling metrics, the loss function dynamically adapted to time-varying market conditions.

3.3 Data Preprocessing and Pipeline Integration

The dataset consisted of historical stock prices and technical indicators for 20 selected assets. Rolling expected returns and covariance matrices were computed over a 60-day window and aligned with the feature matrix. The data was divided into training, validation, and test sets, and processed into batches for efficient computation using PyTorch's DataLoader.

3.4 Hyperparameter Configuration

The model's learning process was fine-tuned by experimenting with hyperparameters, including:

- **Learning Rate**: Reduced from an initial value of 0.1 to 1×10^{-5} to enhance stability during training.
- **Batch Size**: Increased from 32 to 1024 (training) and 256 (testing) to reduce gradient noise and improve convergence.
- **Epochs**: Extended from 50 to 100, allowing the model sufficient time to converge and improve its ability to generalize to the test set.

These adjustments resulted in smoother loss curves and improved Sharpe Ratio optimization on the test set.

4 Computational Results

4.1 Overview of Model Configurations

Seven models were trained to optimize portfolio weights for maximizing the Sharpe ratio. Each model configuration adjusted hyperparameters such as learning rate, batch size, and the number of epochs. Below, we analyze their training, validation, and test losses. Figures depicting training and validation losses for each configuration are provided.

4.2 Model Configurations and Results

The configurations of the models are summarized as follows:

- **Model 1**: Initial configuration with batch size of 32 (training and testing), learning rate 0.1, and 50 epochs. Test loss: 0.2284.
- **Model 2**: Same as Model 1, but with reduced learning rate 10^{-4}. Test loss: 0.3610.
- **Model 3**: Same as Model 2, but with further reduced learning rate 10^{-5}. Test loss: 0.3610.
- **Model 4**: Learning rate 10^{-5}, epochs 50, training batch size 256, and testing batch size 128. Test loss: 0.2285.
- **Model 5**: Same as Model 4, but training batch size increased to 512 and testing batch size to 256. Test loss: 0.2291.
- **Model 6**: Same as Model 5, but training batch size increased further to 1024. Test loss: 0.2326.
- **Model 7**: Same as Model 6, but number of epochs increased to 100. Test loss: 0.2292.

4.3 Analysis of Training and Validation Losses

Figures 1 through 7 illustrate the training and validation loss curves for each model. These results highlight several trends:

- As the learning rate was reduced from 0.1 to 10^{-5} (Models 1 through 3), training and validation losses became significantly more stable, avoiding oscillations and high plateaus observed in the initial configuration.

- Increasing the batch sizes (Models 4 through 6) improved convergence, with training losses becoming smoother and less prone to stagnation or noise.
- Extending the number of epochs to 100 (Model 7) further refined loss curves and stabilized validation performance.

4.4 Test Loss Analysis

Despite dramatic improvements in training and validation losses across models, the test loss remained relatively unchanged, fluctuating between 0.2284 and 0.3610. This indicates that:

- The changes in hyperparameters primarily affected the model's ability to fit the training and validation data, without significantly enhancing its generalization to unseen test data.
- The test loss performance suggests that factors beyond hyperparameters—such as rolling expected returns and covariance matrices—could provide better test set generalization.

4.5 Figures

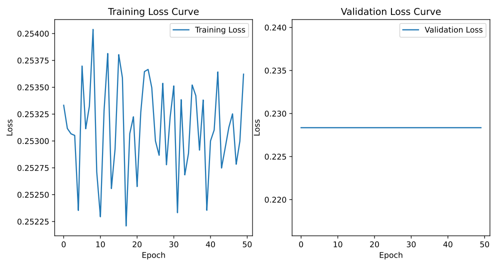

Figure 1: Model 1 Training and Validation Loss Curves

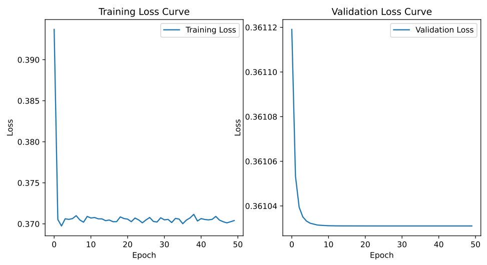

Figure 2: Model 2 Training and Validation Loss Curves

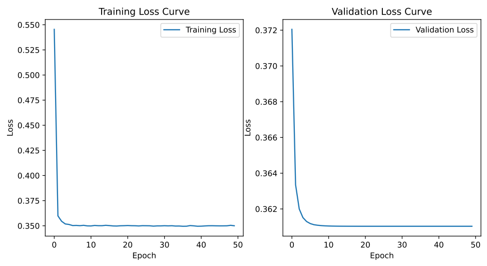

Figure 3: Model 3 Training and Validation Loss Curves

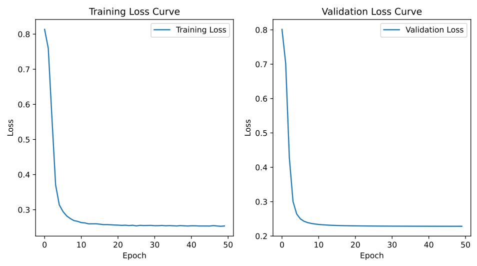

Figure 4: Model 4 Training and Validation Loss Curves

5 Summary and Conclusions

5.1 Summary of Findings

This study aimed to optimize portfolio weights for a set of 20 stocks by maximizing the Sharpe ratio using a Fully Connected Neural Network (FCN) and a custom loss function. Through iterative hyperparameter tuning and architectural refinements, the following key observations were made:

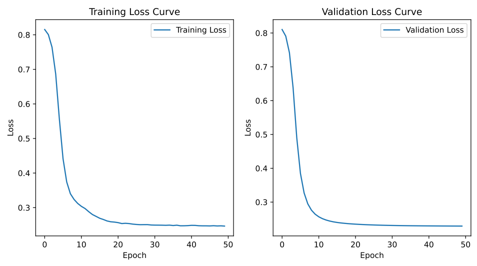

Figure 5: Model 5 Training and Validation Loss Curves

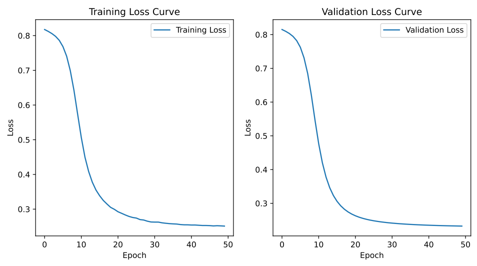

Figure 6: Model 6 Training and Validation Loss Curves

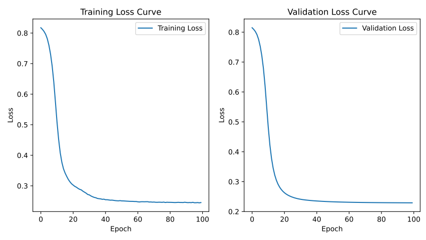

Figure 7: Model 7 Training and Validation Loss Curves

- Training and validation losses showed significant improvement across models, demonstrating increased stability and convergence as hyperparameters were adjusted.
- Lower learning rates (10^{-5}) minimized oscillations in training loss and improved the stochastic behavior of gradient updates.
- Larger batch sizes (up to 1024 for training) reduced noise in gradient estimates, leading to smoother loss curves and preventing premature plateaus.
- Extending the training epochs from 50 to 100 further stabilized loss curves and slightly improved overall model performance.

Despite these advancements, the test loss remained relatively consistent across configurations, suggesting that further improvements in generalization may require adjustments beyond hyperparameters.

5.2 Conclusions

The iterative training process successfully minimized the Sharpe ratio test loss, demonstrating the effectiveness of the FCN architecture and Sharpe Ratio loss function for portfolio optimization. While hyperparameter tuning improved training and validation performance, the test loss plateau indicates the need for enhanced strategies to bridge the gap between model fitting and generalization.

5.3 Possible Next Steps

To further refine the model and improve test set performance, the following avenues are recommended:

- **Learning Rate Schedulers**: Dynamically adjust the learning rate during training to improve convergence, especially in later epochs.
- **Gradient Clipping**: Implement gradient clipping to prevent extreme updates and ensure numerical stability.

- **Rolling Metrics Integration**: Use dynamically computed rolling expected returns and covariance matrices within the Sharpe Ratio loss function to better capture time-varying relationships in the data.

These approaches have the potential to further optimize the model and enhance its generalization to unseen datasets, paving the way for more robust portfolio optimization.

6 Acknowledgements

We express our gratitude to the developers and contributors of PyTorch, whose framework served as the backbone for the neural network implementation in this study. We also acknowledge Yahoo Finance for providing the historical stock price data used for analysis and training. Additionally, we thank researchers and developers in the fields of portfolio optimization and machine learning for the foundational methodologies that inspired the techniques applied here. Their contributions have enabled advancements in finance and computational optimization.

7 References

References

[1] Sharpe, W. F. (1966). "Mutual Fund Performance." *Journal of Business*, 39(1), 119–138.

[2] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

[3] Kingma, D. P., & Ba, J. (2014). "Adam: A Method for Stochastic Optimization." arXiv:1412.6980.

[4] Yahoo Finance API Documentation. `https://finance.yahoo.com/`

[5] Markowitz, H. M. (1952). "Portfolio Selection." *The Journal of Finance*, 7(1), 77–91.

[6] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., *et al.* (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library." arXiv:1912.01703.