Optimizing portfolio weights with machine learning

Ziqi Yuan* Jiaqi Zhang†

July 29, 2025

ABSTRACT

A neural network that directly optimizes portfolio weights outperforms the conventional two-step approach of forecasting returns and then sorting stocks, achieving an out-of-sample Sharpe ratio of 4.06 from 1987 to 2021. Its output layer assigns a weight to each stock, and the loss function is defined directly on the portfolio's return, with an additional quadratic return term to penalize volatility. The model can also optimize net-of-cost returns by incorporating transaction-cost assumptions into the loss function, yielding a portfolio with lower turnover and more liquid stocks.

*Questrom School of Business, Boston University. zqyuan@bu.edu.
†Seidner Department of Finance, Carroll School of Management, Boston College. zhangeq@bc.edu.

1 Introduction

Previous studies use machine-learning models to forecast stock returns and then, as a separate step, form long–short portfolios by ranking those forecasts. By contrast, this paper introduces a one-step approach that trains a model to assign optimal stock weights directly and maximize portfolio return.

The framework employs a neural network whose output layer produces a weight for each stock in every period. These weights generate the portfolio's return on the training sample, and that return is used as the loss function. Training therefore tunes the mapping from stock characteristics to weights with the single objective of maximizing the portfolio's expected return.

The one-step approach outperforms the two-step benchmark because the training objective is identical to the investment objective. Instead of first predicting each stock's return and then converting those forecasts into weights, the one-step model learns the weights directly, so it estimates only how strongly each stock should enter the portfolio. Eliminating this intermediate forecasting layer reduces overfitting and channels all model capacity toward the metric that ultimately matters, yielding superior out-of-sample performance.

When the investment objective shifts from maximizing expected return to maximizing risk-adjusted return, the model adds a quadratic return term to the loss function and jointly optimizes both risk and return. This extra penalty places greater weight on large negative outcomes, yielding a portfolio with more stable performance.

Using the data from Gu, Kelly, and Xiu (2020), we train the model on 94 firm-level predictors and 74 two-digit Standard Industrial Classification (SIC) dummies available since 1957, evaluating performance month by month. Computing the weight at the end of each month t uses only data observable at that time, and the performance is tested with returns in month $t + 1$. Starting in 1987, the model is re-estimated annually on the full sample accumulated to date.

From 1987 to 2021, the long–short portfolio from our one-step model from maximizing expected returns posts an annualized Sharpe ratio of 3.656, largely outperforms the two-step equal-weighted benchmark formed by sorting on expected returns (2.638). By sorting on the model's optimized weights and taking an equal-weighted top–10% minus

bottom–10% portfolio, the Sharpe ratio is 3.492, still largely above the benchmark, though below the portfolio using directly weights from the model output. A value-weighted portfolio built from the model's weights delivers a Sharpe ratio of 1.507, outperforming the benchmark's 1.306.

Performance improves further when the model maximizes risk-adjusted returns. The portfolio that uses the model's output weights reaches a Sharpe ratio of 4.057, while the equal-weighted version attains 3.895. This outcome is as expected: because our evaluation metric, the Sharpe ratio, adjusts for risk, a model that optimizes risk-adjusted returns is better aligned with the objective.

The model can also account for transaction costs. When trading is costly, a portfolio should not only target stocks with high expected returns but also limit its turnover. Instead of a two-step procedure that first estimates expected returns and then adjusts for turnover, our framework identifies the optimal portfolio in a single step by using after-cost portfolio returns (or risk-adjusted returns) in the loss function.

We test two assumptions about transaction costs. The first assumes a flat 0.5% cost on every trade. The portfolio optimized under this after-cost, risk-adjusted objective shows a combined long-plus-short turnover of 52.5% per month, compared with more than 70% for models that do not consider transaction costs in training. The lower trading expense leads to slightly better after-cost performance.

The second one assumes that trading cost is proportional to the ratio of our trading volume to total market volume. A larger relative trading volume implies a greater price impact, so the portfolio should tilt toward high-volume stocks and trade more during high-volume periods. By optimizing after-cost returns with this assumption, the model naturally include volume prediction, and tend to trade on stocks and in periods with high volume.

We assume that trading 10% of a stock's monthly volume moves its price by 2%, implying an average trading cost of 1%. The size of each long and short leg is fixed at 1 billion throughout the sample. Under these assumptions, the portfolio that optimizes after-cost returns still achieves a Sharpe ratio of 1.950, whereas portfolios that ignore such costs deliver much lower, or even negative after-cost returns because they must trade very illiquid stocks.

1.1 Literature

An extensive literature applies machine-learning methods to predict stock returns. Techniques range from linear regression (Lewellen 2015); linear regression with regularization (Feng, Giglio, and Xiu 2020; Freyberger, Neuhierl, and Weber 2020); tree-based models and feed-forward neural networks (Gu, Kelly, and Xiu 2020); to transformer architectures (Kelly, Kuznetsov, Malamud, and Xu 2025). While most studies rely on stock characteristics as predictors, some extract information from textual data (Ke, Kelly, and Xiu 2019; Chen, Kelly, and Xiu 2022) or from the volatility surface (Kelly, Kuznetsov, Malamud, and Xu 2023). Although their primary goal is to forecast returns, or equivalently to estimate the stochastic discount factor, these papers evaluate performance through the returns of long–short portfolios formed from the predictions; portfolios constructed from this "two-step" approach serve as the benchmark in this paper.

By contrast, this paper takes a one-step approach that optimizes portfolio weights directly to maximize expected return. Brandt (1999) use linear regressions to choose optimal weights between a stock index and Treasury bills, arguing that the one-step approach reduces misspecification and estimation error. Brandt, Santa-Clara, and Valkanov (2009) extend the idea to thousands of individual stocks. Building on this motivation, our neural-network model incorporates a richer set of firm characteristics and captures their nonlinear influence on optimal weights. As a result, the portfolio we construct outperforms the optimal portfolio of Brandt, Santa-Clara, and Valkanov (2009), which relies on only a few firm characteristics.

Cong, Tang, Wang, and Zhang (2021) apply a one-step reinforcement learning approach to optimize portfolio weights that update the weights in each month. Our method instead relies on batches on the cross-section: we draw random batches of stocks for each month, and iteratively train the model on calculated returns within each batch. This mini-batch approach reduces the risk of overfitting in a limited and noisy historical sample.

This study also relates to the literature on optimal trading strategies with transaction costs. Goyenko, Kelly, Moskowitz, Su, and Zhang (2024) use machine learning to forecast trading volume and show that net-of-cost strategy returns improve when trades concentrate in high-volume stocks if assuming the trading cost is lower with higher market trading volume. Their framework relies on a two-step procedure to build net-of-cost

portfolios, while we adopt a one-step approach both to reduce noise and to accommodate possible correlation between trading volume and alpha.

Jensen, Kelly, Malamud, and Pedersen (2024) use a one-step approach by analytically solving the optimal-portfolio problem with transaction costs and then finding the optimized weights with a machine learning model. Our method differs from Jensen, Kelly, Malamud, and Pedersen (2024) in that we do not include an analytical model and instead embed net-of-cost returns directly in the machine-learning loss function. Our approach is not only easier to implement, but also able to handle richer settings such as non-linear trading costs, or trading costs that vary with volume.

2 Methodology

2.1 Framework

In this subsection, we introduce a simple framework that optimizes portfolio weights directly. We define the loss function and then compare this approach with the traditional benchmark of estimating returns. Implementation details follow in the next subsection.

In our model, the portfolio weight for stock i at the end of month t is derived from the observable firm-level characteristics vector $\mathbf{x}_{i,t}$ available at that time:

$$w_{i,t} = f(\mathbf{x}_{i,t}). \tag{1}$$

Here, f is a nonlinear feed-forward neural network (FNN). We optimize f by minimizing the loss function.

The portfolio's excess return in month $t+1$ is the weighted return of all stocks:

$$r_{t+1} = \sum_{i=1}^{N_t} w_{i,t} r_{i,t+1}, \tag{2}$$

where N_t denotes the number of stocks available at the end of month t, and $r_{i,t+1}$ is stock i's excess return (its raw return minus the risk-free rate) in month t!+!1.

Because the model aims to maximize the average monthly excess return over the test sample, we define the loss function as the negative of that average return:

$$L(\mathbf{r}, \mathbf{X}) = -\frac{1}{T-1} \sum_{t=1}^{T-1} \sum_{i=1}^{N_t} w_{i,t} r_{i,t+1}, \tag{3}$$

where $\mathbf{r}$ denotes the vector of stock returns, and $\mathbf{X}$ denotes the corresponding vector of stock characteristics.

This is different from the approach in prior studies, where they first fit the returns through minimizing the mean squared error:

$$L^{MSE}(\mathbf{r}, \mathbf{X}) = \frac{1}{N} \sum_{i,t} (r_{i,t+1} - \hat{r}_{i,t+1})^2, \tag{4}$$

and then form the portfolio by sorting on the predicted return $\hat{r}_{i,t+1}$ in each period.

An additional quadratic term can be included when the objective is to maximize expected risk-adjusted returns rather than expected returns alone. A mean–variance-efficient portfolio is obtained by maximizing a linear combination of returns and squared returns, where the coefficient a represents risk tolerance:

$$L^{EXP}(\mathbf{r}, \mathbf{X}) = -\frac{1}{T-1} \sum_{t=1}^{T-1} \left[\sum_{i=1}^{N_t} w_{i,t} r_{i,t+1} - \frac{a}{2} \left(\sum_{i=1}^{N_t} w_{i,t} r_{i,t+1} \right)^2 \right]. \tag{5}$$

2.2 Implementation

2.2.1 Training batches

When training a neural network, it is best to split the data into mini-batches. Mini-batch training runs faster and the added randomness often speeds up convergence. Therefore, although our ultimate goal is to construct a portfolio using all stocks, each training step works with only a subsample of stocks.

Specifically, we randomly pick 10% of the stocks without replacement in each month for each batch. One epoch consists of 10 batches, and covers all stocks in each month.

We use mini-batches only during training. For validation and testing, we calculate portfolio returns with the full set of stocks each month.

2.2.2 Weight constraints

In the conventional equal-weighted construction, the long and short sides each hold the top and bottom 10% of stocks, so every selected stock receives a weight of $\frac{1}{N_t/10}$, where N_t is the total number of stocks in month t. To keep the portfolio from becoming overly concentrated, we require that no individual weight exceed this benchmark. We impose the cap through the following sigmoid transformation:

$$w_{i,t} = f(\mathbf{x}_{i,t}) = \frac{1}{N_t/10}(-1 + \frac{2}{1 + exp(-FFNN(\mathbf{x}_{i,t}))}). \tag{6}$$

The feed-forward neural network outputs $FFNN(\mathbf{x}_{i,t}) \in [-\infty, \infty]$, and the transformation above maps this value to a portfolio weight $w_{i,t} \in \left[-\frac{1}{N_t/10}, \frac{1}{N_t/10}\right]$. Because training uses mini-batches, N_t denotes the number of stocks in the batch for month t.

In addition to capping individual weights, we require that the long-leg weights sum to one and that the short-leg weights also sum to one in absolute value, ensuring a properly scaled long–short portfolio:

$$\sum_{w_{i,t}>0} w_{i,t} = 1, \ \sum_{w_{i,t}<0} -w_{i,t} = 1. \tag{7}$$

This is equivalent to

$$\sum_{i=1}^{N_t} w_{i,t} = 0, \ \sum_{i=1}^{N_t} |w_{i,t}| = 2. \tag{8}$$

We normalize the weight output of each training batch in each period by subtracting the average to ensure the former constraint holds. For the latter one, we add penalty terms to the loss function in each period:

$$Penalty_t^{Weight} = \max(0, \sum_{i=1}^{N_t} |w_{i,t}| - 2). \tag{9}$$

Because training relies on random mini-batches, the long and short weights cannot always sum exactly to one. Nevertheless, the normalization and the penalty term prevents the model from seeking higher performance by expanding its overall position size or by tilting the portfolio too heavily toward the long side.

The final loss function we use is

$$L(\mathbf{r}, \mathbf{X}) = \frac{1}{T-1} \sum_{t=1}^{T-1} (- \sum_{i=1}^{N_t} w_{i,t} r_{i,t+1} + Penalty_t^{Weight}), \tag{10}$$

During testing, the long- and short-leg weights must each sum to exactly one. To satisfy this requirement, we rescale the feed-forward neural-network output $FFN(\mathbf{x}_{i,t})$ by the total output of its respective leg:

$$w_{i,t} = \begin{cases} \frac{FFNN(\mathbf{x}_{i,t})}{\sum_{FFNN(\mathbf{x}_{j,t})>0} FFNN(\mathbf{x}_{j,t})}, \ if \ FFNN(\mathbf{x}_{i,t}) > 0, \\ \frac{FFNN(\mathbf{x}_{i,t})}{\sum_{FFNN(\mathbf{x}_{j,t})<0} FFNN(\mathbf{x}_{j,t})}, \ if \ FFNN(\mathbf{x}_{i,t}) < 0. \end{cases} \tag{11}$$

2.2.3 Variance penalty

In practice, training a model with a full quadratic penalty term (Equation 5) is challenging, because it also applies a large penalty to substantial positive returns, destabilizing the gradients. Instead, we impose the quadratic penalty only when the return falls below a threshold r_0, keeping the loss function linear for returns above r_0:

$$Penalty^{Variance} = \begin{cases} k(r - r_0)^2, & if \ r < r_0, \\ 0, & if \ r >= r_0. \end{cases} \tag{12}$$

This approach still prevents the model from experiencing large negative returns, thus reducing variance, while allowing large positive returns.

2.3 Trading costs

The loss function can be further modified to account for trading costs. For each stock in the batch in month t, we calculate the weight of each stock for month $t-1$ using its one-month-lagged characteristics, $\mathbf{x}_{i,t-1}$.

$$w_{i,t-1} = f(\mathbf{x}_{i,t-1}). \tag{13}$$

Let $\phi_{i,t}$ denote the transaction cost function for stock i in month t. The portfolio's total transaction cost at the end of month t is therefore

$$\sum_{i=1}^{N_t} \phi_{i,t}(w_{i,t} - w_{i,t-1}). \tag{14}$$

For simplicity, we ignore the mechanical change in weights—namely, the shift in each position from month $t-1$ to month t that arises solely from price movements. In practice, those who need greater precision can instead compute trading costs as $\sum_{i=1}^{N_t} \phi_{i,t}((1 + g_t)w_{i,t} - \frac{p_{i,t}}{p_{i,t-1}} w_{i,t-1})$, where g_t is the portfolio's overall growth rate in time t. Because our framework is entirely non-parametric, such refinements are easy to incorporate.

The net-of-cost return is then

$$r_{t+1}^{net} = \sum_{i=1}^{N_t} w_{i,t} r_{i,t+1} - \phi_{i,t}(w_{i,t} - w_{i,t-1}). \tag{15}$$

Note that whether a stock here is actually included in the batch in month $t-1$ is irrelevant, because we can still compute its hypothetical weight in month $t-1$ from its lagged characteristics.

Different assumptions on the transaction cost can be implemented. The simplest assumption is constant trading cost:

$$\phi_{i,t}(w_{i,t} - w_{i,t-1}) = c(w_{i,t} - w_{i,t-1}). \tag{16}$$

Following Frazzini, Israel, and Moskowitz (2018) and Goyenko, Kelly, Moskowitz, Su, and Zhang (2024), we may also assume that the trading cost is proportional to the ratio of our trading volume to the market's trading volume:

$$\phi_{i,t}(w_{i,t} - w_{i,t-1}) = \frac{\frac{1}{2}\lambda W_t}{\tilde{V}_{i,t}}(w_{i,t} - w_{i,t-1})^2, \tag{17}$$

where W_t is the portfolio's size and $\tilde{V}_{i,t}$ is stock i's total trading volume in month t. This quadratic specification stems from the linear price-impact model of Kyle 1985. In this case, purchasing 1% of a stock's monthly volume is assumed to move its price by λ%, yielding an average trading cost of $\frac{1}{2}\lambda$%,.

2.4 Model specification

We use a feed-forward neural network with three hidden layers of 32, 16, and 8 neurons, following Gu, Kelly, and Xiu (2020) and the "pyramid" rule of Masters (1993). Each layer employs the ReLU activation function, $\max(0, x)$, and the model is trained with the Adam optimizer (Kingma, 2014).

To enhance robustness, we include several regularization components: an L_2 weight-decay penalty with $\lambda = 10^{-3}$, a learning-rate schedule that starts at 10^{-2} and decays by a factor $\gamma = 0.8$ in each epoch, and batch normalization after each hidden layer (Ioffe, 2015). To ensure consistency, the random seed is fixed at 42 and training always stops after 30 epochs.

The model is re-estimated at the start of each calendar year from 1987 onward. For each year y, the training sample covers 1957 through $y - 1$. This rolling training sample ensures that most recent information is always included.

We use the same neural-network architecture to train all the models that optimizes portfolio weights with different loss functions, and also the benchmark model that predicts monthly stock returns for comparison.

3 Data

Our primary data source is the monthly stock-level characteristics dataset of Gu, Kelly, and Xiu (2020). It covers every NYSE, AMEX, and NASDAQ listing from March 1957 through December 2021, with an average cross-section of more than 6,000 stocks per month. Stock returns come from CRSP, and the monthly risk-free rate is taken from Kenneth French's Data Library. All portfolio returns are computed as excess returns, which are raw returns minus the risk-free rate.

Following Gu, Kelly, and Xiu (2020), we use 94 monthly firm-level characteristics and 74 two-digit SIC industry dummies as predictors. For variables reported annually or quarterly, we carry forward the most recent observation until a new value appears.

Each firm characteristic is standardized to lie between -1 and 1 by ranking it within the month. Missing observations are set to 0, the cross-sectional median.

The out-of-sample testing period runs from 1987 to 2021, spanning 35 calendar years.

At the start of each test year, the model is retrained to incorporate all newly available training data.

4 Results

4.1 Optimizing returns

The first model we study optimizes only the long–short portfolio's return. At the start of each calendar year, we train the feed-forward neural network $FFN(\mathbf{x}_{i,t})$ on the available training sample by minimizing the loss function in Equation (10). The weights obtained from the updated model $w_{i,t}$ are then applied to the test sample for that year.

The performance metrics of all models are presented in Table 1, and Figure 1 plots the cumulative return of the long–short strategy produced by our one-step model. Panel (a) presents the portfolio performance using optimized weights from the model as in Equation (11). Over the full 1987–2021 test period, this strategy earns an average monthly return of 3.68% and an annualized Sharpe ratio of 3.656. Consistent with the findings in the literature, the returns to long–short equity strategies decline noticeably after 2000.

This one-step weight-optimization approach outperforms the conventional "predict-then-sort" benchmark. Panel (a) of Figure 2 shows cumulative returns for the benchmark strategy, which sorts stocks by predicted returns: the top 10% form the long leg, and the bottom 10% form the short leg. The benchmark model is trained on the same inputs and uses the same neural-network architecture. From 1987 to 2021, it delivers an average monthly return of 3.88% and a Sharpe ratio of 2.638.

Our approach achieves a higher Sharpe ratio than the benchmark in both the full sample and the recent subsample, although its average return is lower because the portfolio weights are more dispersed. To facilitate a closer comparison, we also construct an equal-weighted portfolio by ranking stocks on the model's weight outputs and then assigning identical weights within the top and bottom 10%. In this variant, the optimized weights serve only to select stocks, not to set their positions. Panel (b)of Figure Figure 1 plots the cumulative return. The Sharpe ratio is 3.492, still higher than the benchmark, and the average monthly return 4.19 is also higher than the benchmark.

Because equal-weighted portfolios tend to be dominated by small-cap stocks, we also

evaluate value-weighted portfolios. We again sort stocks by the model's weight outputs, but now assign weights in proportion to each firm's market capitalization from the previous month. Over the full sample, this value-weighted strategy earns an average monthly return of 2.31% and an annualized Sharpe ratio of 1.507. The cumulative return graph is in Panel (c) of Figure 1.

The strategy also surpasses the value-weighted benchmark formed by sorting on predicted returns, whose average returns are 2.29% with a Sharpe ratio of 1.306. The cumulative return graph is in Panel (b) of Figure 2. After 2004, the benchmark's value-weighted returns are essentially zero, whereas the value-weighted portfolio produced by our model continues to generate meaningful gains.

Skewness, kurtosis, and maximum drawdown are similar for our model and the benchmark. Both return distributions are right-skewed with heavy tails. Maximum drawdowns remain low for the optimized and equal-weighted portfolios but are much higher for the value-weighted portfolio, with the worst drawdown occurring during the recovery period after the 2008 financial crisis.

4.2 Optimizing risk-adjusted returns

Portfolio performance improves further when we use risk-adjusted returns as the loss function. The additional penalty term, shown in Equation (12), is applied with threshold $r_0 = 0.1$—so monthly returns above 10% are not penalized—and risk-aversion parameter $k = 10$. Performance might rise with better-tuned values. Results for this alternative specification are also in Table 1, and Figure 3 plots the cumulative returns.

Over the same 1987–2021 test period, the alternative model attains a Sharpe ratio of 4.057, driven by lower volatility; the average monthly long–short return is slightly lower at 3.49%. By penalizing large negative returns, the model delivers more stable performance while giving up only a modest amount of expected return.

There is also an improvement in the performance of the equal-weighted portfolio sorted on the model outputs. However, the value-weighted portfolio does not perform better, which is reasonable because the model is not trained on value-weighted returns.

4.3 Constant trading cost

In this subsection we impose a constant one-way 0.5% trading cost in the loss function, independent of stock liquidity or order size. If all long and short positions are fully turned over in one month, the total trading cost would be 2%—0.5% for buying and 0.5% for selling each leg. Although this is not a proper assumption practically, this assumption lets us show how the model accommodates the trading cost by reducing turnover.

As noted in Section 2.3, the cost of adjusting a position in stock i is $c(w_{i,t} - w_{i,t-1})$, with $c = 0.5\%$. The model therefore maximizes risk-adjusted returns net of this cost, as in the previous subsection.

We retain the same neural-network architecture but modify the inputs. Rather than feeding only the most recent signals, we also include lags of up to one year. These historical signals help the network infer past weights and generate smoother trades. In addition, for each test year we retrain the model on only the most recent 30 years of data.

Table 2 summarizes portfolio performance. The model that optimizes risk-adjusted returns net of cost delivers the best results, posting an after-cost Sharpe ratio of 3.344. It outperforms one-step models that ignores the transaction cost and the two-step benchmark based on sorting predicted returns. Equal-weighted and value-weighted portfolios sorted on the model outputs also outperform the earlier models and the benchmark. The equal-weighted portfolio incurs even a lower trading cost on average and attains a higher Sharpe ratio than the portfolio that uses the model's raw weights.

The advantage arises from lower turnover ratios. Average monthly transaction cost falls to 0.525%, implying a combined long-plus-short turnover of 52.5% per month, whereas models that omit costs turn over more than 70% across the two legs. However, the model that incorporates the transaction cost exhibits slightly higher return volatility, so the overall performance gain is limited.

4.4 Linear price impact

This subsection considers a more realistic assumption: price impact rises as order size grows relative to market trading volume. Under this assumption, the model should assign smaller weights to less liquid stocks and refrain from trading during illiquid periods.

We set $\phi_{i,t}(w_{i,t} - w_{i,t-1}) = \frac{\frac{1}{2}\lambda W_t}{\tilde{V}_{i,t}}(w_{i,t} - w_{i,t-1})^2$, with $\lambda W_t = 0.2$. Consequently, purchasing 10% of a stock's monthly volume in one month moves its price by 2%, producing an average trading cost of 1%. Because some stocks exhibit very low—or even zero—trading volume, which would generate unreasonably high costs under this function, we cap trading cost at 50%. Hence, the trading-cost function we use is:

$$\phi_{i,t}(w_{i,t} - w_{i,t-1}) = \min\left\{\frac{\frac{1}{2}\lambda W_t}{\tilde{V}_{i,t}}|w_{i,t} - w_{i,t-1}|, 0.5\right\}|w_{i,t} - w_{i,t-1}| \tag{18}$$

The long–short portfolio's size (AUM), W_t, is fixed at 1 billion across both the training and test samples. We use the same neural network as in the previous subsection to maximize risk-adjusted, after-cost returns; the only change is the transaction-cost function.

As shown in Table 3, the model trained with a linear price-impact assumption achieves the best test-sample performance. Its average monthly trading cost is just 0.246%, indicating that the model successfully avoids less-liquid stocks and periods. All other models incur average monthly costs above 2%, resulting in much lower, or even negative, after-cost returns. The equal-weighted portfolio sorted on the model's outputs likewise outperforms the alternatives, whereas the value-weighted version does not. Because value-weighted portfolios already have low trading costs under linear price impact, further optimization offers little additional benefit.

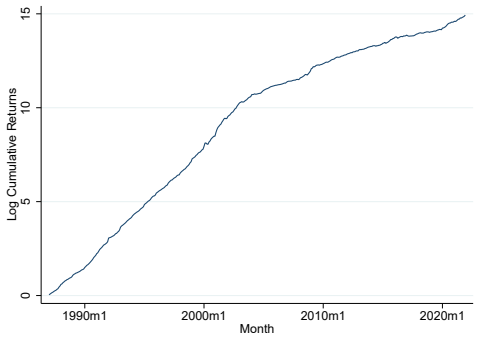
5 Conclusion

This paper proposes a practical method for optimizing portfolio weights directly with a neural network. The network's outputs are the stocks' relative weights, and the loss function is the portfolio's return. By targeting the final objective outright, the model avoids noise from intermediate estimation steps and thus delivers superior performance.

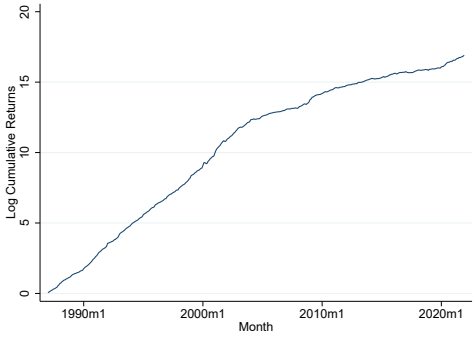
This approach is particularly valuable when the market expected return is not the only consideration. Risk-adjusted performance improves by adding a quadratic term to the loss function, which penalizes downside risk. When after-cost returns enter the loss, the optimized portfolio shows lower turnover and, if trading costs fall with higher market volume, tilts toward more liquid stocks.

Figures and Tables

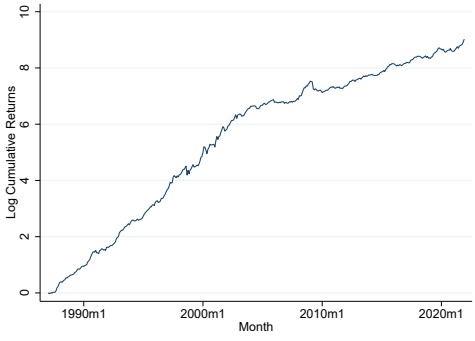
Figure 1: Cumulative returns of portfolio from optimizing portfolio returns



(a) Optimized weights
Sharpe = 3.656



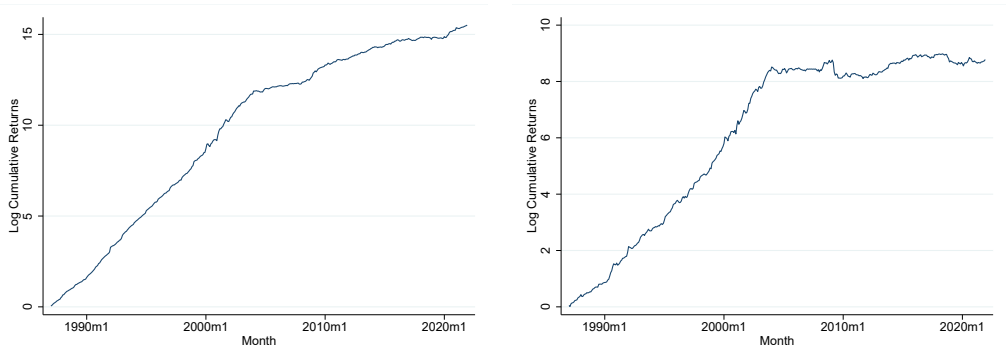
(b) Equal-weighted
Sharpe = 3.492



(c) Value-weighted
Sharpe = 1.507

Notes: The figure plots cumulative returns for our long–short strategy from optimizing portfolio weights across all NYSE, AMEX, and NASDAQ stocks from 1987 to 2021. Panel (a) uses the model's weight outputs; Panel (b) applies equal weights to the top and bottom deciles of those weights; Panel (c) shows the analogous value-weighted portfolio.

Figure 2: Cumulative returns of portfolio from sorting on expected returns

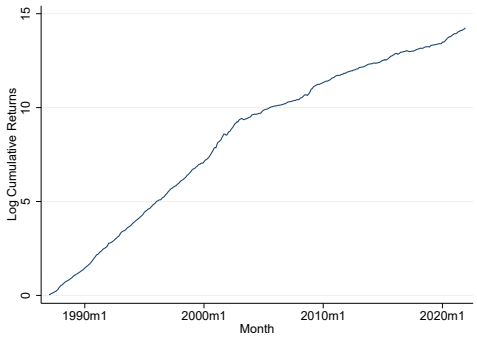


(a) Equal-weighted
Sharpe = 2.638

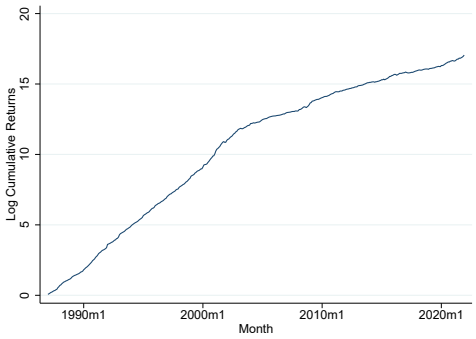
(b) Value-weighted
Sharpe = 1.306

Notes: The figure plots cumulative returns for the benchmark long–short strategy from sorting on expected returns across all NYSE, AMEX, and NASDAQ stocks from 1987 to 2021. Panel (a) applies equal weights to the top and bottom deciles of expected returns; Panel (b) shows the value-weighted portfolio.

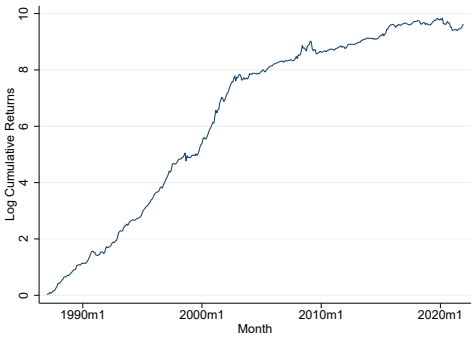
Figure 3: Cumulative returns of portfolio from optimizing risk-adjusted portfolio returns



(a) Optimized weights
Sharpe = 4.057



(b) Equal-weighted
Sharpe = 3.895



(c) Value-weighted
Sharpe = 1.472

Notes: The figure plots cumulative returns for our long–short strategy from optimizing risk-adjusted portfolio weights across all NYSE, AMEX, and NASDAQ stocks from 1987 to 2021. Panel (a) uses the model's weight outputs; Panel (b) applies equal weights to the top and bottom deciles of those weights; Panel (c) shows the analogous value-weighted portfolio.

Table 1: Portfolio performance

Model	Mean(%)	Std(%)	Sharpe	Skew	Kurt	Max DD (%)
One-step Model, optimizing returns						
Optimized weights	3.676	3.484	3.656	1.441	8.051	6.975
Equal-weighted	4.188	4.155	3.492	1.358	6.993	7.534
Value-weighted	2.312	5.315	1.507	-0.070	7.202	33.361
One-step Model, optimizing risk-adjusted returns						
Optimized weights	3.494	2.983	4.057	0.731	5.291	7.204
Equal-weighted	4.209	3.743	3.895	1.234	7.188	6.165
Value-weighted	2.486	5.852	1.472	0.116	6.789	36.205
Benchmark Two-step Model						
Equal-weighted	3.881	5.097	2.638	1.762	9.961	14.468
Value-weighted	2.289	6.074	1.306	0.707	7.220	48.093

Notes: This table reports performance metrics for each strategy. "Optimized weights" uses the model's raw weights, while the equal- and value-weighted versions long the top decile and short the bottom decile based on either optimized weights or expected returns. 'MaxDD' represents maximum drawdown, which is defined as the largest peak-to-trough decline in the portfolio value.

Table 2: Portfolio performance with constant trading costs

Model	Mean(%)	Std(%)	Sharpe	Skew	Kurt	MaxDD(%)	TC(%)
Optimizing after-cost risk-adjusted returns							
Optimized weights	3.020	3.128	3.344	0.971	7.717	11.729	0.525
Equal-weighted	3.941	3.811	3.582	1.764	11.332	7.415	0.282
Value-weighted	1.782	5.632	1.096	0.604	7.594	48.966	0.549
Optimizing risk-adjusted returns							
Optimized weights	2.782	2.975	3.240	0.746	5.336	8.516	0.711
Equal-weighted	3.811	3.725	3.544	1.245	7.238	7.150	0.398
Value-weighted	1.756	5.849	1.040	0.107	6.800	49.906	0.731
Optimizing returns							
Optimized weights	2.952	3.464	2.953	1.454	8.121	8.508	0.724
Equal-weighted	3.803	4.141	3.181	1.368	7.039	9.062	0.385
Value-weighted	1.553	5.313	1.013	-0.085	7.221	39.027	0.765
Benchmark Two-step Model							
Equal-weighted	3.336	4.889	2.364	1.854	11.150	18.959	0.410
Value-weighted	1.545	5.634	0.950	0.772	8.716	63.260	1.197

Notes: This table reports after-cost performance metrics for each strategy, assuming that the trading cost is constant at 0.5%. "Optimized weights" uses the model's raw weights, while the equal- and value-weighted versions long the top decile and short the bottom decile based on either optimized weights or expected returns. 'MaxDD' represents maximum drawdown, which is defined as the largest peak-to-trough decline in the portfolio value. 'TC' represents average monthly transaction costs.

Table 3: Portfolio performance with linear price impact

Model	Mean(%)	Std(%)	Sharpe	Skew	Kurt	MaxDD(%)	TC(%)
Optimizing after-cost risk-adjusted returns, assuming linear price impact							
Optimized weights	2.000	3.553	1.950	-0.720	12.058	29.573	0.246
Equal-weighted	2.312	3.828	2.092	0.786	10.195	17.967	0.662
Value-weighted	0.882	4.133	0.739	0.647	8.517	29.574	0.015
Optimizing after-cost risk-adjusted returns, assuming constant trading costs							
Optimized weights	0.910	3.592	0.877	0.190	5.556	91.909	2.635
Equal-weighted	1.654	4.050	1.414	0.708	8.175	82.165	2.570
Value-weighted	1.704	5.623	1.050	0.584	7.302	47.071	0.627
Optimizing risk-adjusted returns							
Optimized weights	0.293	3.935	0.258	-0.248	3.737	98.437	3.201
Equal-weighted	0.901	4.270	0.731	0.160	4.515	96.941	3.308
Value-weighted	1.555	5.988	0.899	0.204	6.454	49.748	0.932
Optimizing returns							
Optimized weights	-0.112	4.676	(-)	-0.096	4.624	99.645	3.827
Equal-weighted	0.983	4.662	0.731	0.641	5.775	96.909	3.204
Value-weighted	1.461	5.389	0.939	0.090	6.843	46.381	0.856
Benchmark Two-step Model							
Equal-weighted	0.232	5.569	0.144	0.956	8.015	99.103	3.514
Value-weighted	1.257	5.656	0.770	0.804	8.897	57.907	0.957

Notes: This table reports after-cost performance metrics for each strategy, assuming that the trading cost equals to $\frac{0.1W}{\tilde{V}_{i,t}}|w_{i,t} - w_{i,t-1}|$, where $\tilde{V}_{i,t}$ is the monthly volume and W is the portfolio size, assumed to be fixed at 1 billion. "Optimized weights" uses the model's raw weights, while the equal- and value-weighted versions long the top decile and short the bottom decile based on either optimized weights or expected returns. 'MaxDD' represents maximum drawdown, which is defined as the largest peak-to-trough decline in the portfolio value. 'TC' represents average monthly transaction costs.

References

Brandt, M. W. (1999). Estimating portfolio and consumption choice: A conditional euler equations approach. *The Journal of Finance 54*(5), 1609–1645.

Brandt, M. W., P. Santa-Clara, and R. Valkanov (2009). Parametric portfolio policies: Exploiting characteristics in the cross-section of equity returns. *The Review of Financial Studies 22*(9), 3411–3447.

Chen, Y., B. T. Kelly, and D. Xiu (2022). Expected returns and large language models. *Available at SSRN 4416687*.

Cong, L. W., K. Tang, J. Wang, and Y. Zhang (2021). Alphaportfolio: Direct construction through deep reinforcement learning and interpretable ai. *Available at SSRN 3554486*.

Feng, G., S. Giglio, and D. Xiu (2020). Taming the factor zoo: A test of new factors. *The Journal of Finance 75*(3), 1327–1370.

Frazzini, A., R. Israel, and T. J. Moskowitz (2018). *Trading costs*, Volume 3229719. SSRN.

Freyberger, J., A. Neuhierl, and M. Weber (2020). Dissecting characteristics nonparametrically. *The Review of Financial Studies 33*(5), 2326–2377.

Goyenko, R., B. T. Kelly, T. J. Moskowitz, Y. Su, and C. Zhang (2024). Trading volume alpha. *National Bureau of Economic Research Working Paper*.

Gu, S., B. Kelly, and D. Xiu (2020). Empirical asset pricing via machine learning. *The Review of Financial Studies 33*(5), 2223–2273.

Ioffe, S. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jensen, T. I., B. T. Kelly, S. Malamud, and L. H. Pedersen (2024). Machine learning and the implementable efficient frontier. *Swiss Finance Institute Research Paper* (22-63).

Ke, Z. T., B. T. Kelly, and D. Xiu (2019). Predicting returns with text data. *National Bureau of Economic Research Working Paper*.

Kelly, B. T., B. Kuznetsov, S. Malamud, and T. A. Xu (2023). Deep learning from implied volatility surfaces. *Swiss Finance Institute Research Paper* (23-60).

Kelly, B. T., B. Kuznetsov, S. Malamud, and T. A. Xu (2025). Artificial intelligence asset pricing models. Technical report, National Bureau of Economic Research.

Kingma, D. P. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kyle, A. S. (1985). Continuous auctions and insider trading. *Econometrica: Journal of the Econometric Society*, 1315–1335.

Lewellen, J. (2015). The cross section of expected stock returns. *Critical Finance Review* (2511246), 1–44.

Masters, T. (1993). *Practical neural network recipes in C++*. Morgan Kaufmann.