Vienna University of Economics and Business

Department of Finance, Accounting and Statistics

Bachelor Thesis

Beating $1/N$ - A Machine Learning Approach

Author: Christopher HOFMANN
Date of birth: 12.03.1987
Student ID number: 00753983

Supervisor: Univ.Prof.Dipl.-Ing.Dr.techn. Kurt Hornik
Co-Supervisor: Julian Amon, MSc (WU)

October 2, 2020

Contents

List of Tables

1. Introduction

Modern portfolio theory is one of the most researched fields in economics. Starting with the ground-breaking work of Markowitz (1952), building the mathematical framework to find the optimal asset allocation by maximizing the return for a given level of risk, many scholars extended his work but the fundamentals from nearly 70 years ago still remain the bedrock of modern finance.

In theory, when expected return and covariance matrix of the available assets are known the mean-variance portfolio delivers the optimal allocation to a utility-maximizing investor, but in practice these model inputs have to be estimated. In an extensive study by DeMiguel et al. (2009) the authors compared a variety of optimization techniques against a simple equally-weighted baseline over multiple datasets. The conclusion was that none of these mathematical sophisticated strategies, many of which were rooted in the seminal groundwork by Markowitz (1952), were able to consistently show better out-of-sample performance than a naive $1/N$ benchmark that allocates equal portfolio weights to all available assets.

The aim of this thesis is to take a more modern approach towards the question of whether sophisticated strategies of portfolio optimization are able to outperform a naive benchmark. The idea is to utilize recent advancements in deep learning to train a neural network in a supervised fashion, specifically a Long Short-Term Memory architecture introduced by Hochreiter and Schmidhuber (1997). On the basis of this model, the intention is to predict the optimal out-of-sample asset allocation for 15 individual stocks from the S&P 100 index. Since neural networks allow for vector-based predictions the goal is to use the last layer of the network directly as portfolio weights. In order to construct portfolios for rational investors trying to optimize the risk and return, we train the model on optimal in-sample mean-variance weights. We apply the following four input features to our network: stock returns, the Relative Strength Index, CBOE's Volatility Index and the put-to-call ratio. Our objective is not to achieve high model accuracy but good out-of-sample portfolio performance on real data. For this we compare the model in terms of Sharpe ratio, CAPM Alpha and Fama-Frech's Three Factor Alpha against the naive $1/N$ portfolio, which serves as a easy-to-implement benchmark that requires no computational resources for optimization and does not rely on the estimation of the expected asset returns as well as the associated covariance matrix.

2. Literature Review

In his work Markowitz (1952) assumed that both mean and variance are known with certainty but in practice these parameters are subjected to estimation risk. Despite the solid theoretical foundation of modern portfolio theory this shortcoming led to a strand of criticism by many scholars (e.g. Kalymon, 1971; Klein and Bawa, 1976). Furthermore, Frankfurter et al. (1971) performed an oversimplified experiment where the authors ques-

tioned the usefulness of mean-variance optimization after finding strong negative impacts on the portfolio performance caused by estimation errors. In a computational simulation by Best and Grauer (1991) on the behavior of different sized portfolios the authors reached the conclusion that changes in asset mean estimates lead to extreme sensitivity in composition, return and risk of the portfolio. When non-negativity constraints are applied the weights still respond very sensitive but the mean and variance of the portfolio tends to remain almost constant. Similarly, Chopra and Ziemba (1993) showed that, dependent on the investors risk tolerance, even small changes in input parameters result in large changes in the optimal portfolios.

Given the assumptions and shortfalls of the mean-variance approach, Cheng and Liang (2000) brought up the question whether mathematical optimizations are of any practical use to improve the risk-adjusted performance compared to a simple equally-weighted strategy. Their empirical study concluded that - in terms of Sharpe ratio - the in-sample optimization achieved superior performance over the benchmark. On the other hand, the out-of-sample measures delivered no significant gains. In a similar vein, the highly influential research by DeMiguel et al. (2009) tested 14 different optimization techniques across 7 datasets against a naively diversified portfolio. The paper considered the out-of-sample performance measures of Sharpe ratio, certainty-equivalent returns and the turnover relative to the benchmark. To test a variety of strategies, the authors have included Bayesian approaches, portfolios with moment restrictions, optimizations that constrain short selling, and combinations of optimal portfolios. None of these optimized models, many explicitly accounting for estimation error, was able to show significant outperformance relative to the simple baseline over all datasets. Their conclusion was that the errors from estimating the mean and variance overcompensate all gains provided by optimization. In addition, a simulation was carried out indicating that extremely long estimation windows are needed to mitigate this *error-maximizing property* of mean-variance portfolios described by Michaud (1989).

A further path to follow in order to challenge the empirical lack of benefit from portfolio optimization is to employ the expressive power of (deep) neural networks to try to learn a map from measurable characteristics of an asset today to the optimal portfolio weights tomorrow. Despite the fact that machine learning, especially deep learning, is a relative new domain in finance, a large number of different approaches to compute the optimal asset allocation for a portfolio have been studied in recent years. The authors of Paiva et al. (2019) apply support-vector machines identifying stocks with the highest likelihood to reach a specific daily return and then use the classic mean-variance approach for portfolio optimization. They present no risk-adjusted performance measures but show a higher cumulative return against their baselines. Building on their successful deployment for the purposes of image recognition, Hoseinzade and Haratizadeh (2019) utilize convolutional neural networks to predict up and down moves in stock prices. In a study on multiple indices they indicate improved performance on Sharpe ratio and certainty equivalent compared to a simple buy-and-hold strategy. On the other hand, Song et al. (2017) tackle the task with two supervised learn-to-rank algorithm, namely ListNet

and RankNet. Constructing portfolios with the highest and lowest ranked stocks, both networks delivered superior performance against their benchmarks regarding Sharpe ratio. Fischer and Krauss (2018) were among the first predicting stock gains one day ahead with LSTM networks using the last 240 daily returns as input features. Creating a strategy going the best k stocks long and shorting the flop k stocks, this type of deep learning outperformed the random forests, logistic regression and deep neural network baseline. In response to Fischer and Krauss (2018) a wider set of features were proposed by Wang et al. (2020). Furthermore, the authors combined the k best performing stocks with a minimum-variance optimization showing better results than various machine learning benchmarks. One more study confirming the ability of LSTMs predicting time series was conducted by Xiong et al. (2015). One single long short-term memory layer outperformed ridge and lasso regressions as well as an autoregressive model in predicting the volatility of the S&P 500 index. With the recent progress in reinforcement learning, Park et al. (2020) approach the optimization problem by training an agent via Q-learning. The action space, in an environment of five market features, is restricted to a fixed trading size where the agent is allowed to hold, buy or sell the available assets. The results are tested in terms of Sharpe ratio, Sterling ratio and cumulative returns against the baseline of a buy-and-hold portfolio and randomly taken actions. The Q-learning algorithm outperformed the benchmarks on a US portfolio in three test periods.

3. Methodology and Data

In this section we lay out the setup for our investigations. First, we give a brief overview of deep learning and the Long Short-Term Memory (LSTM) network architecture by Hochreiter and Schmidhuber (1997). Second, we describe how features and targets are selected, as well as separated into training/validation/test sets. Third, we explain our network architecture and settings for the training process, where we use the R package *keras* (Allaire and Chollet, 2020) with Google's *TensorFlow* backend to fit the model and make predictions. Finally, to test the portfolio weights given by the last network layer, we explain the methodology for our performance measures and statistical tests. At the end of this section we provide the datasets spanning a time horizon from 01/11/2006 to 04/10/2019. The empirical analyses is performed in *R* (R Core Team, 2020).

With the LSTM outputs we construct an optimal portfolio composed of 15 stocks included in the S&P 100 index with a long enough price history to train the supervised learning algorithm properly. Table 1 lists all of those individual companies with a diverse combination across sectors.

3.1. Deep Learning with recurrent neural networks

Deep learning is a subfield of machine learning in which an artificial neural network (ANN) is utilized to output predictions by recognizing patterns in input data. The basic unit in a neural network is the neuron. It receives inputs from other neurons or in the case of the input layer from external data and computes an output. Several of these units

Table 1: List of companies including Yahoo Finance Ticker and GCSI sector

Company	**Ticker**	**Sector**
Amazon.com, Inc.	AMZN	Consumer Discretionary
American Tower Corporation	AMT	Real Estate
DuPont de Nemours	DD	Materials
Exxon Mobil Corporation	XOM	Energy
Gilead Sciences, Inc.	GILD	Health Care
Honeywell International Inc.	HON	Industrials
JPMorgan Chase & Co.	JPM	Financials
Mastercard Incorporated	MA	Information Technology
Merck & Co.	MRK	Health Care
NextEra Energy, Inc.	NEE	Utilities
Procter & Gamble	PG	Consumer Staples
Target Corporation	TGT	Consumer Discretionary
Texas Instruments Incorporated	TXN	Information Technology
The Walt Disney Company	DIS	Communication
Walmart Inc.	WMT	Consumer Staples

are arranged in a series of layers. The most simple type of ANN, called a *feed forward network with fully connected layers*, connects all neurons in a layer to every neuron in the previous layer and lets the data only flow in one direction. Each unit contains a bias and the values from the incoming connections are weighted. The output vector h of one layer is computed by

$$h(X) = \phi(XW + b), \tag{1}$$

in which X represents the matrix of input features, W is the weight matrix and vector b contains one bias per neuron. ϕ is an activation function that gives the layer output a non-linear property (see Géron, 2019).

Initially, weights and biases are random values hence the representations of the model are meaningless. In supervised learning, where the network outputs are trained on true values, the following training loop is repeated as long as necessary. First, a batch of input features and corresponding targets are drawn from a dataset and the forward pass is performed by calculating the output h for each layer with equation (1). In the next step, the loss of the network is computed by measuring the mismatch between the predictions of the last layer and the true target values. Finally, all parameters must be updated in a way that reduces the loss in the next forward pass. This process is done by the

backpropagation algorithm. It goes through every single layer in reverse and computes the error contribution from each connection by calculating the gradient with the chain rule. The neurons are then updated in the opposite direction to the gradient and the loop starts again with the next batch of data. Learning is stopped after a set amount of epochs (i.e. iterations over the entire training set) with the network (hopefully) being able to make accurate predictions on unseen data (see Chollet and Allaire, 2018).

3.1.1. The Long Short-Term Memory (LSTM)

The main element of our empirical research is a special type of recurrent neural network (RNN) architecture, called a Long Short-Term Memory (LSTM). In contrast to simple feed forward networks, RNNs have the ability to learn long term dependencies by iterating through the elements of a sequence and keep information of what they have seen so far. During training of such deep networks with long sequences, the gradient of the loss can become extremely small. Since the derivative tells the weights and biases in which direction to move, the network stops training. This *vanishing gradient problem* was found and solved by Hochreiter and Schmidhuber (1997). The authors propose a special LSTM cell that maintains a state vector and at each time step the network can choose to add/remove information or even reset the cell state. Metaphorically speaking, along the normal sequence processing runs a conveyor belt where data can jump on and off at any time (see Chollet and Allaire, 2018). With LSTMs being a trainable class of neural networks that are able to process past information, they are the perfect choice to learn the long-term dependencies in time series and predict optimal asset allocations.

3.1.2. Feature generation

A good set of input features is the foundation for our research. Since we are predicting daily portfolio weights, our focus lies on technical indicators and sentiment variables rather than long term company fundamentals.

The first input features to the LSTM network are the past stock returns. From our dataset of length T including N stocks we calculate the N-dimensional vector of daily excess returns over the risk-free rate r_f at time t. For stock i, represented by the ith component of vector R_t, this calculation is hence performed as

$$R_{i,t} = \frac{P_{i,t}}{P_{i,t-1}} - 1 - r_{f,t}, \qquad \begin{aligned} t &= 1, 2, \ldots, T, \\ i &= 1, 2, \ldots, N, \end{aligned} \tag{2}$$

where $P_{i,t}$ is the closing price of stock i at time t.

The second feature, the Relative Strength Index (RSI) developed by Wilder jr. (1978), belongs to the technical indicators and provides signals when a stock is oversold or overbought. The RSI ranges from 0 to 100 and is calculated by

$$RSI_t = 100 - \frac{100}{1 + RS_t}, \qquad t = n+1, n+2, \ldots, T, \tag{3}$$

where

$$RS_t = \frac{\frac{1}{n}\sum_{i=1}^{n} \max\{P_{t-i+1} - P_{t-i}, 0\}}{-\frac{1}{n}\sum_{i=1}^{n} \min\{P_{t-i+1} - P_{t-i}, 0\}}, \quad t = n+1, n+2, \ldots, T, \tag{4}$$

in which P_t is the stock price at times t and n is the lookback period. We follow Wilder jr. (1978) by choosing a period of 14 days and calculate the daily RSI for every stock by using the R package *TTR* (Ulrich, 2019).

Measuring stock market sentiment is a difficult task. Simon and Wiggins (2001) find a significant forecasting power for S&P 500 futures returns in the Volatility Index (VIX) and the put-call ratio (PCR). The VIX - our third input feature - measures the implied volatility of the S&P 500 using options rather than stocks (see CBOE, 2019). The fourth and final feature is a simple ratio between the volume of all put and call options in the US equity market. Both of these indicators are provided by the Chicago Board Options Exchange on a daily basis.

To sum up, our full set of inputs now contains 32 features for every single trading day: 15 stock returns and 15 RSI (one for each stock) as well as the VIX and the PCR for the market sentiment. Since a recurrent neural network is trained the input layer needs sequences of data. We use information of the past 250 trading days hence one sequence contains 8000 data points (32 features $\times$ 250 days). How these features are put together so that the network can process them is further described in section 3.1.4.

3.1.3. Asset allocation strategies: generating the right targets

At this stage, we hypothesize that the features defined in the last section contain enough information to find the relationship between inputs and outputs. For these target outputs we stay in the realm of Markowitz (1952) and his efficient frontier. By estimating the vector of mean returns μ_t and the covariance matrix Σ_t at time t of a chosen portfolio we can solve the quadratic optimization problem with linear constraints for the optimal portfolio weights w_t

$$\begin{aligned} \max_{w_t} \quad & w_t'\mu_t - \frac{\gamma}{2} w_t'\Sigma_t w_t \\ \text{s.t.} \quad & w_t'\mathbf{1}_N = 1, \\ & w_t'\mathbf{I}_N > 0 \end{aligned} \tag{5}$$

where γ can be interpreted as the investor's risk aversion. $\mathbf{1}_N$ is a N-dimensional vector of ones and $\mathbf{I}_N$ indicates the $N \times N$ identity matrix. With the basic principle laid out, we suggest four different strategies in total, all being based on the idea of using the solution to the optimization problem in equation (5) with different values for the involved parameters at every point in time as the desired output for the network that is based

on the aforementioned 32 input features.

The first two are based on the **mean-variance** optimization described in equation (5). We compute the vector of in-sample weights w_t^{mv} with $\gamma = 1$ by applying a rolling window of 250 days for our sample estimates over the entire dataset (**"mv-r"**). By using an expanding window over all available data we are able to obtain more accurate estimates, especially for the expected return demonstrated by Merton (1980) (**"mv-e"**).

As shown by DeMiguel et al. (2009) estimating only the covariance matrix is one of the few approaches that is able to significantly outperform the $1/N$ benchmark, at least in one of their datasets. On this basis, we apply the in-sample **minimum-variance** weights w_t^{min} as targets for the last network layer giving us strategy three and four. This is an adaption of equation (5) where $\gamma = \infty$ and the mean return estimates are ignored:

$$\begin{aligned} \min_{w_t} \quad & w_t' \Sigma_t w_t \\ \text{s.t.} \quad & w_t' \mathbf{1}_N = 1, \\ & w_t' \mathbf{I}_N > 0 \end{aligned} \tag{6}$$

with the same 250 day rolling window (**"min-r"**) and an expanding window (**"min-e"**) for the sample estimates. Both mean-variance and minimum-variance optimizations are performed with the R package *quadgprog* (Turlach and Weingessel, 2019).

3.1.4. Training, validation and test sets

After features and targets are generated, we have 3238 trading days worth of data. This dataset is separated into three parts:

- day 1 to 1988 for learning,
- day 1989 to 2738 to validate the model architecture and
- the remaining 500 days for measuring the performance of our out-of-sample predictions

In the next step, we standardize each feature s by calculating its mean μ_s^{train} and standard deviation σ_s^{train} from the training data. All three sets with a combined length of T are then scaled by

$$x'_{s,t} = \frac{x_{s,t} - \mu_s^{train}}{\sigma_s^{train}}, \qquad \begin{aligned} & t = 1, 2, \ldots, T, \\ & s = 1, 2, \ldots, 32, \end{aligned} \tag{7}$$

where $x'_{s,t}$ is the standardized and $x_{s,t}$ the unscaled feature at time t. This operation

centers the feature values around 0 with a standard deviation of 1.

LSTM networks process sequences of input data. The generation of these sequences is shown in Figure 1. Utilizing information from approximately one year, we look back 250 trading days which starts the first sequence at time $t = 251$. Since we are predicting the in-sample portfolio weights our targets are taken from time t. For a dataset of length T we are rolling this window forward until we hit the last day $t = T$. Therefore the network's first layer has the dimension (lockback $\times$ features) 250×32 with 1737 sequences of training data, 500 sequences of validation data and 250 sequences for testing the out-of-sample performance. This process is applied to every single strategy with the only difference in the drawn targets.

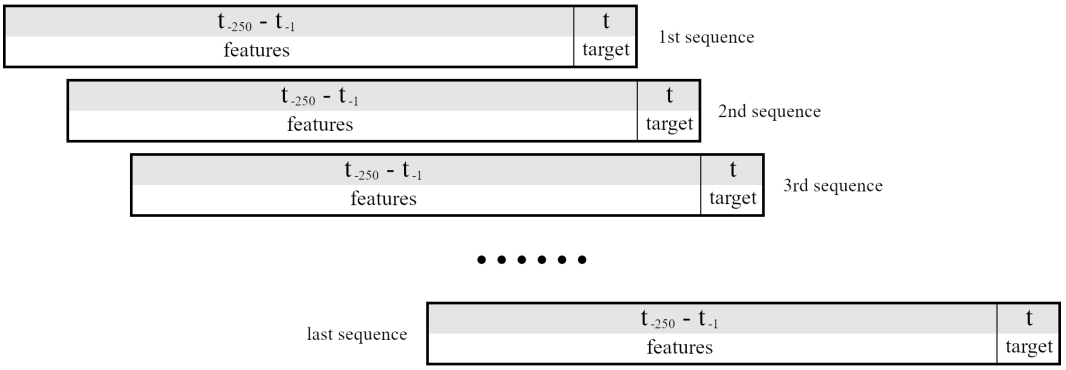


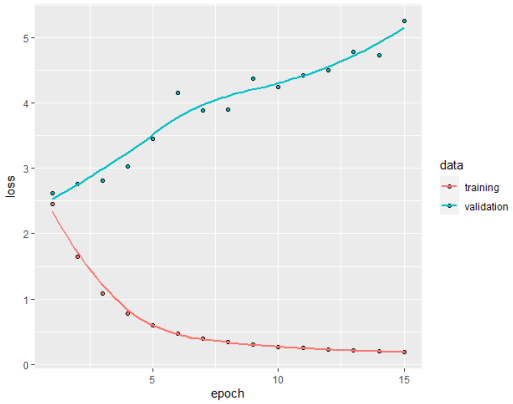
Figure 1 illustrates the method used to generate input layer sequences. At time t we use the target values as outputs and the features from the 250 days lookback period as network inputs, creating one sequence. This process is repeated until we hit the end of the dataset.

3.1.5. Model architecture

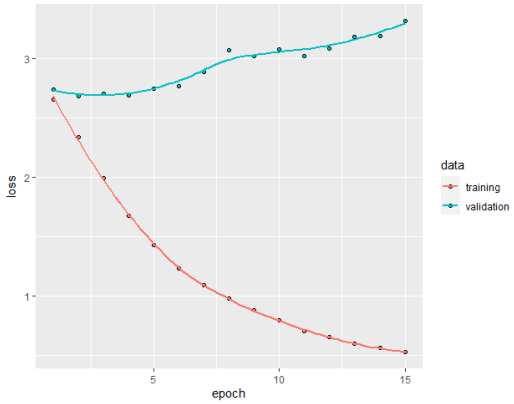
Building deep neural networks, even more sophisticated like LSTM, has become a simple task with open-source libraries like *keras*. These frameworks focus on a user-friendly and modular approach. The fundamental building blocks in neural networks - the *layers* - can be replaced or changed with ease, and as Chollet and Allaire (2018) pointed out, one can think of them as the LEGO bricks of deep learning. Designing networks is as easy as playing LEGO, but finding a good model that is able to map your network inputs to your targets and making accurate predictions is a challenging task. The topology of our LSTM is as follows:

- Input layer with 250 time steps and 32 features.
- LSTM layer with 7 neurons, dropout of 0.1 and L1 regularization set to 0.05.
- Output layer with 15 neurons and softmax activation function.

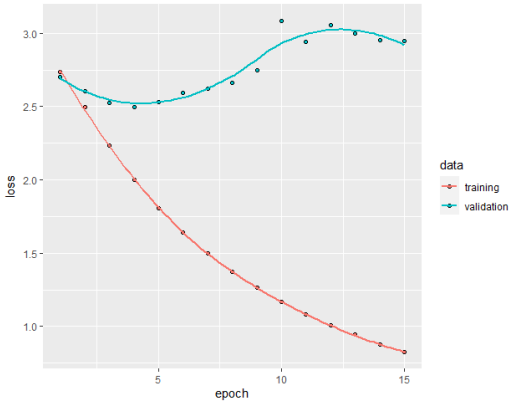
The validation process for this architecture is shown in figure 2 where we start with the same network as Fischer and Krauss (2018). Unfortunately this setup in combination with our dataset leads to extreme overfitting. That means the neural network is good at learning the training set by remembering patterns in the data, but does not have the ability to generalize well and performs bad on unseen examples. After reducing the number of neurons and the addition of L1 regularization we ended up with the architecture presented at the beginning of this subsection. The training is finally done after 8 epochs of training. Although our model is validated on the mean-variance model with a rolling window (**"mv-r"**), we use this network setup for each strategy.



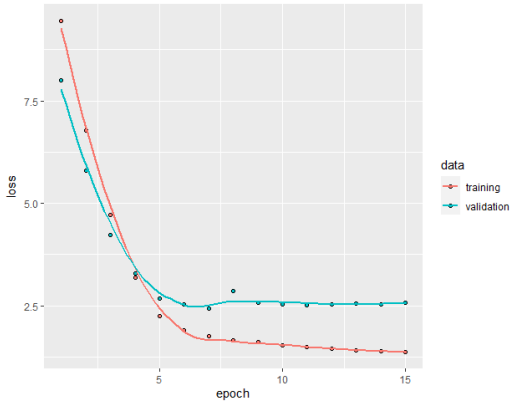
(a) units = 25, dropout = 0.1



(b) units = 10, dropout = 0.1



(c) units = 7, dropout = 0.1



(d) units = 7, dropout = 0.1, L1 = 0.05

Figure 2 shows the model validation process. On the first training we see instant overfitting(2a). In (2b) the neurons were reduced by 15 and a flatter validation curve is seen. Setting the units to 7 leads to a smaller validation loss and less overfitting (2c). The last step is adding L1 regularization to the weights showing same validation loss as before but limited overfitting (2d).

The last layer uses a softmax activation function which is the generalization of a sigmoid function. It outputs a vector that represents a probability distribution where the values range between 0 and 1. Furthermore, the sum of the vector equals one. This property puts the output of the last layer into the same shape as the weights of a fully invested portfolio with short selling constraints. With the same characteristic for the output and the targets described in section 3.1.3 the network can be trained properly. Formally, the softmax function is given by

$$softmax(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}, \tag{8}$$

where z are the unnormalized log probabilities predicted by a linear layer (see Goodfellow et al., 2016).

While the network learns it minimizes the loss between predictions $\hat{y}$ and true values y. For this process, we chose the Kullback–Leibler (KL) divergence loss function, a measure of how different a probability distribution is from another:

$$loss = \frac{1}{N} \sum_{t=1}^{T} \sum_{i=1}^{N} y_{t,i} \cdot \log \left(\frac{y_{t,i}}{\hat{y}_{t,i}} \right) \tag{9}$$

where y_i is the ith component of the target portfolio weights vector, $\hat{y}_i$ is the ith element of the probability distribution given by the softmax function in equation (8), N is the number of available assets and T describes the number of training samples.

Gradient descent is performed by the RMSprop optimizer with default learning rate and a batch size of 32. Here we follow Chollet and Allaire (2018) who pointed out that in most cases this is a good choice.

3.2. Performance evaluation

Our research goal is to study the performance of the aforementioned strategies against the naive diversification where one puts his wealth equally across all available asset. We implement this $1/N$ benchmark as a daily rebalanced portfolio. Referring to DeMiguel et al. (2009), we employ the Sharpe ratio and turnover to compare the examined strategies in terms of their out-of-sample performance. Furthermore, we calculate CAPM Alpha and Fama-French Three-Factor Alpha as a worthwhile extension. All three risk-adjusted measures are computed in absence and presence of transaction costs.

3.2.1. Sharpe ratio

The Sharpe ratio of strategy k is calculated as follows (see Sharpe, 1966):

$$\widehat{SR}_k = \frac{\hat{\mu}_k}{\hat{\sigma}_k}, \tag{10}$$

where $\hat{\mu}_k$ is the sample mean excess return and $\hat{\sigma}_k$ the sample standard deviation of the out-of-sample excess returns. For testing the statistical difference between one strategy and the benchmark we are interested in the hypothesis:

$$H_0 : \qquad \widehat{SR}_k - \widehat{SR}_l = 0, \tag{11}$$

where $\widehat{SR}_l$ is the benchmark's Sharpe ratio. This performance measure as well as the statistical procedure for testing this hypothesis is implemented by the R package *SharpeR* (Pav, 2020).

3.2.2. CAPM Alpha and Fama-French Three-Factor Alpha

In addition to the Sharpe ratio we propose two alternative risk-adjusted performance measures based on *ordinary least-squares* (OLS) regressions. For every strategy k we run two regressions to compute the CAPM Alpha (see Sharpe, 1964) and the Fama-French Three-Factor (TF) Alpha (see Fama and French, 1993), given by

$$R_{k,t} = \alpha^{CAPM} + \beta^{CAPM} R_t^{mkt} + \varepsilon_{k,t}, \tag{12}$$

$$R_{k,t} = \alpha^{TF} + \beta_1^{TF} R_t^{mkt} + \beta_2^{TF} R_t^{smb} + \beta_3^{TF} R_t^{hml} + \varepsilon_{k,t}, \tag{13}$$

where R_t denotes the out-of-sample portfolio return in time period t and R_t^{mkt}, R_t^{smb}, R_t^{hml} are the three Fama-French factors at time t, i.e market, size and value.

To compare the Alphas of strategy k and benchmark l given by the regression intercepts, we test for the following two hypotheses:

$$H_0 : \qquad \alpha_k^{CAPM} - \alpha_l^{CAPM} = 0, \tag{14}$$

$$H_0 : \qquad \alpha_k^{TF} - \alpha_l^{TF} = 0 \tag{15}$$

These hypothesis are examined on the basis of standard F-tests using the R package *car* (Fox and Weisberg, 2019), where we do correct for heteroskedasticity in the portfolio returns.

3.2.3. Portfolio turnover

For every implemented strategy and the $1/N$ benchmark we compute the portfolio turnover to get a sense of the amount of trading required. Here we follow DeMiguel et al. (2009) in calculating the average sum of the absolute value of the trades across N assets under strategy k over the entire test set of length T:

$$turnover = \frac{1}{T-1} \sum_{t=1}^{T-1} \sum_{j=1}^{N} (\mid \hat{w}_{k,j,t+1} - \hat{w}_{k,j,t^+} \mid), \tag{16}$$

where $\hat{w}_{k,j,t^+}$ is the portfolio weight *before* rebalancing at time $t+1$ and $\hat{w}_{k,j,t+1}$ is the desired portfolio allocation *after* rebalancing at time $t+1$. The absolute turnover is shown for the $1/N$ benchmark where all other strategies are reported relative to said equally weighted portfolio.

3.2.4. Transaction costs

To give a more practical view on the impact of trading, we present our findings before and after transaction costs. As proposed by Hsu et al. (2018), we compute the net out-of-sample portfolio return $R_{k,t}^{net}$ for strategy k at time t with N available assets, according to

$$R_{k,t+1}^{net} = (1 + \sum_{j=1}^{N} R_{j,t+1} \hat{w}_{k,j,t})(1 - c \times \sum_{j=1}^{N} \mid \hat{w}_{k,j,t+1} - \hat{w}_{k,j,t^+} \mid) - 1, \tag{17}$$

in which c represents the one-way transaction costs. As in equation (16) $\hat{w}_{k,j,t^+}$ denotes the optimal portfolio weights before rebalancing and $\hat{w}_{k,j,t+1}$ right after. The transaction cost-adjusted results reported in section 4 are calculated by setting $c = 0.0005$.

3.3. Data

We conduct our empirical research on a daily basis in the period from 01/11/2006 to 04/10/2019 given by the availability of CBOE's Put/Call ratios. Missing values were set to zero which seems appropriate for all features used. Table 2 gives an overview of the datasets considered.

The portfolio of 15 individual companies from the S&P 100 is presented in table 1 with the stock closing prices retrieved from Yahoo Finance. Both Volatility Index and Put/Call ratio were downloaded from the Chicago Board Options Exchange website. For measuring the CAPM and Three-Factor Alpha we obtained the returns on the market portfolio (mkt) as well as the "small minus big" (smb) and "high minus low" (hml) factors from Kenneth R. French's website. The risk free rate used to calculate excess returns is the simple daily rate that, over the number of tradings days in the month, compounds to the one-month nominal US T-bill rate. It is provided by the same dataset that contains the Fama-French factors.

Table 2: Overview of datasets

Data	**Time period**	**Source**
15 stocks from the S&P 100	01/11/2006 - 04/10/2019	Yahoo Finance
CBOE Volatility Index (VIX)	02/01/2004 - 02/09/2020	CBOE
CBOE Equity Put/Call ratio	01/11/2006 - 04/10/2019	CBOE
Fama/French 3 Factors	01/07/1926 - 30/06/2020	Kenneth French
Risk free rate	01/07/1926 - 30/06/2020	Kenneth French

4. Results

In this section, we present the results from our empirical study. The performance of four asset allocation optimizations is measured against a daily rebalanced $1/N$ benchmark, where stocks are weighted equally. We identify a strategy to be superior if it shows higher performance and the null hypothesis is rejected at the 5% significance level. Furthermore, to analyze the effects of trading involved, we display all metrics before and after transaction costs as well as the turnover relative to the equally weighted baseline calculated by equation (16). In each table, the strategies are listed in rows, while the performance measures are shown in columns. To reiterate the models described in section 3.1.3, the LSTM networks are trained on in-sample optimized weights by:

- mean-variance with rolling window (**"mv-r"**)
- mean-variance with expanding window (**"mv-e"**)
- minimum-variance with rolling window (**"min-r"**)
- minimum-variance with expanding window (**"min-e"**)

In table 3 the risk-adjusted measures prior to transaction costs, with the return calculation in equation (17) performed by setting $c = 0$, are displayed. In terms of Sharpe ratio, the baseline shows a positive value of 0.018218, where our mean-variance approaches are both negative with the rolling window being the worst. On the other hand, both minimum-variance strategies deliver a ratio roughly five times higher, but are not able to outperform the benchmark significantly. Looking at the CAPM Alpha we see almost the identical outcome with a greater than zero α for the equally-weighted portfolio and the models *"mv-r"* and *"mv-e"* showing the worst results with the latter one being "better". Again, the minimum-variance strategies produce the best outcomes but no significant superiority. The last column in table 3 provides the Fama-French Three-Factor (TF) Alpha. We can observe that the overall findings are identical to the CAPM Alpha and Sharpe ratio, showing a marginal positive α of 0.000063 for the baseline, weak performance of the mean-variance targets and a nearly ten times better, but not

significant TF Alpha for the *"min-r"* and *"min-e"* approaches compared to the $1/N$ benchmark. Interestingly, the extended window mean-variance strategy shows poorer results than the rolling window compared to the other key figures.

Table 3: Sharpe ratio, CAPM Alpha and TF Alpha (before transaction costs)

Strategy	**Sharpe ratio**	**CAPM Alpha**	**TF Alpha**
1/N	0.018218	0.000167	0.000063
mv-r	-0.020975	-0.000301	-0.000482
	(0.6622)	(0.1195)	(0.0528)
mv-e	-0.002740	-0.000065	-0.000509
	(0.8153)	(0.6713)	(0.2269)
min-r	0.096385	0.000829	0.000622
	(0.3842)	(0.1055)	(0.1343)
min-e	0.103323	0.000967	0.000755
	(0.3435)	(0.0868)	(0.1108)

Table 3 presents the daily Sharpe ratio, CAPM Alpha and Three-Factor (TF) Alpha for the 1/N benchmark and the strategies **before** transaction costs by setting $c = 0$ in equation (17). The values in parentheses show the p-value of the statistical difference between each strategy and the 1/N model.

Presented in table 4, the results for the net returns after transaction costs calculated by equation (17) with $c = 0.0005$ show the same outcome as table 3 with slightly weaker performance in each measure given the presence of costs for trading. The low turnover in the minimum-variance strategies contributes to a small but insignificant gain in performance against the naive benchmark.

Contrary to the findings of DeMiguel et al. (2009), pointing out the extreme turnover for all tested strategies, we can see in table 5 that minor trading is required compared to the $1/N$ benchmark. The only outlier being the *"mv-r"* strategy producing a turnover nearly two times higher than the baseline. Furthermore, the least amount of trading can be observed in the models relying on the expanding window. Figure 3 clearly shows that, in contrast to strategy *"mv-r"*, the weights in the three low trading strategies do not change over time. These results are in line with the relative turnovers shown in table 5. Furthermore, both mean-variance strategies selected the same top stocks but with different weights. The same behavior applies to the minimum-variance models with the distinction that the chosen stocks are different from *"mv-r"* and *"mv-e"*.

To further examine the reasons for the surprisingly low turnovers the LSTM network is trained without L1 regularization. The results of this model are presented in appendix

Table 4: Sharpe ratio, CAPM Alpha and TF Alpha (after transaction costs)

Strategy	**Sharpe ratio**	**CAPM Alpha**	**TF Alpha**
1/N	0.017785	0.000166	0.000059
mv-r	-0.021575	-0.000310	-0.000490
	(0.6609)	(0.1165)	(0.0511)
mv-e	-0.002894	-0.000068	-0.000512
	(0.8177)	(0.6732)	(0.2281)
min-r	0.096025	0.000826	0.000619
	(0.3837)	(0.1049)	(0.1335)
min-e	0.103032	0.000964	0.000752
	(0.3427)	(0.0862)	(0.1100)

Table 4 presents the daily Sharpe ratio, CAPM Alpha and Three-Factor (TF) Alpha for the 1/N benchmark and the strategies **after** transaction costs by setting $c = 0.0005$ in equation (17). The values in parentheses show the p-value of the statistical difference between each strategy and the 1/N model.

A with the risk-adjusted measures shown in tables 6 and 7. Table 8 provides the relative turnover and figure 4 shows the portfolio weights over time. Compared to the amount of trading involved in the original network we observe a more than tenfold raise in *"mv-e"* and *"min-r"* as well as an roughly five times increase in *"mv-r"* and *"min-e"*. With regards to the risk-adjusted measures, the new LSTM model shows inferior results and suffers an even higher decrease in performance when we take transaction costs into account. This is not unexpected given the much higher turnover. It is important to highlight the fact that the top stocks for each strategy are the same as the ones in the original network.

Table 5: Portfolio turnovers

Strategy	Turnover
1/N	0.008510
mv-r	1.940075
mv-e	0.658907
min-r	0.735232
min-e	0.648833

Table 5 shows the daily portfolio turnovers calculated by using equation (16). The first line reports the absolute turnover from the 1/N benchmark. For the optimization strategies, explained in section 3.1.3, the turnover is presented relative to the 1/N model.

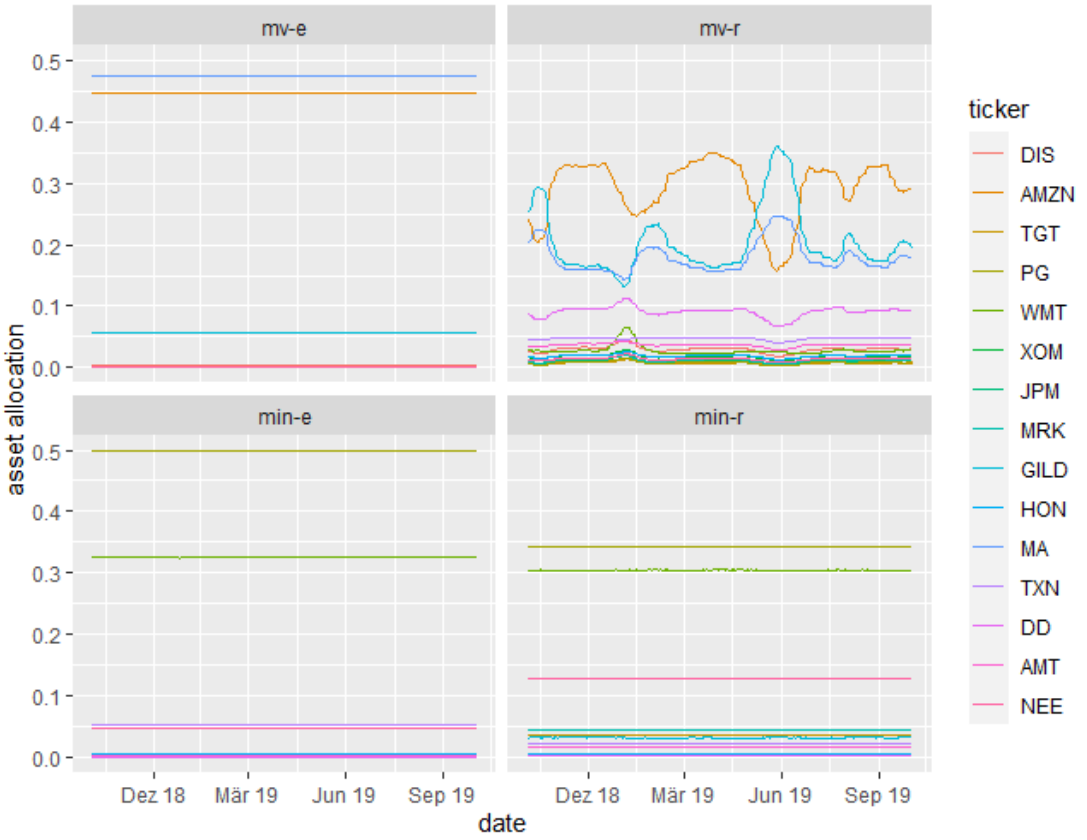


Figure 3 presents the portfolio weights over time for each strategy. The first row shows the mean-variance and the second row displays the minimum-variance approaches.

5. Conclusion

Building portfolios with optimal asset allocations is an extremely difficult undertaking. In this thesis, we designed a Long Short-Term Memory neural network capable of directly outputting the weights of a fully-invested portfolio with no short selling allowed. Given the sophisticated foundations of the mean-variance optimization, the network was trained on this technique. To test and compare the performance of our models, we choose the most naive benchmark that simply allocates a weight of $1/N$ to each of the N available assets. Although two of our four strategies showed better results in all risk-adjusted performance measures, none of them was statistically superior to the baseline. The most interesting finding is the extremely low turnover. In three strategies we can observe even less trading than in the benchmark where the only reallocation is due to the change in wealth from the previous day. After training the network without regularization the amount of trading increased drastically which opens the possibility to control the turnover by changing the settings for the regularizer. In line with previous studies, we showed how challenging it is to beat the naive diversification. Therefore, even more effort should be made to find the perfect portfolio for a risk-averse investor seeking for the best balance between risk and return.

A. LSTM without L1 regularization

Here we provide the results for the predictions made by a LSTM network trained without any regularization. The performance is evaluated with the same methodology as described in section 3.2.

Table 6: Sharpe ratio, CAPM Alpha and TF Alpha (before transaction costs, without regularization)

Strategy	**Sharpe ratio**	**CAPM Alpha**	**TF Alpha**
1/N	0.018218	0.000167	0.000063
mv-r	-0.037363	-0.000480	-0.000657
	(0.5356)	(0.0577)	(0.0284)
mv-e	-0.004275	-0.000094	-0.000534
	(0.8020)	(0.06406)	(0.02203)
min-r	0.093360	0.000798	0.000604
	(0.4028)	(0.1127)	(0.1365)
min-e	0.101579	0.000952	0.000743
	(0.3534)	(0.0908)	(0.1146)

Table 6 presents the daily Sharpe ratio, CAPM Alpha and Three-Factor (TF) Alpha for the 1/N benchmark and the strategies **before** transaction costs by setting $c = 0$ in equation (17). The values in parentheses show the p-value of the statistical difference between each strategy and the 1/N model. None of the four proposed approaches shows a significant outperformance over the baseline but positive results with the minimum-variance strategies compared to the other two.

Table 7: Sharpe ratio, CAPM Alpha and TF Alpha (after transaction costs, without regularization)

Strategy	**Sharpe ratio**	**CAPM Alpha**	**TF Alpha**
1/N	0.017785	0.000166	0.000059
mv-r	-0.040877	-0.000525	-0.000701
	(0.5133)	(0.0440)	(0.0205)
mv-e	-0.005849	-0.000123	-0.000563
	(0.7922)	(0.6099)	(0.2021)
min-r	0.088119	0.000753	0.000559
	(0.4335)	(0.1380)	(0.1687)
min-e	0.099738	0.000935	0.000726
	(0.3616)	(0.0963)	(0.1217)

Table 7 presents the daily Sharpe ratio, CAPM Alpha and Three-Factor (TF) Alpha for the 1/N benchmark and the strategies **after** transaction costs by setting $c = 0.0005$ in equation (17). The values in parentheses show the p-value of the statistical difference between each strategy and the 1/N model. The high turnover displayed in table 8 leads to a decrease in performance for all approaches in each risk-adjusted measure compared to the results before transaction costs (see table 6).

Table 8: Portfolio turnovers (without regularization)

Strategy	**Turnover**
1/N	0.008510
mv-r	10.36980
mv-e	6.766382
min-r	10.62438
min-e	4.096573

Table 8 shows the daily portfolio turnovers calculated by using equation (16). The first line reports the absolute turnover from the 1/N benchmark. For the optimization strategies, explained in section 3.1.3, the turnover is presented relative to the 1/N model. In line with previous studies, the amount of trading in strategies relying on expanding windows is considerably lower than the turnover in rolling window approaches.

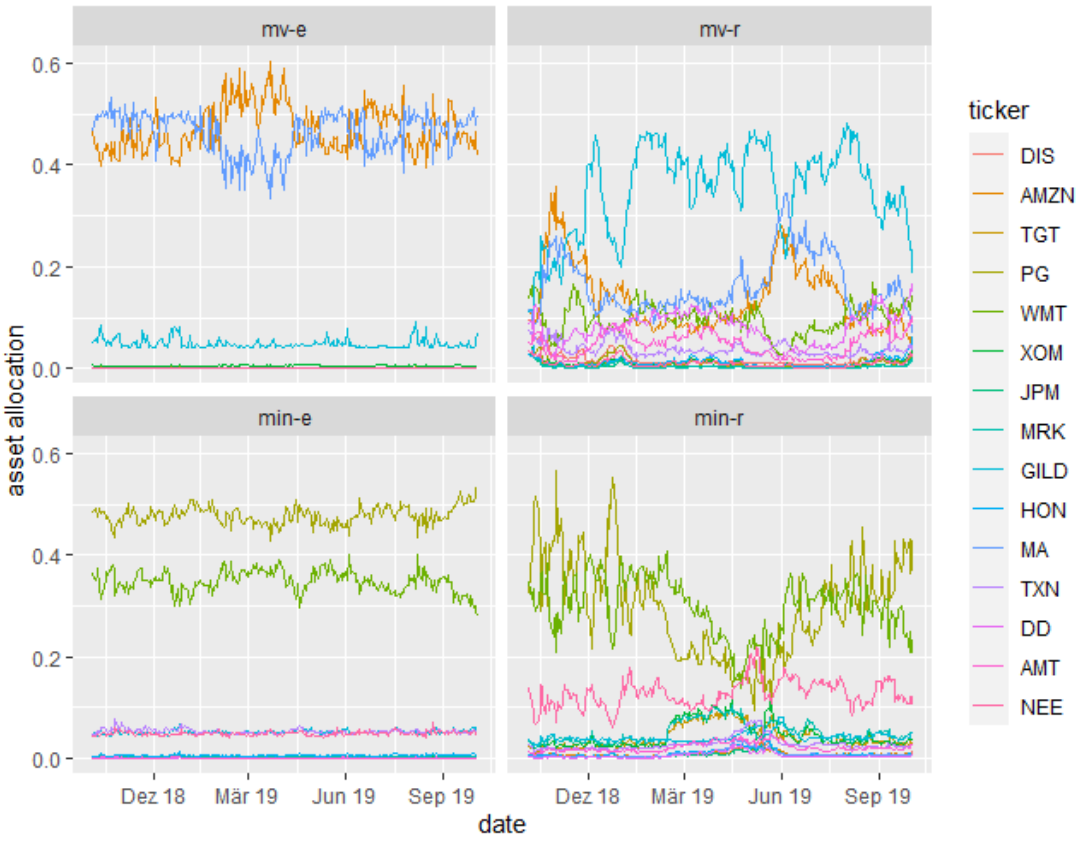


Figure 4 presents the predicted portfolio weights over time for each strategy. The model is trained without L1 regularization. The first row shows the mean-variance and the second displays the minimum-variance approaches.

References

Allaire, J. and Chollet, F. (2020). *keras: R Interface to 'Keras'*. R package version 2.3.0.0.

Best, M. and Grauer, R. (1991). On the Sensitivity of MeanVariance-Efficient Portfolios to Changes in Asset Means: Some Analytical and Computational Results. *Review of Financial Studies*, 4(2):315.

CBOE (2019). The CBOE Volatility Index-VIX, White paper. https://www.cboe.com/micro/vix/vixwhite.pdf. [Online; accessed 26-August-2020].

Cheng, P. and Liang, Y. (2000). Optimal Diversification: Is It Really Worthwhile? *Journal of Real Estate Portfolio Management*, 6(1):7–16.

Chollet, F. and Allaire, J. J. (2018). *Deep learning with R*. Manning, Shelter Island.

Chopra, V. K. and Ziemba, W. T. (1993). The effect of errors in means, variances, and covariances on optimal portfolio choice. *The Journal of Portfolio Management*, 19(2):6.

DeMiguel, V., Garlappi, L., and Uppal, R. (2009). Optimal Versus Naive Diversification: How Inefficient is the 1/N Portfolio Strategy? *Review of Financial Studies*, 22(5):1915–1953.

Fama, E. F. and French, K. R. (1993). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, 33(1):3–56.

Fischer, T. and Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669.

Fox, J. and Weisberg, S. (2019). *An R Companion to Applied Regression*. R package version 3.0-9.

Frankfurter, G. M., Phillips, H. E., and Seagle, J. P. (1971). Portfolio Selection: The Effects of Uncertain Means, Variances, and Covariances. *The Journal of Financial and Quantitative Analysis*, 6(5):1251.

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly, Beijing and Boston and Farnham and Sebastopol and Tokyo, second edition edition.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts and London.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hoseinzade, E. and Haratizadeh, S. (2019). CNNpred: CNN-based stock market prediction using a diverse set of variables. *Expert Systems with Applications*, 129:273–285.

Hsu, P.-H., Han, Q., Wu, W., and Cao, Z. (2018). Asset allocation strategies, data snooping, and the 1/N rule. *Journal of Banking & Finance*, 97:257–269.

Kalymon, B. A. (1971). Estimation Risk in the Portfolio Selection Model. *The Journal of Financial and Quantitative Analysis*, 6(1):559.

Klein, R. W. and Bawa, V. S. (1976). The effect of estimation risk on optimal portfolio choice. *Journal of Financial Economics*, 3(3):215–231.

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

Merton, R. C. (1980). On estimating the expected return on the market. *Journal of Financial Economics*, 8(4):323–361.

Michaud, R. O. (1989). The Markowitz Optimization Enigma: Is 'Optimized' Optimal? *Financial Analysts Journal*, 45(1):31–42.

Paiva, F. D., Cardoso, R. T. N., Hanaoka, G. P., and Duarte, W. M. (2019). Decision-making for financial trading: A fusion approach of machine learning and portfolio selection: Expert systems with applications, 115, 635-655. *Expert Systems with Applications*, 115:635–655.

Park, H., Sim, M. K., and Choi, D. G. (2020). An intelligent financial portfolio trading strategy using deep Q-learning. *Expert Systems with Applications*, 158:113573.

Pav, S. E. (2020). *SharpeR: Statistical Significance of the Sharpe Ratio*. R package version 1.2.1.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Sharpe, W. F. (1964). Capital Asset Prices: A Theory of Market Equilibrium under Conditions of Risk. *The Journal of Finance*, 19(3):425.

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

Simon, D. P. and Wiggins, R. A. (2001). SP futures returns and contrary sentiment indicators. *Journal of Futures Markets*, 21(5):447–462.

Song, Q., Liu, A., and Yang, S. Y. (2017). Stock portfolio selection using learning-to-rank algorithms with news sentiment. *Neurocomputing*, 264:20–28.

Turlach, B. and Weingessel, A. (2019). *quadprog: Functions to Solve Quadratic Programming Problems*. R package version 1.5-8.

Ulrich, J. (2019). *TTR: Technical Trading Rules*. R package version 0.23-6.

Wang, W., Li, W., Zhang, N., and Liu, K. (2020). Portfolio formation with preselection using deep learning from long-term financial data. *Expert Systems with Applications*, 143.

Wilder jr., J. W. (1978). *New concepts in technical trading systems*. Trend Research, Greensboro/N. C.

Xiong, R., Nichols, E. P., and Shen, Y. (2015). Deep Learning Stock Volatility with Google Domestic Trends. https://arxiv.org/pdf/1512.04916. [Online; accessed 19-September-2020].