Junior Management Science

journal homepage: www.jums.academy

Modern Approaches to Dynamic Portfolio Optimization

Philipp Schiele

Ludwig-Maximilians-Universität München

Abstract

Although appealing from a theoretical point of view, empirical assessments of dynamic portfolio optimizations in a mean-variance framework often fail to reach the high expectations set forth by analytical evaluations. A major reason for this shortfall is the imprecise estimation of asset moments and in particular, the expected return. This work levers recent advancements in the field of machine learning and employs three types of artificial neural networks in an attempt to improve the accuracy of the asset return estimation and the expected associated portfolio performances. After an introduction of the dynamic portfolio optimization framework and the artificial neural networks, their suitability for the considered application is analyzed in a two asset universe of a market and a risk-free asset. A comparison of the corresponding risk-return characteristics and those achieved using a more traditional exponentially weighted moving average estimator is subsequently drawn. While outperformance of the artificial neural networks is found for daily and monthly estimated returns, significance can only be established in the latter case, especially in light of trading costs. Multiple robustness checks are performed before an outlook for subsequent research opportunities is given.

Keywords: Portfolio optimization; machine learning; multilayer perceptron; convolutional neural network; long short-term memory.

1. Introduction

Since the establishment of Modern Portfolio Theory through the seminal work of Markowitz (1952) on portfolio selection, this field has seen a persistent rise of attention ever since. Indicated by both a vast body of literature as well as ever-growing interest from practitioners, the continued importance of ideal portfolio composition in the spirit of Markowitz is apparent, with the benefits of diversification and trade-offs between risk and return being cornerstones of modern investment practices.

A key implementational challenge to the originally proposed optimization problem remains as to how returns in financial markets can be forecast in order to make informed allocation decisions and avoid portfolio compositions that ex-post turn out to be suboptimal. Historically, the great difficulty of these forecasts has challenged both academia and practitioners alike. Generally, financial markets are known for a low signal-to-noise ratio, thereby making it hard to detect patterns within the data, to begin with. However, perhaps an even bigger challenge is posed by the constantly evolving and competitive nature of financial markets, which quickly incorporate the information contained in relationships having lead to an outperformance of some market participants, thereby changing the data generation process itself. As famously formulated in the random walk hypothesis (e.g. Cootner, 1964), many would even argue that forecasting financial returns is virtually impossible as prices are not predictable based on previous observations. This argument is closely linked to the efficient market hypothesis (e.g. Fama, 1970), a fundamental work of financial literature which in its strongest form states that markets immediately reflect all public and private information. Others, however, propose that markets are "efficiently inefficient" (e.g. Gârleanu & Pedersen, 2018), implying that at least minor information inefficiencies can persist, therefore potentially making prediction efforts worthwhile. Since Markowitz-style portfolio optimizations are highly sensitive to estimated future returns (Best & Grauer, 1991; Black & Litterman, 1992), uncovering even subtle and intricate patterns in financial data could indeed result in substantial differences in performance.

While the improvement of return forecasts is undoubtedly a major challenge, other fields have recently seen strong progress on tasks previously deemed hard by

DOI: https://doi.org/10.5282/jums/v6i1pp149-189

leveraging machine learning techniques. These include advances in medical applications like cancer detection (Esteva et al., 2017) or natural language processing (Radford et al., 2019). In particular, artificial neural networks seem to show continued improvements in model performance with increasing data sets (Sun, Shrivastava, Singh, & Gupta, 2017). While these models were already introduced by McCulloch and Pitts (1943), their recent resurgence is fueled by improved training algorithms and a sharp increase in computational power available to researchers (Livni, Shalev-Shwartz, & Shamir, 2014). Although in financial markets, the sheer amount of data generated on a daily basis is vast, the mentioned challenges unique to this field still have to be overcome in order to improve forecasts. Incidentally, the identification of complex patterns is one of the key strengths of artificial neural networks, which makes it natural to consider incorporating these techniques in the portfolio allocation context. The research objective is thus to investigate the usefulness of artificial neural networks in the context of dynamic portfolio optimization. More specifically, three types of artificial neural networks are being used to estimate the expected asset returns, followed by a subsequent assessment of the realized portfolio performances based on these estimates.

The remainder of the thesis is structured as follows: Chapter 2 reviews the literature concerning both dynamic portfolio optimization and artificial neural networks. Chapter 3 presents the methods and implementations used for the empirical analysis. In Chapter 4, the results of this empirical analysis are presented and discussed. Finally, Chapter 5 concludes the results of the thesis and presents an outlook for further research.

2. Literature Review

Since this thesis considers the use of neural networks in the context of dynamic portfolio optimization, both elements are introduced in this chapter. First, a brief overview of the considered portfolio optimization framework is given. Then, the different types of neural networks used for the succeeding analysis are reviewed, before the chapter concludes by formulating the research hypotheses.

2.1. Dynamic Portfolio Optimization

Modern Portfolio Theory is inevitably tied to Markowitz (1952), who introduced a framework considering a tradeoff between the expected return of a portfolio and its risk. Markowitz measures the risk of a portfolio by its variance, and thus this approach is also referred to as the *mean-variance* framework. In this setting, a portfolio is considered efficient if it has the highest expected return for a given variance, or equivalently, the lowest variance for a given return. Formally, let $\mathbf{w}$ be the vector of asset weights and let $\boldsymbol{\mu}$ be the vector of their expected returns. Further, let $\boldsymbol{\Sigma}$ be the covariance matrix of these assets[1]. To find an efficient portfolio, one can minimize the portfolio variance σ_p^2 for a given expected portfolio return $\bar{\mu}$:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \sigma_p^2 = \mathbf{w}^\intercal \boldsymbol{\Sigma} \mathbf{w} && \text{(1a)} \\ \text{s.t.} \quad & \boldsymbol{\mu}^\intercal \mathbf{w} \geq \bar{\mu} && \text{(1b)} \\ & \mathbf{1}^\intercal \mathbf{w} = 1 && \text{(1c)} \\ & (w_i \geq 0 \quad \forall i \in \{1, \dots, n\}) && \text{(1d)} \end{aligned}$$

This optimization problem can be stated as a quadratic program with linear constraints. The additional constraint 1c[2] ensures that the portfolio weights sum up to one. If the constraint 1d is added, short sales of assets are prevented. However, efficient portfolios can also be found by maximizing the expected portfolio return μ_p for a given level of portfolio variance $\bar{\sigma}$:

$$\begin{aligned} \max_{\mathbf{w}} \quad & \mu_p = \boldsymbol{\mu}^\intercal \mathbf{w} \\ \text{s.t.} \quad & \mathbf{w}^\intercal \boldsymbol{\Sigma} \mathbf{w} \leq \bar{\sigma} \\ & \mathbf{1}^\intercal \mathbf{w} = 1 \\ & (w_i \geq 0 \quad \forall i \in \{1, \dots, n\}) \end{aligned} \qquad (2)$$

Here, quadratically constrained linear programming can be used to solve the optimization problem. The set of efficient asset allocations that can be reached for varying $\bar{\sigma}$ or $\bar{\mu}$ values is referred to as the *efficient frontier*. Merton (1972) first suggested an analytical solution to finding these efficient portfolios in the case without the no short sales constraint 1d. However, additionally constrained settings often require an optimization for different risk or return targets to obtain the efficient frontier. The *optimal* portfolio is defined as the allocation for which an investors' indifference curve is tangential to the efficient frontier.

In this framework, only a single investment period is considered, making it *static*. However, it is of both theoretical and practical interest to optimize portfolios not only in a single period but *dynamically* in a multi-period setting, taking into account new information about the assets each period. Numerous contributions in this field have been made, such as Smith (1967), where at each point in time the portfolio is reevaluated based on the change in expected returns and covariances of the assets. The parameter estimation was performed by using a least-squares estimator, placing a higher weight on more recent data. The study considered a trade-off between the possibly inefficient allocation given the new moment estimates and costs involved when trading towards a new efficient allocation. The costs of trading thereby included

[1]Here and throughout the thesis, uppercase bold letters refer to matrices, lowercase bold letters to vectors and lowercase light letters to scalars.

[2]$\mathbf{1}$ represents the vector $(1, \dots, 1)^\intercal$.

direct costs such as brokerage fees and indirect costs in the form of forgone profits on deferred tax obligations when taxable gains are realized early.

Brown and Smith (2011) start from a dynamic portfolio optimization model under the assumption of frictionless markets and consider different heuristic strategies to improve portfolio performances when this assumption is relaxed. These heuristics include the consideration of transaction costs in repeated static optimizations as well as the iterative maximization of expected utilities at a future date until which the chosen allocation needs to be held (i.e., no readjustment of the portfolio is possible). In numeric experiments, the performance of these heuristic strategies for different utility functions and levels of transaction costs is assessed, concluding that the strategies perform well in many cases.

A more analytical approach is taken by D. Li and Ng (2000), who derive an optimal investment policy for an unconstrained multiperiod mean-variance investor in the absence of trading costs. This provides a deepened understanding of portfolio optimality in a dynamic setting and can possibly be extended to include market frictions and portfolio constraints.

Other fundamental contributions to the general approach of optimizing dynamic investments include Mossin (1968), Samuelson (1969), Dumas and Luciano (1991) and Elton and Gruber (1974).

However, many of these theoretical contributions do not consider the question of how to optimally estimate moments of the asset return distribution, with the necessity of this estimation becoming clear when looking at the optimization problems themselves. Therefore, any empirical work on Modern Portfolio Theory needs to estimate the two main components of the optimization problems, namely the expected asset returns and their covariance matrix. Focusing on the estimation of the expected returns, Michaud (1989) states that in many applications the sample mean is used to estimate the expected asset returns, which he assesses to be a suboptimal choice, thus arguing for more advanced estimators. A crucial shortcoming of the sample mean in the estimation of time series moments is the equality of weights given to each observation, irrespective of their recency. Assuming more recent observations are more similar to future ones, the *exponentially weighted moving average (EWMA)* provides a natural extension to the sample mean by assigning exponentially decaying weights and thereby embedding the assumption into the estimation process (Severini, 2017). With this estimator, the expected return at time t can be recursively calculated as

$$\hat{\mu}_t = \lambda \hat{\mu}_{t-1} + (1-\lambda) r_t \tag{3}$$

for $t > 1$, where r_t refers to the return at time t, μ is the expected return and λ is a smoothing factor in the interval $(0, 1)$. Given its simplicity and the described inherent advantage over the sample mean, the EWMA estimator forms the baseline to which the neural networks are compared.

2.2. Neural Network Expected Return Estimators

Since the main research object is to evaluate the usefulness of artificial neural network expected return estimators in a portfolio optimization context, this section introduces the three different network architectures used throughout the analysis. In an abstract sense, artificial neural networks, henceforth just referred to as neural networks, can be thought of as versatile mappings of input and output values. This was formalized in the universal approximation theorem (Cybenko, 1989; Hornik, 1991) which states that finitely sized neural networks can approximate many continuous functions arbitrarily closely. In particular, these mappings can be nonlinear in their parameters, which makes them useful for many applications such as natural language processing (Goldberg, 2016), image processing (Egmont-Petersen, de Ridder, & Handels, 2002), and time series forecasting (Hill, O'Connor, & Remus, 1996). The latter scope makes them especially useful for the estimation of portfolio moments and provides the foundation for the considered analysis. Neural networks can be used in supervised and unsupervised learning settings. In supervised learning, the neural networks seek to minimize the prediction error made based on a model trained on data which contains the set of input values, so-called features, and the desired output values. In the context of financial forecasting, inputs could include historical returns, fundamental or macroeconomic data, with the outputs often represented by the future returns or just a binary representation of their sign. In contrast, unsupervised learning is concerned with tasks where no desired output is presented to the models explicitly. The focus of this thesis lies on supervised learning neural network applications, which can further be divided into regression and classification tasks, depending on whether class memberships or continuous outputs are modeled (Alpaydin, 2010). A general introduction to financial time series forecasting using neural networks is given by Kaastra and Boyd (1996) and Azoff (1994), with recent applications including the improvement of time series momentum strategies (Lim, Zohren, & Roberts, 2019), the forecast of ETF returns (Liew & Mayster, 2017) and the development of a statistical arbitrage strategy using neural network predictions (Krauss, Do, & Huck, 2017).

2.2.1. Multilayer Perceptrons

One widely used and sometimes referred to as the "vanilla" neural network architecture (Hastie, Tibshirani, & Friedman, 2009) is the *multilayer perceptron (MLP)*. As schematically represented in Figure 1, it consists of one input layer, one or more hidden layers, and one output layer. The total number of layers in the network is denoted L. More specifically, the illustrated network has three input nodes, two hidden layers consisting of three nodes (or *units*) each and one output node. If two or more hidden layers are present, the network is referred to as a *deep* neural network (Bengio, 2009; LeCun, Bengio, & Hinton, 2015).

Following standard notation, the vector of input parameters is denoted as $\mathbf{x}$ and consists of the features $(x_1, \ldots, x_n)^\intercal$ and an additional bias term x_0 which can be seen as an intercept term, as it always takes on a value of one. For reasons of simplicity, this bias term is not depicted in the figures in this chapter. In the shown illustration, the input vector would thus be

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} . \tag{4}$$

The values $\mathbf{a}^{(2)} = (a_1^{(2)}, \ldots, a_m^{(2)})^\intercal$ are obtained by first multiplying the input vector with the weight matrix $\mathbf{\Theta}^{(1)} \in \mathbb{R}^{m\times(n+1)}$ which assigns a single weight to each input value for every node in the first hidden layer. $\mathbf{\Theta}_{ij}^{(1)}$ thereby refers to the weight applied to the connection between the i^{th} node in the first hidden layer and the j^{th} node in the input layer. In general, when two adjacent layers are considered, let $n+1$ be the number of the preceding layers' nodes (including the bias unit) and let m be the number of nodes in the succeeding layer. Likewise, j always refers to a node in the preceding layer and i to a node in the succeeding layer. Again, in the illustrative example this weight matrix would be

$$\mathbf{\Theta}^{(1)} = \begin{bmatrix} \Theta_{10} & \Theta_{11} & \Theta_{12} & \Theta_{13} \\ \Theta_{20} & \Theta_{21} & \Theta_{22} & \Theta_{23} \\ \Theta_{30} & \Theta_{31} & \Theta_{32} & \Theta_{33} \end{bmatrix} . \tag{5}$$

This yields $\mathbf{z}^{(2)} = \mathbf{\Theta}^{(1)}\mathbf{x}$, a vector of weighted inputs to which an activation function $g(z)$ is element-wise applied. For this purpose, McCulloch and Pitts (1943) proposed a binary threshold function modelled after neuron activations in the brain. This function is defined as

$$g_0(z) = \begin{cases} 0 & \text{for } z \leq 0 \\ 1 & \text{for } z > 0 \end{cases} . \tag{6}$$

Since then, many activation functions have been suggested, such as the sigmoid function[3] (sometimes also referred to as the logistic or soft step function)

$$g_1(z) = \sigma(z) = \frac{1}{1+e^{-z}} , \tag{7}$$

the hyperbolic tangent function

$$g_2(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} , \tag{8}$$

[3]Note that depending on the context, σ refers to the sigmoid activation function or the standard deviation.

or the rectified linear unit function (ReLU), introduced by Hahnloser, Sarpeshkar, Mahowald, Douglas, and Seung (2000) and again inspired by biological processes,

$$g_3(z) = \begin{cases} 0 & \text{for } z \leq 0 \\ z & \text{for } z > 0 \end{cases} = \max(z, 0) . \tag{9}$$

To add intuition on the functional principle of MLPs it should be noted that if the sigmoid function is used, each node in the hidden layer is a logistic regression on its input values, with the weights representing the regression coefficients. As in the MLP the activation of each node depends on all nodes in the previous layer, the hidden layers and the output layer are referred to as *fully connected* layers. As shown by Hayou, Doucet, and Rousseau (2018), the selection of an appropriate activation function needs to be performed under the consideration convergence performance, computational efficiency and the vanishing gradient problem, which is discussed below. Independent of the of activation function, the application of $g(z)$ on each element of $\mathbf{z}^{(2)}$ now yields $\mathbf{a}^{(2)}$. For each subsequent hidden layer k, again the previous nodes $\mathbf{a}^{(k-1)}$ and a bias term are multiplied by a weight matrix $\mathbf{\Theta}^{(k-1)}$ and the activation function is applied on the resulting vector $\mathbf{z}^{(k)}$ as shown in Equation 10:

$$\mathbf{a}^{(k)} = g(\mathbf{\Theta}^{(k-1)}\mathbf{a}^{(k-1)}) \tag{10}$$

For the output layer, $\mathbf{a}^{(L)}$, which again can consist of one or more nodes, the last hidden layer is multiplied by a final weight matrix $\mathbf{\Theta}^{(L-1)}$. Depending on the objective, different activation functions need to be applied here. For classification tasks, where output values in the interval $[0,1]$ represent probabilities of a positive classification, often sigmoid or softmax functions are used, depending on whether exactly two or more than two classes are to be differentiated. For regression tasks where output values can lie outside of the interval $[0,1]$, it is common to use no activation function (or equivalently, the linear activation function $g(z) = z$). Computing the output layer for a given input vector and weight matrices is referred to as *forward propagation* or a *forward pass*. As indicated by the arrows in Figure 1, the flow of information in an MLP is always directed from the input layer to the output layer. In particular, there are no cycles in this network architecture, which is why an MLP can be described as a *feedforward neural network*.

Backpropagation

Before a useful regression or classification can be performed on new input vectors using forward passes, the appropriate weight matrices have to be derived from training samples, which are sets of input vectors with known or *ground truth* output values denoted $\mathbf{y}$. The optimization of these weights is often achieved by employing the backpropagation algorithm, whose outline

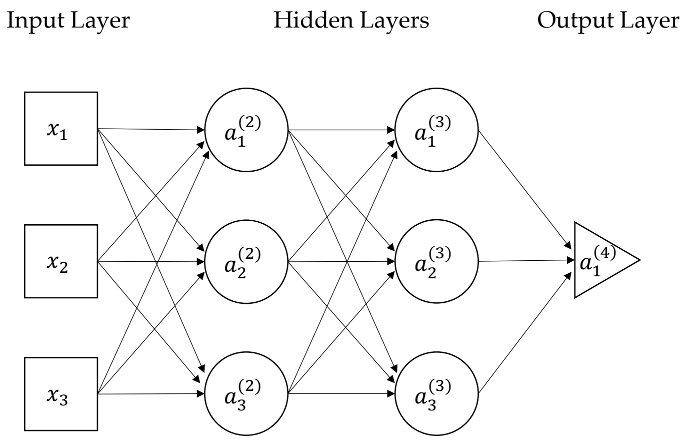

Figure 1: Schematic representation of an MLP

and origin is presented by Schmidhuber (2015). The underlying idea of backpropagation is to minimize the error made by the network on training data, starting with randomly initialized weights and iteratively adjusting them in the direction which decreases the overall error the most. A common analogy, among others used by F.-F. Li (2019), is to descend from a hill (point of high error) by only looking at the immediate surrounding and stepping in the direction which has the steepest downward slope (gradient) until the valley (a (local or global) minimum) is reached. For a single training example and randomly initialized weights, the activations of the output layer can be computed by performing a forward pass. Next, the error made using these weights in the output layer is calculated and denoted $J(\mathbf{\Theta})$. $\mathbf{\Theta}$ here refers to all individual weights from all layers, obtained by first unrolling all weight matrices $\mathbf{\Theta}^{(1)}, \dots, \mathbf{\Theta}^{(L-1)}$ into vectors and concatenating them. A typical error function for regressions is the *mean squared error function* (*MSE*, Equation 11) with the *categorical cross-entropy function* (*CCE*, Equation 12) often being used for classification tasks.

$$J_{MSE}(\mathbf{\Theta}) = \frac{1}{m}\sum_{\nu=1}^{m}(a_\nu^{(L)} - y_\nu)^2 \tag{11}$$

$$J_{CCE}(\mathbf{\Theta}) = -\sum_{\nu=1}^{m} y_\nu \log(a_\nu^{(L)}) \tag{12}$$

The derivative of this error with respect to $\mathbf{\Theta}$ is indicative of the necessary adjustments that need to be made in order to decrease the error. It is also referred to as the gradient of $\mathbf{J}$ and can be written as

$$\nabla \mathbf{J} = \frac{\partial J(\mathbf{\Theta})}{\partial \mathbf{\Theta}} = \begin{bmatrix} \frac{\partial J(\mathbf{\Theta})}{\partial \Theta_{10}^{(1)}} \\ \vdots \\ \frac{\partial J(\mathbf{\Theta})}{\partial \Theta_{m(n+1)}^{(L-1)}} \end{bmatrix}. \tag{13}$$

For each of the weights in the final layer, $\Theta_{ij}^{(L-1)}$, one can apply the chain rule as shown in Equation 14 in order to decompose the derivative of the error with respect to this weight into a change in $z_i^{(l)}$ caused by the initial change in $\Theta_{ij}^{(L-1)}$. This in turn changes $a_i^{(L)}$ through the activation function, thereby directly influencing the error term $J(\mathbf{\Theta})$.

$$\frac{\partial J(\mathbf{\Theta})}{\partial \Theta_{ij}^{(L-1)}} = \frac{\partial J(\mathbf{\Theta})}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial \Theta_{ij}^{(L-1)}} = \frac{\partial J(\mathbf{\Theta})}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial \Theta_{ij}^{(L-1)}} \tag{14}$$

Next, the derivatives of $J(\mathbf{\Theta})$ with respect to the weights in the penultimate layer, $\mathbf{\Theta}^{(L-2)}$, are needed. Again, using the chain rule, this derivative can be decomposed as follows:

$$\frac{\partial J(\mathbf{\Theta})}{\partial \Theta_{ij}^{(L-2)}} = \frac{\partial J(\mathbf{\Theta})}{\partial z_i^{(L-1)}} \frac{\partial z_i^{(L-1)}}{\partial \Theta_{ij}^{(L-2)}} = \frac{\partial J(\mathbf{\Theta})}{\partial a_i^{(L-1)}} \frac{\partial a_i^{(L-1)}}{\partial z_i^{(L-1)}} \frac{\partial z_i^{(L-1)}}{\partial \Theta_{ij}^{(L-2)}} \tag{15}$$

However, $\frac{\partial J(\mathbf{\Theta})}{\partial a_i^{(L-1)}}$ is not known but needs to be derived first. The node $a_i^{(L-1)}$ contributes to the total error through all its connections to the output layer. Since the layers are fully connected, the error induced in all output nodes needs to be considered if $a_i^{(L-1)}$ changes. A change in in $a_i^{(L-1)}$ introduces a change in $\mathbf{z}^{(L)}$ which again through the activation function changes $\mathbf{a}^{(L)}$, leading to a change in $J(\mathbf{\Theta})$. Thus, it follows that:

$$\frac{\partial J(\mathbf{\Theta})}{\partial a_j^{(L-1)}} = \sum_{i=1}^{m} \frac{\partial z_i^{(L)}}{\partial a_j^{(L-1)}} \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \frac{\partial J(\mathbf{\Theta})}{\partial a_i^{(L)}} \tag{16}$$

Note that the index of $a^{(L-1)}$ was changed from i to j as now not the layers $L-2$ and $L-1$, but $L-1$ and L are considered. Also note that the derivative with respect to $a_0^{(L-1)}$, the bias node, is omitted since its activation is fixed at a value of 1. With this, the derivative of the error with respect to the weights of all layers can be obtained. This is achieved by propagating the error from the output layer back to the currently analyzed layer through a recursive application of the single-layer error propagation introduced in Equation 16. This behavior is eponymous for the algorithm. There, the error can then be decomposed similarly to Equation 15. A generalized form of the derivative of the loss functions with respect to the weights for all previous layers $l = L-3, ..., 1$ is shown in Equation 17 which makes use of recursively propagating the error from the output layer to layer l by making use of Equation 18.

$$\frac{\partial J(\mathbf{\Theta})}{\partial \Theta_{ij}^{(l)}} = \frac{\partial J(\mathbf{\Theta})}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial \Theta_{ij}^{(l)}} = \frac{\partial J(\mathbf{\Theta})}{\partial a_i^{(l+1)}} \frac{\partial a_i^{(l+1)}}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial \Theta_{ij}^{(l)}} \tag{17}$$

$$\frac{\partial J(\mathbf{\Theta})}{\partial a_j^{(l+1)}} = \sum_{i=1}^{m} \frac{\partial z_i^{(l+2)}}{\partial a_j^{(l+1)}} \frac{\partial a_i^{(l+2)}}{\partial z_i^{(l+2)}} \frac{\partial J(\mathbf{\Theta})}{\partial a_i^{(l+2)}} \tag{18}$$

At this point, the complete gradient vector $\nabla\mathbf{J}$ is known and thus the weight vector $\mathbf{\Theta}$ can be adjusted to reduce the error. This is done by an incremental change in the negative direction of the gradient. The improved weight vector, denoted $\mathbf{\Theta}^+$, is thus given by:

$$\mathbf{\Theta}^+ = \mathbf{\Theta} - \alpha\nabla\mathbf{J} \tag{19}$$

An iterative application of forward propagations and backpropagations is used to further improve weights until ideally the gradient vector approaches $\mathbf{0}$ and a minimum of the error function is reached. The constant $\alpha > 0$ is referred to as the *learning rate* and has to be chosen based on a trade-off between convergence speed and convergence probability (Bengio, 2012). In order to simplify the notation, the description of the backpropagation algorithm above only focuses on one training example. For a set of o training samples, $\{(x^{(1)}, y^{(1)}, ..., x^{(o)}, y^{(o)}\}$, the overall gradient $\nabla\mathbf{J}_{total}$ is defined as the arithmetic mean of all individual sample gradients:

$$\nabla\mathbf{J}_{total} = \frac{1}{o}\sum_{p=1}^{o} \nabla\mathbf{J}_p \tag{20}$$

Gradient Descent

Starting from a random initialization and repeatedly updating the weights using the overall gradient $\nabla\mathbf{J}_{total}$ until convergence is reached is referred to as *(batch) gradient descent (GD)*. Algorithm 1 gives an overview of the gradient descent algorithm, making use of both forward propagation and backpropagation.

```
**Data:** Training samples $\{(x^{(1)}, y^{(1)}, ..., x^{(o)}, y^{(o)}\}$
Randomly initialize $\mathbf{\Theta}$
**while** *not converged* **do**
    Set $\nabla\mathbf{J} = \mathbf{0}$
    **for** $i = 1$ *to* $o$ **do**
        Set $\nabla\mathbf{J}_i = 0$
        Set $\mathbf{a}^{(1)} = \mathbf{x}^{(i)}$
        Perform forward propagation to compute
          $\mathbf{a}^{(l)}$ for $l = 2, 3, ..., L$
        **for** $l$ = *L-1 to 1* **do**
            Obtain $\frac{\partial J(\mathbf{\Theta})}{\partial \mathbf{a}^{(l+1)}}$ by backpropagation the
              error
            Compute $\frac{\partial J(\mathbf{\Theta})}{\partial \mathbf{\Theta}^{(l)}}$ and concatenate to $\nabla\mathbf{J}_i$
        **end**
        $\nabla\mathbf{J} = \nabla\mathbf{J} + \frac{1}{o}\nabla\mathbf{J}_i$
    **end**
    $\mathbf{\Theta} = \mathbf{\Theta} - \alpha\nabla\mathbf{J}$
**end**
```

Algorithm 1: Gradient descent

An alternative implementation exists where the weights are not updated based on the cumulative gradient of all samples, but rather after each individual sample based on the gradient as measured by this sample alone. This method is called *stochastic gradient descent (SGD)*. The name refers to the stochastic approximation of the overall gradient descent algorithm through repeated adjustments based on the single gradients. This approximation is often supported by shuffling the data set first in order to avoid clustering. There are several advantages and disadvantages to both GD and SGD, such as the stable gradient of GD and more memory efficient implementations of SGD (since only one sample needs to be held in memory at any point in time, not the complete data set). Thus, for practical applications oftentimes *mini-batch gradient descent* is used, which aims to combine the advantages of GD and SGD without introducing too many of the disadvantages. In mini-batch gradient descent, the weights are updated after the gradient of a subset of n training examples is computed, with $1 < n < o$. Its main advantages are a more stable gradient compared to SGD while still, only the samples of the current batch need to be held in memory. On the other hand, one drawback is that the batch size is an additional variable which needs to be chosen by the researcher. An extended overview of the introduced gradient descent variations is given by Ng

(2019a).

After choosing a subtype of the gradient descent algorithm, the weights of the MLP network can be optimized using the training data and predictions can subsequently be made using forward passes on new samples, thereby completing the functional principle of the MLP.

2.2.2. Convolutional Neural Networks

Another class of neural networks is the *convolutional neural network (CNN)*. In their current form, CNNs were pioneered by LeCun et al. (1989, 1990), with a comprehensive introduction being provided by Lecun, Bottou, Bengio, and Haffner (1998). Until today, following their original domain of application, CNNs are most commonly used in the analysis of image data (Ciresan, Meier, & Schmidhuber, 2012; Lawrence, Giles, Ah Chung Tsoi, & Back, 1997). However, their applications also extend beyond image data to tasks like natural language processing (Collobert et al., 2011; Grefenstette, Blunsom, de Freitas, & Hermann, 2014; Kalchbrenner, Grefenstette, & Blunsom, 2014; Kim, 2014) and even time series forecasting (Borovykh, Bohte, & Oosterlee, 2017). As shown in Figure 2, CNNs differ from MLPs through the inclusion of oftentimes multiple and alternating *convolutional layers* and *pooling layers* and usually a single *flatten layer*, which is further discussed below. In univariate time series applications, the input vector $\mathbf{x}$ represents the individual observations of the time series. Multivariate time series or gray-scale image data, on the other hand, require the input layer to be a two-dimensional matrix, with colored images or video data requiring even higher dimensional inputs.

Convolutional Layers

The convolutional layers of a CNN are at the core of its functional principle. As shown in Figure 2, these layers take subsets of the input layer and apply a *filter* (also called *kernel*) to them, resulting in a weighted sum of these input values. This aggregation process is referred to as a *convolution* and is intended to extract higher-level features from the input while retaining spatial structures. For the more general case with a two dimensional input, let this input layer be a matrix $\mathbf{X}$ with dimensions $[k \times l]$. Further, let the kernel be the weight matrix $\mathbf{\Theta}^{(1)}$ with dimensions $[m \times n]$. Convolutions usually are performed on a central element and its surroundings, thus m and n are uneven in these cases. In the one-dimensional example depicted in Figure 2, the dimensions of the input are $[8 \times 1]$ with a kernel size of $[3 \times 1]$. The kernel is thus given by:

$$\mathbf{\Theta}^{(1)} = \begin{bmatrix} \Theta_{11} \\ \Theta_{21} \\ \Theta_{31} \end{bmatrix} \tag{21}$$

Define $\mathbf{X}_{sub}$ as a subset of $\mathbf{X}$ with the shape of the kernel $[m \times n]$. For this subset, the scalar output of a convolution is given by

$$z_{k_i l_j} = \sum_{\nu=1}^{m} \sum_{\eta=1}^{n} \Theta^{(1)}_{\nu\eta} X_{sub,\nu\eta} = \mathbf{1}^{\intercal}(\mathbf{\Theta}^{(1)} \odot \mathbf{X}_{sub})\mathbf{1} \tag{22}$$

with $\odot$ denoting the Hadamard product. A convolution, therefore, is the sum of the elements resulting from an element-wise multiplication of $\mathbf{X}_{sub}$ and the kernel $\mathbf{\Theta}^{(1)}$[4]. The indices of the output $z_{k_i l_j}$ thereby refer to the position of the central element of $\mathbf{X}_{sub}$ within $\mathbf{X}$. While theoretically conceivable, it is uncommon to apply a nonlinear activation function to this output, thus setting $a_{k_i l_j} = z_{k_i l_j}$. The full convolutional layer is obtained by repeating the convolution process for each possible subset $\mathbf{X}_{sub}$ in $\mathbf{X}$. More specifically, starting in one corner of $\mathbf{X}$, after each convolution the filter is offset by s in the horizontal direction, where s is referred to as the *stride* parameter. This is repeated until no further horizontal shifting is possible. Then, the kernel is reset to its original horizontal position and shifted by s in the vertical direction before starting another horizontal pass. This is repeated until the end of the input matrix is reached. This "serpentine" shifting pattern of the kernel can also be visualized as a typewriter which moves from left to right until the end of the line is reached and it starts again from the left in the next line. It is important to note here that in contrast to a fully connected layer the spatial structure in the input layer is contained during the convolution since one node in the output layer only contains information from the surrounding nodes in the input layer. More specifically, the same weights are used whenever the filter is moved across the input, which is referred to as *weights sharing*. In the one-dimensional case, a horizontal shift is not possible, so only vertical shifts take place. Thus, as shown in Figure 2, for a stride value of $s = 1$, after the initial convolution, the filter can be shifted five times vertically. The size of the convolutional layer is therefore given as $[6 \times 1]$. In general, in a one-dimensional setting the length of the convolutional layer is given by

$$k_a = \left\lfloor \frac{k-m}{s} \right\rfloor + 1\,. \tag{23}$$

However, for some applications, it is desirable to have convolutional layers with the same shape as the input layer. In these cases, the input matrix can be extended by padding inputs with a value of zero (Goodfellow, Bengio, & Courville, 2016). As again shown in Figure 2, different kernels can be used in order to extract different higher-level features from the input, thus resulting in another dimension for the output of the convolutional layer.

[4]In some applications the kernel is flipped both horizontally and vertically to differentiate a convolution from a cross-correlation. While results would vary for a given, non-symmetric kernel, this has no effect in a CNN since the filter weights are not given but learned by the model.

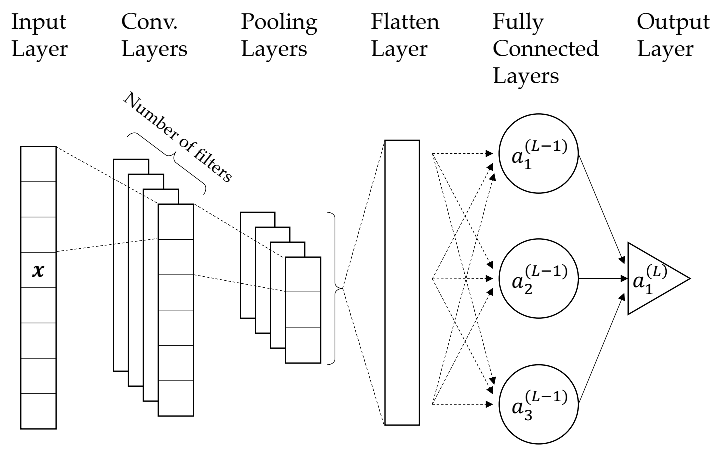

Figure 2: Schematic representation of a CNN

Pooling Layers
After a convolutional layer, often a pooling layer is used in order to reduce the dimensionality of the layers. Since CNNs often have a large input matrix, this reduction in dimensionality can lead to a significant improvement in the training speed of the model and also reduces its number of trainable parameters. The underlying idea is to aggregate neighboring nodes, ideally capturing most of their information content while reducing the layer dimensions. Similar to the convolutions, an aggregation function is applied to subsets of nodes of the previous layer with size $[m \times n]$, where m and n can be different from the filter size of previous layers. Popular aggregations include the arithmetic mean (*average pooling*) or the maximum (*max pooling*) of these nodes (Goodfellow et al., 2016). Again, this subset is moved along the previous layer by stride increments of s, which is often chosen such that the subsets do not overlap. In Figure 2 the aggregation subsets are chosen to have size $[2 \times 1]$ with $s = 2$. This reduces the size of the layers by 50% from $[6 \times 1]$ to $[3 \times 1]$.

Flatten Layer
In a CNN the number of filters used determines the number of resulting convolutional layers. Turning again to Figure 2, it is apparent that the number of layers also persists when pooling layers are applied. In order to use the higher-level features extracted through convolutions and pooling, a single matrix needs to be formed based on the forwardmost convolutional or pooling layer. This is performed by a flatten layer, which effectively concatenates all matrices of the preceding layer.

Fully Connected Layers
The fully connected layers can finally take the extracted higher-level features from the previous layers and model their dependencies, just like a discrete MLP network would if these features were provided externally. This behavior makes CNNs especially useful for extensive data sets with many potential features, since learning these features is part of the model training, thus requiring little domain knowledge for manual feature engineering (Kang, Ye, Li, & Doermann, 2014). Rather, the network itself can first find higher-level features and then train an MLP network on them.

Gradient Descent
Like MLP networks, CNNs are often trained from a random initialization using the gradient descent algorithm. Thus, after a forward pass, as described above, was performed, the error made using these weights is computed. For the fully connected layers, the derivative of the error with respect to their weights can again be obtained through the application of Equation 17. Arriving at the flatten layer, no derivative with respect to weights needs to be computed since this layer is merely a reshaping of the preceding nodes and it has no impact on the gradient. Since pooling layers also do not have weights which are learned by the model, again no derivative with respect to their weights can be computed. They do, however, affect the composition of the error through the way they aggregate the nodes of the previous layer. Thus, when average pooling is used, the error also has to be equally split to the nodes of the previous layer. For a subset size of $[m \times n]$, the error is thus multiplied by $\frac{1}{mn}$ when passing it to the previous nodes. For max pooling layers, the full error is propagated to the "winning" node of the forward pass (i.e., the node with the maximum activation, the value of which was used by the max pooling node) and an error of 0 is propagated to all other nodes, since their activations eventually had no effect on the error. Next, the derivative

of the error with respect to the weights of a convolutional layer is considered. Since a filter shares the same weights when moved across the input layer, it is intuitive to see that a single weight in a filter contributes to the total error through multiple paths. Thus, the total contribution of a weight in a filter is given by the sum of the derivatives of the backpropagated errors over all nodes of the convolutional layers' output nodes. Since convolutional layers are not restricted to succeed the input layer, the output of the convolutional layer is denoted $\mathbf{a}^{(l)}$ and its inputs $\mathbf{a}^{(l-1)}$. Further, let $[k^{(l-1)} \times l^{(l-1)}]$ denote the dimensions of $\mathbf{a}^{(l-1)}$, with $[k^{(l)} \times l^{(l)}]$ denoting the dimensions of $\mathbf{a}^{(l)}$. Then, the total contribution of weight $\Theta_{ij}^{(l-1)}$ is given by

$$\frac{\partial J(\mathbf{\Theta})}{\partial \Theta_{ij}^{(l-1)}} = \sum_{\nu=1}^{k^{(l)}} \sum_{\eta=1}^{l^{(l)}} \frac{\partial J(\mathbf{\Theta})}{\partial a_{\nu\eta}^{(l)}} \frac{\partial a_{\nu\eta}^{(l)}}{\partial z_{\nu\eta}^{(l)}} \frac{\partial z_{\nu\eta}^{(l)}}{\partial \Theta_{ij}^{(l-1)}} . \tag{24}$$

In case there are other layers preceding the convolutional layer, also the derivative of the error with respect to the input layer $\mathbf{a}^{(l-1)}$ needs to be obtained. Again, each node in this layer can contribute to the overall error through multiple paths since moving a filter across the input layer exposes the node to different weights in the filter matrix. The total contribution of node $a_{ij}^{(l-1)}$ is again the sum over all these paths through the different weights of the filter, as shown in Equation 25.

$$\frac{\partial J(\mathbf{\Theta})}{\partial a_{ij}^{(l-1)}} = \sum_{\nu=1}^{k^{(l)}} \sum_{\eta=1}^{l^{(l)}} \frac{\partial z_{\nu\eta}^{(l)}}{\partial a_{ij}^{(l-1)}} \frac{\partial a_{\nu\eta}^{(l)}}{\partial z_{\nu\eta}^{(l)}} \frac{\partial J(\mathbf{\Theta})}{\partial a_{\nu\eta}^{(l)}} \tag{25}$$

Since convolutional layers are not fully connected, the sparsity of the connections results in $\frac{\partial z_{\nu\eta}^{(l)}}{\partial a_{ij}^{(l-1)}} = 0$ for unconnected nodes.

With that, the error can be propagated back through pooling and convolutional layers, and thus the gradient vector $\nabla \mathbf{J}$ can be obtained. The gradient descent algorithm can then be applied, as shown in Algorithm 1.

2.2.3. Recurrent Neural Networks

While MLPs and CNNs evaluate each sample of inputs and outputs independently from others, for specific tasks, it would be beneficial to model a dependence structure between samples. In fact, sequential tasks like the modeling of the next word based on parts of a sentence or a time series prediction often require information about multiple preceding samples. One approach would be to feed multiple previous values to an MLP network. However, one could also conceptualize to only feed one sample at a time to a network which itself keeps information about previous samples. This is where *recurrent neural networks (RNNs)* emerge. As insinuated in Figure 3, at every step t in the training process, the inputs of an RNN do encompass not only the original input features $\mathbf{x}_t$ at that point, but also information about previous samples.

Elman Networks

Like for most neural network architectures, many implementations of RNNs exist. The specific network depicted in Figure 3 and described in the following was suggested by Elman (1990) and is thus referred to as the *Elman network*. In this network, information is passed from $t-1$ to t by providing the activations of the hidden layer as further inputs to the network.

Mathematically, at each time step t, this is achieved by first multiplying a weight matrix $\mathbf{\Theta}_1$ by the input vector $\mathbf{x}_t$. Next, the weight matrix $\mathbf{\Theta}_2$ is multiplied by the previous samples' hidden layer activations $\mathbf{a}_{t-1}^{(L-1)}$. Both results are then added element-wise to obtain the inputs for the hidden layer, $\mathbf{z}_t^{(L-1)}$. Like in any neural network, in order to obtain the hidden state, an activation function $g(z)$ is applied to each element of $\mathbf{z}_t^{(L-1)}$, as shown in Equation 26. While the dimensions of $\mathbf{x}_t$ and $\mathbf{a}_{t-1}^{(L-1)}$, i.e., the input vector and previous activations, can differ, it is required for $\mathbf{\Theta}_1$ and $\mathbf{\Theta}_2$ to match in their first dimension such that the resulting vectors can be added element-wise.

Based on the hidden layer $\mathbf{a}_t^{(L-1)}$, the output layer $\mathbf{a}_t^{(L)}$ can again be computed similar to the MLP case, namely by multiplying the weight matrix $\mathbf{\Theta}_3$ by the hidden activations and applying an activation function.

$$\mathbf{a}_t^{(L-1)} = g(\mathbf{\Theta}_1 \mathbf{x}_t + \mathbf{\Theta}_2 \mathbf{a}_{t-1}^{(L-1)}) \tag{26}$$

$$\mathbf{a}_t^{(L)} = g(\mathbf{\Theta}_3 \mathbf{a}_t^{(L-1)}) \tag{27}$$

For the next time step $t+1$, the hidden layer of t, $\mathbf{a}_t^{(L-1)}$, is combined with a new input vector $\mathbf{x}_{t+1}$ and with the same weights $\mathbf{\Theta}_1$, $\mathbf{\Theta}_2$ and $\mathbf{\Theta}_3$ again the hidden and output layer can be computed. This process is repeated iteratively until the end of the sequence is reached.

In order to keep Figure 3 easily understandable, vectors, matrices and single connections between them were drawn. However, just like in the MLP case in Figure 1, each line is fully connecting all nodes of one layer to all nodes of the subsequent layer. Further, it should be noted that the activation functions $g(z)$ of the hidden and output layer do not necessarily have to be identical. This is especially relevant as the activation function of the output layer often is determined by the application of the neural network.

Considered at each point in time, simple RNNs usually consist of only one hidden layer. The *deepness* of this network architecture rather comes from the possibility to access information from previous observations. This is achieved by first performing a forward pass as described above, starting at the first time step. Since for the first time step, no previous activations are present, a randomly initialized vector or a vector of zeros can be used. This

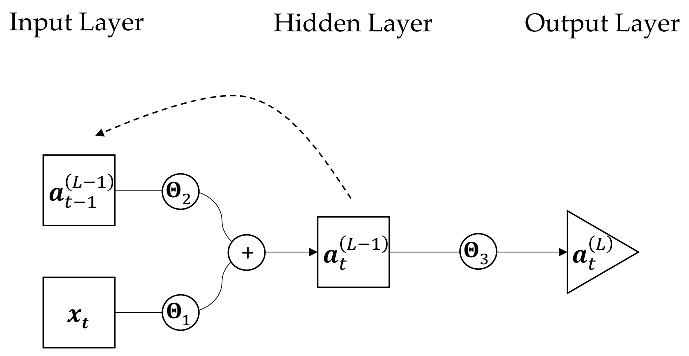

Figure 3: Schematic representation of an RNN

forward pass can be conceptualized by chaining copies of Figure 3, only adjusting the time index at each point. This is also called unfolding the network (Goodfellow et al., 2016). Then, the error at each time step can be computed, resulting in a loss of $J(\mathbf{\Theta})$. Through propagating the error back from the final to the first observation and applying the chain rule as shown in Equation 17, $\frac{\partial J(\mathbf{\Theta})}{\mathbf{\Theta}_1}$, $\frac{\partial J(\mathbf{\Theta})}{\mathbf{\Theta}_2}$ and $\frac{\partial J(\mathbf{\Theta})}{\mathbf{\Theta}_3}$ are obtained. The weight matrices are then adjusted before another pass is started. Since this retains the time dimension of the observations, this method of training is referred to as *backpropagation through time (BPTT)*, which was proposed by Werbos (1988), Mozer (1989) as well as Robinson and Fallside (1987).

Another popular simple RNN network was introduced by Jordan (1997), with the major differentiating factor being that not the previous hidden layer $\mathbf{a}_{t-1}^{(L-1)}$, but the previous output layer $\mathbf{a}_{t-1}^{(L)}$ is provided as an additional input at time t.

Long Short-Term Memory Networks

RNNs allow for information to persist in a network between observations. However, if the relevant information was not contained in a recent observation, but rather many observations before, simple RNNs can have great difficulty to represent these dependencies. Although there is no theoretical limit to how far back the information in these RNNs can persist, in empirical applications it has been difficult to model long term dependencies with them (Bengio, Simard, & Frasconi, 1994). The so-called *vanishing gradient problem*, as analyzed by Hochreiter (1991) and Hochreiter, Bengio, Frasconi, and Schmidhuber (2001), was found to play a major role in this restriction. Vanishing gradients describe a phenomenon where the gradient continuously decreases with the distance to the output layer. One example of this phenomenon can be considered where the chain rule is applied repeatedly by the backpropagation algorithm, as described in Equation 18. The derivative of many common activation functions, $\frac{\partial a}{\partial z} = g'(z)$, is bound to the range $(0, 1)$. The derivative of the popular sigmoid function (7) is even bound to the range $(0, \frac{1}{4})$. Thus, when these activation functions are used, and the error is propagated back many layers, the multiplicative linkage of these derivatives tends towards zero exponentially. One possible mitigation can be to use activation functions which do not exhibit this property, such as the ReLU function (9). However, while here the derivative equals one for $z > 0$, a single node with $z \leq 0$ leads to multiplication with zero and thus stops the gradient for earlier layers. Another approach to counteract the vanishing gradient problem is to use network architectures based on RNNs which are specifically designed to pass information along for many time steps, such as the *Long Short-Term Memory (LSTM)* network. Building on previous work in the field of RNNs, the LSTM architecture was introduced by Hochreiter and Schmidhuber (1997). It was much more recent, however, that using larger data sets on this model has produced numerous state-of-the-art results in speech recognition (Graves, Mohamed, & Hinton, 2013), handwriting recognition (Graves et al., 2009), the supersedence of humans in complex computer games (OpenAI, 2018) and also time series forecasting (Laptev, Yosinski, Li, & Smyl, 2017).

Essential to the ability of these networks to retain information over extended periods is the so-called cell state $\mathbf{c}$, also called the memory of the model. This vector can hold information for many time steps, or more specifically until it is *actively* forgotten. A full overview of the network architecture at time t is again given in Figure 4.

Looking at the functional principles of the network, there are four layers with individual weights in an LSTM which control the flow of information, indicated by the ellipses in the figure. These layers, also referred to as *gates*, are defined as follows:

The *forget gate* (Equation 28), as implied by the name, is trained to forget information by selectively removing it from the cell state, based on the current inputs $\mathbf{x}_t$ and

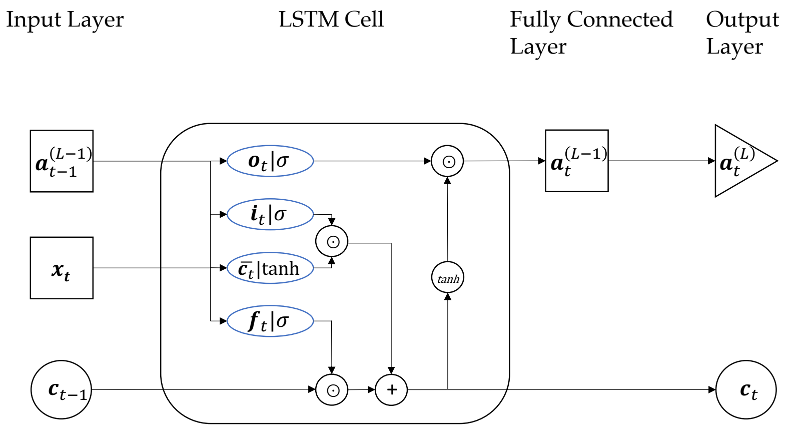

Figure 4: Schematic representation of an LSTM Network

$\mathbf{a}_{t-1}^{(L-1)}$. This is achieved by applying the sigmoid function to the weighted inputs, thus mapping them into the interval $(0,1)$. Since the Hadamard product of the resulting vector and the cell state is computed, values close to one correspond to a retention of the information in the cell state, whereas values close to zero discard most of the information.

$$\mathbf{f}_t = \sigma(\mathbf{x}_t * \Theta_{x,f} + \mathbf{a}_t^{(L-1)} * \Theta_{a,f}) \tag{28}$$

$$\bar{\mathbf{c}}_t = \tanh(\mathbf{x}_t * \Theta_{x,c} + \mathbf{a}_t^{(L-1)} * \Theta_{a,c}) \tag{29}$$

$$\mathbf{i}_t = \sigma(\mathbf{x}_t * \Theta_{x,i} + \mathbf{a}_t^{(L-1)} * \Theta_{a,i}) \tag{30}$$

$$\mathbf{o}_t = \sigma(\mathbf{x}_t * \Theta_{x,o} + \mathbf{a}_t^{(L-1)} * \Theta_{a,o}) \tag{31}$$

Similarly, information can also be added to the cell state dependent on the inputs through a combination of the *candidate gate* (Equation 29) and the *input gate* (Equation 30). The candidate gate first produces possible values to update the cell state with. Since it has a hyperbolic tangent activation function, the results lie in the interval $(-1,1)$ such that both positive and negative states can be achieved. The input gate, on the other hand, has a sigmoid activation function in order to control the extent to which each of the candidate nodes is added to the cell state. The described updating process of the cell state is expressed in Equation 32. With the cell state updated, the output of the network can now be computed. Therefore, first the *output cell* (Equation 31) combines the weighted inputs $\mathbf{x}_t$ and $\mathbf{a}_{t-1}^{(L-1)}$ and applies a sigmoid activation function. These activations are now amended by the information of the cell state. In order to make sure the cell state is in the interval $(-1,1)$, the hyperbolic tangent function is applied to it again. Finally, the Hadamard product of the output gate and the squashed cell state yield the hidden layer output of the LSTM cell, $\mathbf{a}_t^{(L-1)}$, as shown in Equation 33. The multiplicative link of the cell state thus can either weaken or even invert the activation of the output gate. The effect of this step can be conceptualized well with an example from natural language modeling. A negating word at the beginning of a sentence might be held in the cell state with a negative value, which would then invert the prediction otherwise made based on words occurring much later in the sentence.

$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t + \mathbf{i}_t \odot \bar{\mathbf{c}}_t \tag{32}$$

$$\mathbf{a}_t^{(L-1)} = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{33}$$

To obtain the appropriate activation for the task at hand, a final fully connected layer is added before reaching the output layer, which yields $\mathbf{a}_t^{(L)}$. Although the LSTM is is more complex than a simple RNN, it is also trained using the BPTT algorithm by first unrolling the network and propagating the error back from the last to the first observation, finally updating the weight matrices $\Theta_{x,\nu}$ and $\Theta_{a,\nu}\ \forall\ \nu \in \{f,c,i,o\}$.

Again, numerous adaptions to the LSTM model were suggested since its introduction. Gers and Schmidhuber (2000) introduce so-called *peepholes* to the network, which add the incoming cell state $\mathbf{c}_{t-1}$ as inputs to the gates. Another alteration was suggested by Cho et al. (2014), who combine the functionally related candidate and input gates to a single gate referred to as the *update gate*.

2.3. Hypotheses formulation

Following the presented literature, the research objective is to compare the performance of portfolios optimized using expected return estimates stemming from the introduced neural networks to those optimized using a more traditional EWMA estimator. It is well known that in the presented framework the estimation of the expected return is a key determinant of the resulting portfolio performance as the optimal portfolio weights are highly sensitive to these estimates (Best & Grauer, 1991; Black & Litterman, 1992). If neural networks indeed are superior in estimating the expected return, a substantial increase in portfolio performance is to be expected. As higher portfolio returns can also come at the expense of increased volatility, the Sharpe ratio (Sharpe, 1966) of the portfolios is chosen as a performance metric to address this trade-off.

With the presented EWMA estimator referred to as the *baseline estimator*, and the MLP, CNN and LSTM estimators jointly referred to as the *neural network estimators* throughout the thesis, the considered working hypothesis is defined as follows:

Hypothesis 1 (H1): *The neural network estimators are able to extract information from the return history of the assets. Based on their estimates, they can outperform the baseline estimator in terms of Sharpe ratio by reallocating the portfolio according to the optimized portfolio weights at every trading day without considering trading costs.*

However, it is likely that the neural network estimators result in significantly increased trading volumes, as in contrast to the EWMA estimator, vast differences in the return estimations are possible from one observation to the next, potentially leading to a drastic change in optimal portfolio weights. Thus, the trading volumes are measured in the daily rebalancing case, and it shall be analyzed if common measures used to decrease trading volumes such as a reduced rebalancing frequency or intertemporal smoothing of the optimized portfolio weights (Würtz et al., 2009) can mitigate the potential increase. The following secondary hypothesis, therefore, is also considered:

Hypothesis 2 (H2): *In conjunction with common measures to reduce trading volume, the neural network estimators are able to outperform the baseline estimator in terms of Sharpe ratio in the presence of trading costs.*

3. Methodology

Having covered the theoretical background of the thesis and the hypotheses it seeks to examine, the general structure of the analysis carried out is presented in the following, covering the data set itself, the covariance estimator as well as the neural network and optimization implementations.

3.1. Data

The analysis was performed on data obtained from the data library of Kenneth R. French (Fama & French, 2019) which itself uses stock price data from the *Center for Research in Security Prices (CRSP)* and interest rate data provided by *Ibbotson Associates*. The data set contains synthetic daily returns of a market asset, a risk-free asset, a *high minus low* asset, and a *small minus big* asset. The construction of these assets follows Fama and French (1993). The market asset reflects a value-weighted mean of US stocks listed on the NYSE, AMEX, or NASDAQ. The risk-free asset represents the return on the one-month US Treasury bill. A detailed description of the composition methodology is also available at the previously mentioned data library. In order to keep the scope of the analysis on the comparison of expected return estimators, only the market and risk-free asset are considered, reducing the setting to the two-asset case. In line with the naming conventions of the data set, the market asset is henceforth abbreviated as *Mkt*, while the risk-free asset is referred to as *RF*. The date range covered spans from 1926-07-01 to 2019-06-28, thus covering 93 years with a total of 24,515 observations. In terms of pre-processing, the data set was first analyzed for missing or implausible data points. Since none were observed, no data cleaning was required. The return of the market asset r_{Mkt} is not provided directly, but as excess over the risk-free return. Thus, the return of the risk-free asset must be added to reflect the total returns obtained by an investor in the asset. A first look at the total return indices associated with the asset returns reveals that, over the long observation period, both assets performed as one would expect based on the asset classes. As shown in Figure 5 on a logarithmic ordinate, the market asset shows significant long term growth accompanied by substantial risk and even years with consecutive large drawdowns. Notably, periods of extended upward trends are interrupted by financial crises like the Great Depression from 1929-1939, the oil crisis of 1973, the burst of the dot-com bubble in the early 2000s, the global financial crisis of 2007-2008 and the subsequent European debt crisis. The distribution of these events ensures that the neural network estimators are exposed to different market environments in both the training and the testing phase and therefore, potentially capable of finding subtle patterns in the market returns. On the other hand, the risk-free asset by its nature shows a very smooth return profile. However, the yield of the risk-free asset has undergone significant changes during the time span of the data set. Starting at a relatively low yield of 0.9 basis points per day, a period of prevailing increases in the return on the risk-free asset followed, reaching a peak of 6.1 basis points per day in 1981. Afterward, the yield decreased again, incidentally back to the exact initial yield of 0.9 basis points per day, resulting in a sigmoid-like total return graph. While the EWMA estimator eventually follows both upwards and downwards trends, it remains to be seen if the neural networks could improve on the baseline estimator if the models were to be trained on data stemming from a regime with predominantly increases in the risk-free rate but tested on data with predominantly decreases in the risk-free rate. This could be the case if the regime switch aligns with the split

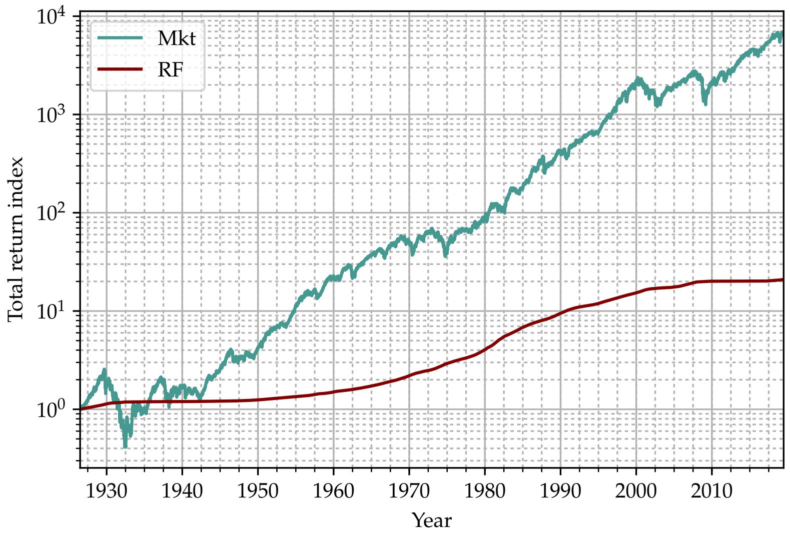

Figure 5: Total return index of assets

of training and testing data.

Next, the asset returns were analyzed for autocorrelation, with the results illustrated in Figures 6 and 7.

As expected, over the full observation period the sample *autocorrelation function (ACF)* is close to zero for the returns of the market assets, with the highest value of 0.067 observed at a lag of one day, as shown in Figure 6. While some of these autocorrelations are significant at the 5% level when compared to the confidence intervals obtained from Bartlett's formula (Bartlett, 1946), they are not stable over time, exhibiting both negative and positive estimates for different subperiods of the return series. This is shown in Figure 7, where the rolling one year (251 day) sample autocorrelation function for a lag of one day on the market asset returns is plotted over time. On the other hand, the risk-free return is by construction almost perfectly positively autocorrelated for all analyzed ten lags. For reference, a correlogram for the risk-free asset is thus provided in the Appendix. While not surprising, this analysis allows for an assessment of the estimators' prediction performance. Since the risk-free asset returns are highly positively autocorrelated, a reasonable prediction is always provided by the previous return. Thus, the differences in the estimates are expected to be small. For the market asset, even though ex-post linear relations to previous returns can easily be identified, finding these linear (or also nonlinear) relationships ex-ante is a different challenge, leading to an expectation of more substantial differences in the estimates.

3.2. Covariance Estimator

While the varied factor in this analysis is the expected return estimator, it is also necessary to estimate the covariance matrix to perform the portfolio optimization. The sample covariance matrix can be a suboptimal estimator when the number of assets approaches the number of observations per asset (Ledoit & Wolf, 2003, 2004). However, in the case at hand, the number of assets is small, and the return history has the same length for all assets. Thus, the sample covariance could be considered a valid estimator. In fact, Lopez and Walter (2002) find that in a Value-at-Risk context, similar to the optimization approach considered in this thesis, simple covariance estimators are not inferior compared to more complex estimators like GARCH models. Especially highlighted by them is the EWMA covariance estimation, which again overweighs more recent observations compared to older ones. Thus, the covariance was estimated using an EWMA model with a smoothing parameter of $\lambda = 0.94$. For daily data, this value was suggested by J.P. Morgan (1996) as well as Walter and Lopez (2000). The resulting annualized standard deviation estimates are shown in Figure 8. The standard deviation of the market asset generally lies between 5% and 20%, with further elevated levels during periods of financial distress and less frequent spikes reaching almost 80%. While the standard deviation of the risk-free asset is not zero, it is minuscule compared to the market asset and thus indistinguishable from zero when viewed on the same scale. The same applies to the covariance of the market asset and the risk-free asset. While the annualized figures are used to provide a more intuitive interpretation, one should note that these values are obtained by scaling the daily volatilities with the factor $\sqrt{251}$. However, this is only accurate when logarithmic returns are used and remains an approximation for the discrete returns used in the portfolio optimization context of this thesis. Therefore, all figures displaying annualized volatilities are replicated using

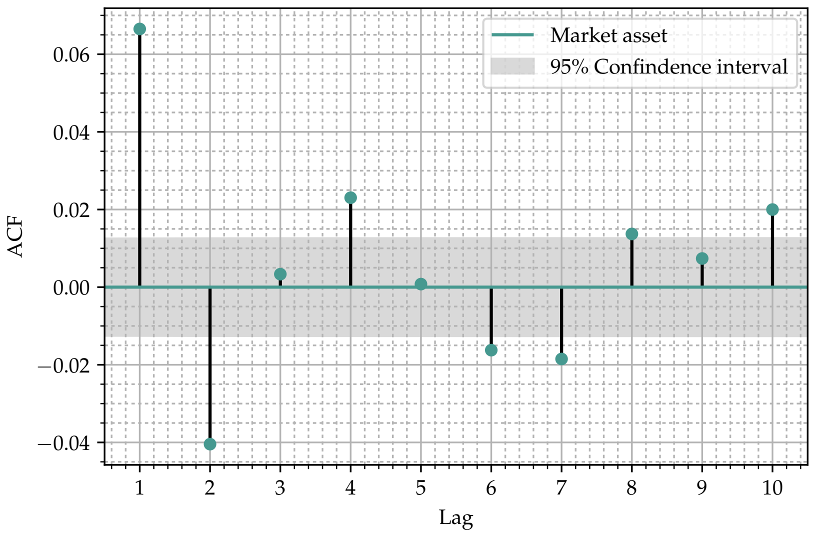

Figure 6: Autocorrelation analysis – Full sample market autocorrelation

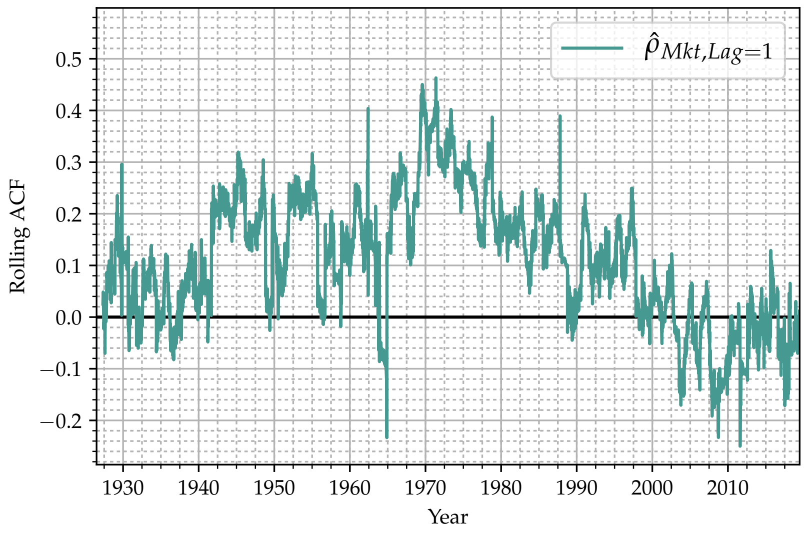

Figure 7: Autocorrelation analysis – Rolling one year market autocorrelation

unscaled daily values in the Appendix.

3.3. Neural Network Implementation

The neural network estimators are implemented in Python using the *TensorFlow* machine learning library (Abadi et al., 2015). More specifically, the high-level application programming interface *Keras* was used within TensorFlow. The following section introduces the three chosen network implementations and describes the considerations behind the choice of the model *hyperparameters*. While model parameters are the weights and biases that are optimized during model training, hyperparameters are any additional configuration factors that themselves influence the training of the neural network, such as the size and number of hidden layers as well as the batch size. Among other approaches, these hyperparameters can be optimized by performing a grid search on a set of reasonable hyperparameter combinations, which can lead to additional improvements in the model performance. It is important to underline, however, that the scope of this thesis is to analyze the general suitability of neural network estimators in the context of dynamic portfolio optimization. Thus, hyperparameter optimization was not performed before the main analysis but is nev-

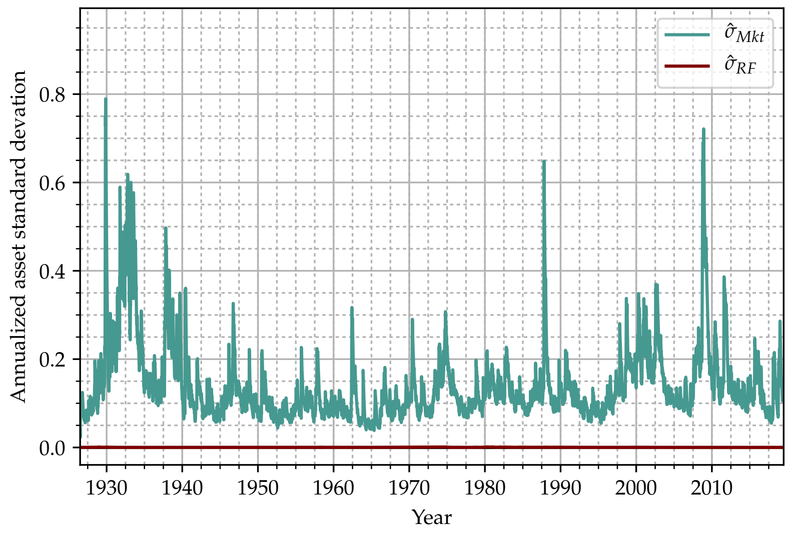

Figure 8: Annualized asset standard deviation over time

ertheless considered in the sensitivity analysis. Rather, sensible hyperparameters were chosen based on related research and best practices.

3.3.1. Multilayer Perceptron

Before implementing an MLP network, the input features have to be selected. As the EWMA estimator only contains information from historical returns, the neural networks also were constrained to this information set. However, while the EWMA estimator encompasses information from all previous observations, a very large number of lagged daily returns would swiftly lead to a dramatic increase in model parameters for the MLP. Thus, a reduced number of features encompassing information about the recent returns had to be chosen. Since the EWMA estimator predicts positive returns if recent returns were also positive and vice versa, a strategy based on this estimator can be considered a *momentum strategy*. Momentum strategies invest in assets which have performed well (compared to other assets or evaluated only on their own return history) over a fixed window of recent observations, usually the last year. These strategies were analyzed by numerous researchers, such as Antonacci (2014); Asness, Frazzini, Israel, and Moskowitz (2014); Baz, Granger, Harvey, Le Roux, and Rattray (2015); Goetzmann and Huang (2018); Hurst, Ooi, and Pedersen (2017); Jegadeesh and Titman (1993); Lempérière, Deremble, Seager, Potters, and Bouchaud (2014); Swinkels (2004), with most finding at least partial support for the strategy. Closely related to the topic of this thesis, Lim et al. (2019) even combined a momentum strategy on futures contracts with neural network estimators and concluded that the neural network estimators outperformed the traditional signals. In order to reflect the informational content of these momentum strategies, the trailing one year return (251 days), one month return (21 days), one week return (5 days), as well as the most recent daily return, are used as features. Additionally, the rolling one month sample standard deviation is added to provide the model with information about the current level of return uncertainty.

Next, the number of hidden layers and their node count had to be decided upon. Given the relatively small size of the data set compared to other machine learning tasks, the model was also kept compact. Goodfellow et al. (2016) find that in different settings, deeper networks provided better generalization. Additionally, Di Persio and Honchar (2016) find that for a similar application in prediction of financial time series, two hidden layers were optimal. Thus, two hidden layers were also used in this model, as shown in the schematic representation in Figure 9.

The number of nodes in each hidden layer was chosen next. A typical default parameterization chooses a number of hidden nodes between one and two times the number of input nodes (Berry & Linoff, 1997; Blum, 1992). Conventionally, the number of nodes are often chosen as powers of two, as this allows for broad coverage of the search space when multiple node counts are compared. Sticking with the convention, with five input nodes, the number of nodes in the hidden layers was set to eight.

For the activation function in the hidden layers, the ReLU function was used. This activation function has numerous theoretical and empirically founded advantages. One example is that its activations are linear and thereby unsaturated for positive inputs, making models using this activation function easier to train (Goodfellow et al., 2016). Additionally, this property also mitigates the van-

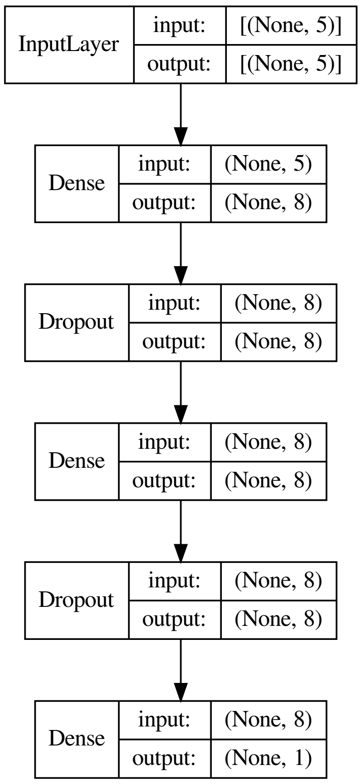

Figure 9: MLP implementation

ishing gradient problem as described by Glorot, Bordes, and Bengio (2011) who also found that ReLU activations more closely reflect the principle of operation of neurons in the brain compared to other popular activation functions. While these considerations led to the selection of the ReLU activation function for the hidden layer, a linear activation was necessary for the output layer, since the prediction of returns is a regression problem.

The number of trainable model parameters in a neural network can often be large compared to other statistical models. Intuitively, it makes sense that neural networks with many parameters can more easily overfit smaller data sets. This was also observed by Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov (2014), who proposed the implementation of *dropout* layers into the model. A dropout layer randomly deactivates some of its nodes (along with their corresponding weights) with a pre-specified probability during the forward pass and the backpropagation steps of the gradient descent algorithm. The underlying idea is that whenever a set of nodes is deactivated, the other nodes each learn an internal representation of the model, thus reducing a co-adaption of neighboring nodes. This technique was also described by Hinton, Srivastava, Krizhevsky, Sutskever, and Salakhutdinov (2012), suggesting a dropout probability of 0.5. It is therefore often considered best practice to add a dropout layer when implementing a neural network. The chosen model architecture thus introduces dropout layers, as shown in the network architecture schematics in Figure 9. Note that in this representation, the dropout layer does not have its own nodes, but rather sets each node of the preceding dense layer to 0 with probability 0.5. Additionally, it should be noted that the fully connected layers of the MLP model are referred to as *Dense* layers in the Keras framework. Thus, for each sample, there are five input values which are passed through two hidden layers with eight nodes each. Finally, the output layer has a single node since only the return on the next day is estimated. The *None* values refer to the batch size, which is not set yet during the model construction and is further addressed below.

3.3.2. Convolutional Neural Network

For the CNN implementation, schematically depicted in Figure 2, a new set of input features had to be decided upon.

Since the extraction of higher dimensional features is at the core of CNNs, rather than one month, one week and one day return, all 21 lagged daily returns of the preceding month of a given observation were used as features.

Following this argument, the rolling one month standard deviation was also removed from the inputs since this information too could be learned by the model. The one month period was chosen in particular since the half-life and center of mass of the baseline estimator also lie in this region with 17 and 24 days, respectively. However, the rolling one year return was kept as an input in order to retain the longer term information content without dramatically increasing the model parameters through the inclusion of the full year of lagged daily returns. Thus, the total number of input features for each observation is 22. For the convolutional layer, or *Conv1D* as termed in the Keras framework, following Di Persio and Honchar (2016), the size of the filter was chosen to be $[3 \times 1]$ with a stride of $s = 1$. While the number of filters in their analysis was 32 and 64 respectively, considering the size of the data set and the comparability of the number of parameters across the different models of this thesis, only 16 features were used. It should be noted that the Conv1D layer in the network outputs one vector of nodes for each filter, as indicated by the output of an additional dimension by this layer. Furthermore, since no padding is used, the size of the convolutional layer decreases from $[22 \times 1]$ to $[20 \times 1]$ per filter, as per Equation 23. Considering all filters, the total shape of the convolutional layer thus is $[20 \times 16]$. A max pooling layer follows, which uses an input size of $[2 \times 1]$ and stride $s = 2$, thus reducing the number of nodes by 50%. This parameterization again follows Di Persio and Honchar (2016). After the technically required flatten layer, the abstracted features are combined through a fully connected layer with 16 nodes and ReLU activation function. Before the final output layer, a dropout layer is added to counteract overfitting.

3.3.3. Long Short-Term Memory Network

For the LSTM architecture shown in Figure 4, the input layer again had to be created differently from the previous models since it allows the network to create its own internal representation of the importance of previous returns.

At each point in time, the individual daily returns of the previous month (21 days) are given to the model as inputs, again encompassing the half-life and center of mass of the baseline estimator. This window is shifted one day ahead at each new training sample. Since these inputs are not 21 different features but rather 21 lagged observations of a single feature, again a second dimension in the input layer is required. Hence, the tuple $(21, 1)$ is used to specify the input shape.

The 16 nodes in the LSTM layer were again chosen to be approximately equal to the number of input nodes. In this model, the number of nodes describes the length of the vectors used in its gates and the cell state. To allow for further long term dependencies, it is crucial not to shuffle the data set before training, which otherwise is the default implementation in Keras. While other model implementations treat each training sample independently with the input and output values they provide, the very definition of RNNs requires the sequences to be ordered for both training and testing. When it comes to regularization techniques in an RNN network such as the LSTM model, it is possible to not only conceptualize dropouts on the direct path from the input to the output layer, but also on the gates which control the influence of previous hidden layers. Indeed, this was noted by Gal and Ghahramani (2015), who proposed theoretically founded dropouts for RNNs and provided empirical evidence for their usefulness. Thus, in the LSTM model, recurrent dropouts are used in addition to the previously introduced dropouts. Only the regular dropout layer is shown in the schematic, as the recurrent dropouts are included in the LSTM layer in Keras.

Even though the architecture of the LSTM model might look simpler compared to the MLP and CNN models, it is in no way inferior in its capability to represent complex dependencies through its internal gate structure and the transfer of information between samples.

3.3.4. Model training

In order to train the models, the data set first needs to be split into train and test data sets. It is common practice to use the older part of the data as training data and the more recent data as test data in order to improve the likelihood that the performance of the model in an out-of-sample implementation would be closely reflected by the test data performance. The ratio by which to divide the data set is dependent on the application and especially the size of the data set. While on vast data sets the training data can encompass up to 99 percent of the total samples, smaller data sets can require a relatively larger testing set, such that equal splits of the data are also not uncommon (Ng, 2019b). In the data set at hand, a split of 60 percent for the training data and 40 percent for the test data was chosen. Theoretically, this would leave 14,709 observations for the training data set and 9,806 observations for the test data set. However, as shown in Figure 12, not all of this data can be used for training and testing. At the beginning of the training phase, the data required to obtain the first set of input features cannot be used as training examples. If for example, the trailing one year return is one of the input features, this data is not available for data points up to one year into the data set. At the very end of the data set, one also needs to account for the output values. If the prediction target is the return on the next day, then a single observation at the end needs to be omitted from the test data set, as otherwise, no observed return for the prediction on the very last day is available. If returns for longer time horizons are predicted, more observations need to be spared for validating the predictions. Furthermore, it is especially important to separate the training and testing data by a gap with the combined length of the data points required for the input features and the number of days predicted

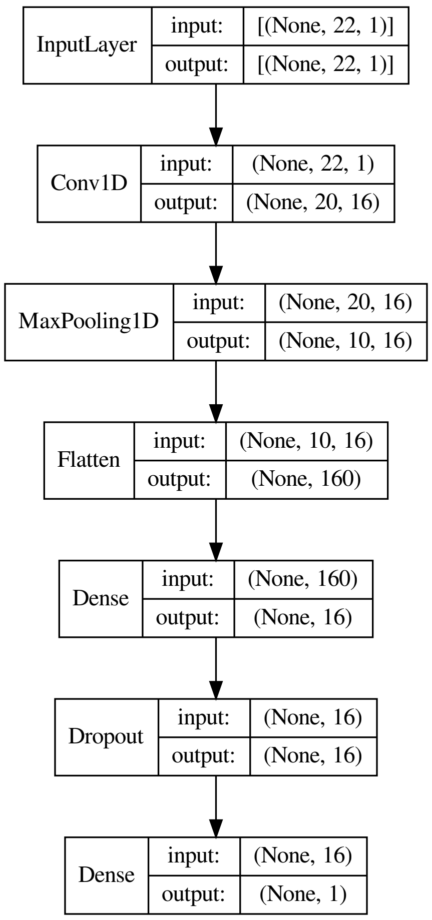

Figure 10: CNN implementation

ahead. The importance of this step stems from the phenomenon that neural networks are outstanding in finding subtle information about the test data within the training data. This is called *information leakage*. In particular, it needs to be assured that the test data does not contain any information the network could have seen during the training phase. Continuing the example of trailing one year returns used as a feature, if no gap between the train and test data sets were present, the returns of the final year of training observations would be part of both the train and test data set and thus contribute to information leakage. A further overview of information leakage is given by Kaufman, Rosset, and Perlich (2011). Removing the observations in question leaves 9,552 samples or approximately 38 years for backtesting the neural networks against the baseline estimator.

While the EWMA estimator is agnostic to the size and variance of the inputs, neural networks can be affected significantly by inputs of varying scales. In particular, if inputs are on different scales, the loss function of the neural network is dominated by the larger features (Theodoridis & Koutroumbas, 2009). Conceptually, for unscaled features, the minima of the error surface become more elliptical, with the gradient descent algorithm potentially oscillating towards the optimum in an inefficient manner. The increased convergence efficiency on scaled features was also noted by Ioffe and Szegedy (2015). Therefore, it is beneficial to scale the input and output data for the training and testing data set to a comparable range. Since the resulting estimates are used in a mean-variance optimization setting, the commonly used method of *z-score normalization* was used to ensure the input features do match in these dimensions. Formally, for each feature x_i with given sample mean $\hat{\mu}$ and sample standard deviation $\hat{\sigma}$, the normalized feature x_i' was computed as

$$x_i' = \frac{x_i - \hat{\mu}}{\hat{\sigma}}. \tag{34}$$

However, as suggested by Müller and Guido (2016), scaling the data can also be another source of informa-

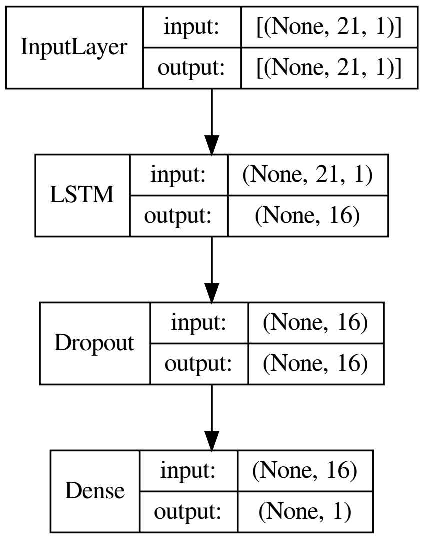

Figure 11: LSTM implementation

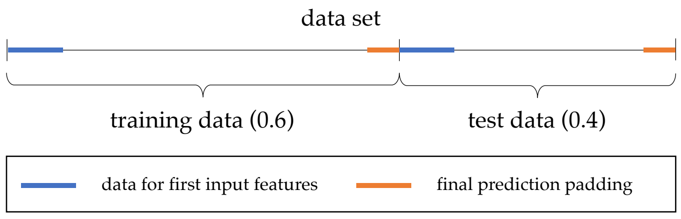

Figure 12: Schematic representation of the train/test split

tion leakage. If the data set is scaled before being split, implicitly the training data contains the mean and standard deviation of the full sample even though this information would not have been available at that time. Thus, the training data needs to be scaled first based only on the estimates obtained on these samples. Next, the test data can be scaled by using the same mean and standard deviation estimates. The same scale is then also used when re-scaling the estimates after training and prediction have been performed.

Finally, before training the model, a batch size had to be chosen. As discussed in section 2.2.1, this parameter can influence the convergence speed and probability of the model considerably. A batch size of 32 is recommended by Bengio (2012) as well as Masters and Luschi (2018). Their research shows that batches in this size region provide a good trade-off between training stability and model generalization across different applications. Thus, this value was chosen as a starting point.

Under consideration of these implementation details, the models could now be trained. The return of the following trading day was used as the output, and the neural networks were trained to minimize the mean squared error with respect to these targets. Computation time was decreased significantly by training the models on a graphics processing unit (GPU) rather than a central processing unit (CPU), with the particular implementation making use of the optimized linear algebra subroutines provided by the Nvidia CUDA framework (Nickolls, Buck, Garland, & Skadron, 2008). Each model was trained for 30 epochs, i.e., the training set was looped over 30 times by the gradient descent algorithm. It was monitored that the loss was decreasing during the training phase (indicating convergence on the training data) and that the loss on the test data was not significantly larger than the loss on the training data (indicating overfitting). It should also be noted that for training the neural networks, a slightly improved version of gradient descent was used, namely the Adaptive Moment Estimation algorithm (*Adam*). This optimizer was suggested by Kingma and Ba (2014) and has since found wide adoption in the field. Compared to the single, fixed learning rate parameter α in the introduced GD algorithm, Adam uses varying learning rates for each parameter which are updated based on exponential moving averages of the first and second moment of recent gradients. It is the default optimization algorithm recommended by Ruder (2016).

3.4. Optimization Implementation

When implementing the portfolio optimization, either the approach of maximizing the portfolio return for a given level of risk or the minimization of risk for a given level of return had to be chosen. As seen in Figure 8, the risk of the assets was comparatively stable over time. By contrast, market returns are much harder to predict, and therefore the optimized portfolio could deviate significantly from the targeted return. Additionally, assuming a constant relative risk aversion was found to be a good representation of some investors' utility functions (Chiappori & Paiella, 2011; Wakker, 2008) and thus a constant level of targeted risk for the portfolio is reasonable. With that in mind, the optimization with respect to a prespecified level of risk was selected. In order to reflect different degrees of risk aversion, multiple risk targets were considered. The lowest volatility target was chosen at 2%, well below the lowest volatility estimate for the market asset at over 5% and thus always requiring a portfolio composed of both assets whenever the expected return of the market asset is above the expected return of the risk-free asset. Additional risk targets were placed in increments of 2.5 percentage points, covering a range up to 19.5% annualized volatility, the upper end of the market volatility during most of the observation period.

To obtain the optimized weights necessary for the analysis, the portfolio optimization had to be performed for each date where all estimators delivered estimates (the EWMA estimator does not need training data, but since this split was required for the neural networks, the intersection of the estimation periods had to be used). This included 9,552 dates for which the optimization needed to be performed. For the EWMA estimator, the same smoothing parameter of $\lambda = 0.94$ as for the covariance estimator was chosen. Additionally, alongside the baseline and neural network estimators a *RANDOM* "estimator" was added as a point of reference. This model uses the sample mean and standard deviation of the assets during the training period to parameterize a normal distribution from which a sample was drawn for each testing observation. With the five estimators and the eight considered levels of risk tolerance, this equates to a total of 382,080 optimizations. Conceptually, the optimization seeks to find weights $\mathbf{w}_{target}$, which maximize the expected return for a given level of risk $\bar{\sigma}$, as shown in the optimization problem 2. However, solving this linear program with quadratic constraints was not easily implementable in the chosen programming framework. Thus, the efficient portfolio for the given risk level was obtained by repeatedly minimizing the risk for a given expected return (optimization problem 1a). In the chosen implementation, first, the portfolio weights with minimal standard deviation $\mathbf{w}_{\sigma_{min}}$ were computed. If the risk target was below the standard deviation of this portfolio, σ_{min}, the optimal portfolio weights were set to $\mathbf{w}_{\sigma_{min}}$, which have an expected return of μ_{min}, respectively. Similarly, the point on the efficient frontier with the highest standard deviation, an investment solely in the asset with the highest expected return μ_{max} was obtained. For risk targets larger than the standard deviation of this upper end of the efficient frontier, σ_{max}, again the weights of this portfolio $\mathbf{w}_{\sigma_{max}}$ were used. It should be noted that investing solely in the asset with the highest expected return marks the upper bound of the efficient frontier only when the no short sale constraint mentioned in 1d is added. This was the case throughout the analysis, as otherwise extreme leverage of the portfolio could not be prevented. Since this is held constant between estimators, it is not expected to influence the overall result.

For risk targets between these upper and lower bounds, first define $\Delta_{\mathbf{w}}(\sigma(\mu)) := \sigma(\mu) - \bar{\sigma}$. The original optimization problem of maximizing the expected return for a given level of risk is now equivalent to finding the μ for which the weights with the minimal standard deviation set $\Delta_{\mathbf{w}}(\sigma(\mu)) = 0$. To find this root, *Brent's method* (Brent, 1973) was used, which stands out due to its high reliability and rate of convergence. This is achieved by combining fast converging methods like the *secant method* with the more reliable *bisection method*. As a starting point, either σ_{min} or σ_{max} was used, depending on which had the lower absolute distance to the risk target. Then, for every iterative change in μ, first the optimization problem 1a was solved, yielding the corresponding efficient level or standard deviation $\sigma(\mu)$. The resulting change in $\Delta_{\mathbf{w}}(\sigma(\mu))$ was then used for the next iteration in Brent's method. Dybvig (1984) shows that the efficient frontier in the presence of a no short sale constraint is continuous but not always differentiable at every point, disproving a conjecture of Ross (1977). However, only continuity is required for Brent's method to always converge to $\mathbf{w}_{target}$ (Brent, 1973). Indeed, the convergence was asserted during all optimizations and could always be achieved using the employed implementation.

A disadvantage of this approach, however, is the multiplicatively increased number of optimizations that had to be performed since every initial return maximizing optimization problem was substituted by multiple risk minimizing optimizations. An overview of the full optimization approach is given in Algorithm 2. By using this algorithm, the optimized weights for all data points, estimators, and risk levels were obtained.

4. Results

In the following chapter, the results of the performed analysis are presented and discussed. First, the case of daily portfolio adjustments (also referred to as *rebalancing*) without the consideration of trading costs is ana-

```
Compute minimum variance portfolio with $\sigma_{min}$,
  $\mu_{min}$ and $\mathbf{w}_{\sigma_{min}}$
Compute maximum return portfolio with $\sigma_{max}$,
  $\mu_{max}$ and $\mathbf{w}_{\sigma_{max}}$
if $\bar{\sigma} \leq \sigma_{min}$ then
  $\mathbf{w}_{target} = \mathbf{w}_{\sigma_{min}}$
else if $\bar{\sigma} \geq \sigma_{max}$ then
  $\mathbf{w}_{target} = \mathbf{w}_{\sigma_{max}}$
else
  $\mu = \underset{\mu \in \{\mu_{min}, \mu_{max}\}}{\arg\min} |\sigma(\mu) - \bar{\sigma}|$
  $\Delta_{\mathbf{w}} = \sigma(\mu) - \bar{\sigma}$
  while $|\Delta_{\mathbf{w}}| > \epsilon_{tol}$ do
    Adjust $\mu$ according to Brent's method
    Minimize portfolio volatility for $\mu$, yielding
      $\sigma_{\mu}$ and $\mathbf{w}_{\sigma_{\mu}}$
    $\Delta_{\mathbf{w}} = \sigma(\mu) - \bar{\sigma}$
  end
  $\mathbf{w}_{target} = \mathbf{w}_{\sigma_{\mu}}$
end
```

Algorithm 2: Risk targeting optimization

lyzed. Afterward, two approaches to reduce trading volumes are considered: a reduction in trading frequency and the intertemporal smoothing of optimized portfolio weights. In this setting, the portfolio performances are evluated in the presence of trading costs. Finally, a sensitivity analysis is performed to evaluate the robustness of the results.

4.1. Daily Rebalancing

With the daily rebalancing setting chosen as a starting point, the neural networks were trained, and the optimization algorithm was employed for each observation in the test data set. The results of this analysis are presented in the following.

4.1.1. Predictive Accuracy

Before the performances of the dynamically optimized portfolios are analyzed, the underlying return predictions are assessed. Relating to the error function of the networks, a natural first criterion is the mean squared error of the estimates. An overview of this metric for all estimators and both assets is provided in Table 1. Additionally, the related *mean absolute error (MAE)* is stated to add a measure which is less sensitive to outliers.

Unsurprisingly, all estimators perform substantially better on the risk-free asset compared to the market asset. On the risk-free asset, the CNN network has both the lowest MSE and MAE, with the EWMA estimator as a runner-up in both measures. The LSTM and MLP follow, with the largest error made by the RANDOM estimator. For the market asset, the lowest squared and absolute errors are made by the MLP model, with the LSTM, CNN, and EWMA asset closely behind. The relative difference of the baseline and neural network estimators to the RANDOM model is considerably smaller for the market asset, illustrating the large uncertainty in the returns of this asset. To compare the predictive accuracy of two estimators, Diebold and Mariano (1995) propose a test for non-nested models which can be used across a wide variety of error functions, including MSE and MAE. Diebold (2015) suggests that the underlying assumption of the error differential being covariance stationary often provides a reasonable approximation for economic time series. Moreover, Harvey, Leybourne, and Newbold (1997) propose a bias correction for the test statistic and the Student-t distribution to compare this corrected statistic against in order to improve the properties of the test for smaller samples. Table 2 presents the bias-corrected, two-sided test statistics of predictive accuracy comparisons between the neural networks against the baseline estimator, with the null hypothesis stated as a zero error differential. Again, the RANDOM estimator is added as a reference point. The sign of these test statistics indicates whether the baseline estimator had the lower (negative sign) or higher (positive sign) error, corresponding to the errors stated in Table 1.

Although on the risk-free asset the absolute losses were small, all tests show a highly significant loss differential against the baseline, both for the more accurate CNN model as well as the other, less accurate models. On the market asset, the test statistics generally turn out to be smaller. While the lower error of the MLP is still significant against the baseline for both error measures, the error differentials of the CNN and LSTM model do not show significance even at the 10% level on the MSE metric. For the MAE, the inferiority of the LSTM against the baseline estimator also is not significant, but the more precise estimation of the CNN model is. Again, as expected, the RANDOM model is outperformed on both assets with the highest significance. No model has therefore significantly outperformed the EWMA estimator on both assets and error metrics, with the CNN model coming closest and only one error differential being insignificant. Since the magnitude of the error on the risk-free asset is much smaller, it is likely that the accuracy on the market asset dominates the performance of the dynamically optimized portfolios as statistic significance not always implies likewise economic significance. Additionally, while the considered error measures can give a first assessment of the predictions, their accuracy might not be linearly reflected in the portfolio performances. While a lower prediction error indicates a better fit and performance *ceteris paribus*, two different estimators cannot solely be compared on these measures. This can be demonstrated with a simple example. When considering a day with a positive return in an asset and an even higher estimated return for this asset, the portfolio would still benefit from the positive realized return, yet perhaps in an ex-post suboptimal portfolio composition. If, however, the return was underestimated by the same dis-

Table 1: MSE and MAE of daily return predictions

	MSE		MAE	
	RF	Mkt	RF	Mkt
EWMA	*3.09e-10*	1.18e-04	*1.06e-05*	7.24e-03
MLP	2.69e-09	**1.15e-04**	3.65e-05	**7.17e-03**
CNN	**7.52e-11**	1.17e-04	**6.41e-06**	*7.19e-03*
LSTM	5.66e-10	*1.16e-04*	1.88e-05	7.26e-03
RANDOM	2.80e-08	2.28e-04	1.33e-04	1.15e-02

Bold and italic table entries represent the lowest and second lowest error per column, respectively.

Table 2: Diebold-Mariano test for predictive accuracy against baseline estimator

	MSE		MAE	
	RF	Mkt	RF	Mkt
MLP	*−35.182****	*3.835****	*−71.605****	*5.687****
CNN	21.667***	0.764	30.121***	3.049***
LSTM	−19.023***	1.550	−36.320***	−1.125
RANDOM	**−64.953*****	**−36.036*****	**−120.705*****	**−50.786*****

Bold and italic table entries represent the highest and second highest absolute test statistics per risk target, respectively. Negative test statistics correspond to lower errors by the baseline estimator, positive test statistics correspond to higher errors by the baseline estimator.
$^{***}p < 0.01$, $^{**}p < 0.05$, $^{*}p < 0.1$ for a two-sided test.

tance, perhaps even leading to a negative prediction, the optimization might not have chosen this asset, and the portfolio would forgo the positive return. However, the portfolio performance is not only asynchronously dependent on the prediction error, but the portfolio weights resulting from an optimization are also discontinuous in the estimated returns, as for example two assets with the same estimated volatility are swapped when the asset estimated to have the higher expected return changes.

4.1.2. Risk-Return Characteristics

The portfolio returns based on the predictions are assessed next. Since the portfolios are rebalanced daily, it is assumed that at the end of each day t, the portfolios are adjusted towards the optimized weights based on the predictions obtained from the returns preceding and including day t. Denoting these weights as $\mathbf{w}_t$, the portfolio return on the next day $r_{pf,t+1}$ is obtained by summing the returns of each asset after multiplying them by their respective weights within the portfolio. In matrix notation, the portfolio return is obtained as

$$r_{pf,t+1} = \mathbf{r}^{\mathbf{T}}_{assets,t+1}\mathbf{w}_t \,. \tag{35}$$

Iteratively applying this computation to each day in the test period across both risk targets and estimators yields a total of 40 portfolio return time series. A good overview of the performance characteristics of these time series can be obtained by plotting them in the same risk-return space they were optimized in. This is shown in Figure 13[5], where all portfolio performances, as well as the two underlying assets, are shown in the μ-σ-space. The first thing to note is that even the strategies with the highest risk targets are well below their risk budget, with all but the LSTM model reaching a maximum portfolio volatility of around 10%. Since even the market asset only reached around 17% annualized volatility, it is clear that no portfolio could have achieved the upper limit of the risk targets. Additionally, whenever the expected return of the market falls below the risk-free asset, the portfolio is wholly allocated in the latter asset. These periods further decrease the upper risk limit of the portfolios. Since the underlying assets are essentially uncorrelated, it is natural to observe a risk-return profile above the line connecting the realizations of the market and the risk-free asset. As a notable exception, the RANDOM estimator does not even experience this diversification benefit. However, the risk-return profiles of this estimator by construction showed significant inconsistency between runs, sometimes also exhibiting slightly better or worse realizations. While no model quite achieved the return of the market, the CNN model came closest whilst still provid-

[5] All figures in this chapter showing approximated annualized volatilities are again presented in the Appendix using daily figures.

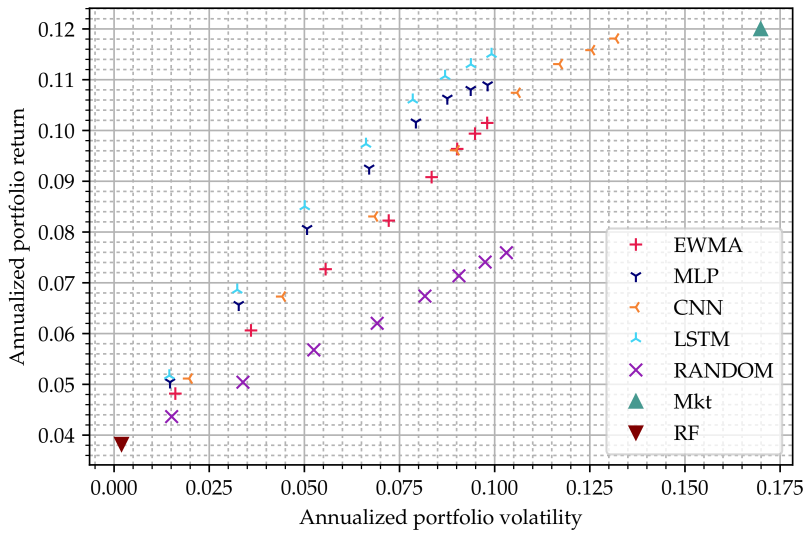

Figure 13: Portfolio performances in μ-σ-space (daily rebalancing, no TC)

ing a lower level of volatility. The LSTM and MLP model further reduce the volatility at their highest risk target, but at an again slightly reduced return. Overall, the MLP and LSTM models achieve higher portfolio returns compared to the EWMA model while staying close to it in the risk space. The realizations using the CNN estimator instead extend the "realized frontier" of the EWMA estimator by providing higher returns in exchange for higher volatility. While the EWMA realizations are inefficient compared to the MLP and LSTM model, the efficiency criterion does not allow for a comparison between the EWMA and the CNN models.

Nevertheless, a closer look at the total return indices of the individual time series is required to assess properties not reflected by their first two moments. The time series for the medium risk target of 12% are shown in Figure 14. Most notably, the MLP and LSTM models achieve substantial outperformance in the beginning, approximately until the turn of the millennium. At that point, the market asset, which is closely tracked by the CNN model, reduces the gap to the MLP and LSTM model, eventually overtaking them for the given risk level. One possible explanation for the decreasing outperformance of the neural networks could be given by the emergence of increased research focusing on the application of these models in the financial industry around that time (see for example Bergerson and Wunsch (1991); Chen, Leung, and Daouk (2003); Deboeck (1994); Gately (1995); Tino, Schittenkopf, and Dorffner (2001)). Practitioners following this research could potentially have lead to the incorporation of this information into the market, reducing the edge these models may have had before. The EWMA estimator also tracks the market asset, however, more loosely than the CNN model. Compared to the market asset, the EWMA model partly avoids the largest drawdowns at the cost of fast recoveries, which is especially apparent at the global financial crisis of 2008, where the market asset dropped sharply by over 50%, followed by an extended period of predominantly positive returns. While the EWMA model fell significantly less during the crisis, it also did not fully experience the following upward trend. In this figure, the improvement of all models over the RANDOM model is also clearly visualized. Looking at the total return indices for the highest and lowest volatility targets generally provided similar insights, and these figures are therefore presented in the Appendix.

The performance of the models during financial downturns is further emphasized in Figure 15, where the risk axis represents the *maximum drawdown (Max DD)* of a portfolio, i.e. the largest loss sustained from a previous high-water mark, rather than its volatility. Again, all models reduce the risk substantially compared to the market asset. What stands out, however, is that the EWMA estimator decreases the maximum drawdown the most, topping at around 26% for the highest volatility target, thereby reducing the risk in this metric by over ten percentage points compared to the neural networks. By construction, the EWMA estimator projects a negative outlook after repeated negative returns, which results in withdrawal from the risky market asset and a shift towards the risk-free asset, thus reducing losses if the market continues to decline. This perhaps is one of the main reasons why the EWMA estimator is often chosen for this application. Conversely, the higher maximum drawdown for the neural networks might be particularly harmful for these estimators. The high degree of complexity of these

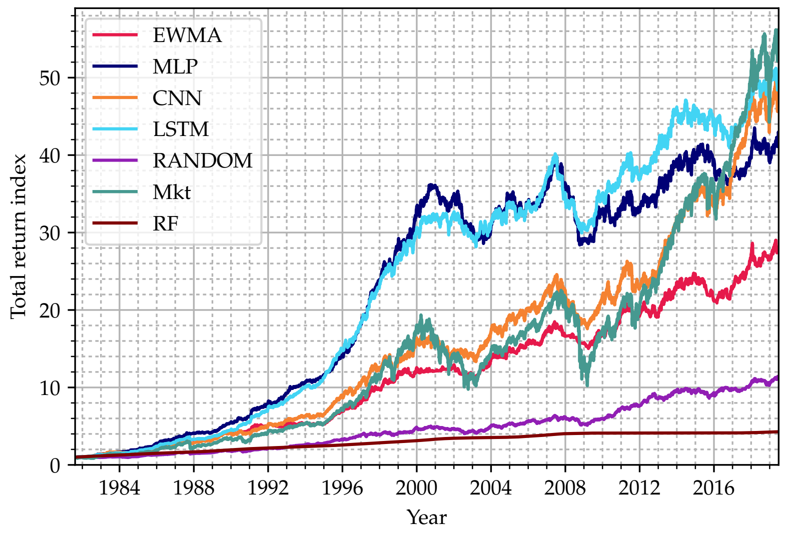

Figure 14: Total return indices for the medium risk target (12% volatility)

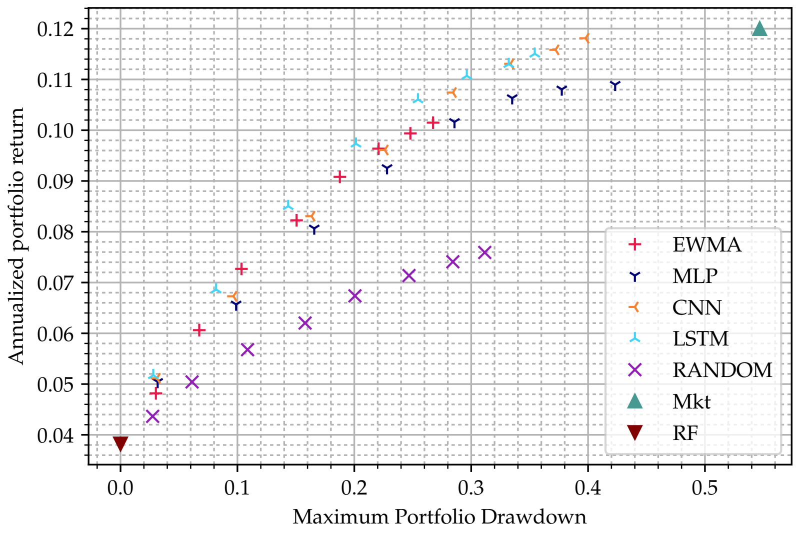

Figure 15: Portfolio performances in μ-Max DD-space
(daily rebalancing, no TC)

models compared to more traditional estimators is likely to have an adverse effect on less sophisticated investors from a behavioral point of view, especially since the estimates of these models are not easily traceable. This applies not only to periods marked by large drawdowns but also to periods of relative underperformance against a benchmark.

4.1.3. Portfolio Sharpe Ratios

As the hypotheses are formulated on this basis of the portfolio Sharpe ratios, this metric is considered next. For a more intuitive interpretation, the annualized rather than the daily Sharpe ratios are considered throughout the analysis, with the non-approximated, daily ratios being presented in the Appendix.

Formally, the ex-post Sharpe ratio is defined as $SR = \frac{\bar{r}_e}{\sigma_{r_e}}$, where $\bar{r}_e$ represents the mean return in excess of the risk-free asset and σ_{r_e} represents the volatility of these returns (Sharpe, 1994). Table 3 presents an overview of all 40 Sharpe ratios obtained by the portfolios. As was already apparent from the risk-return figure, the LSTM achieves the highest Sharpe ratio of up to 0.942 on the lower risk targets. Following the hyperbolic shape of the efficient frontier, the Sharpe ratios decrease for the neural network estimators with increasing risk targets. The Sharpe ratio of the EWMA estimator does not experience this decline and even shows a slight upward trend towards the higher risk targets. Compared to the markets' Sharpe ratio of 0.481 over the same period, all but the RANDOM estimator showed a substantial increase in this metric. While the MLP and LSTM networks achieve higher Sharpe ratios compared to the EWMA estimator across all risk levels, the CNN only leads in the lower risk range.

To assess the significance of these results, a test of Sharpe ratio differentials suggested by Ledoit and Wolf (2008) was conducted. This test is suited for the commonly observed heavy tails of financial time series. It uses a heteroskedasticity and autocorrelation (HAC) robust kernel estimation to compute consistent standard errors on the Sharpe ratio differential. The two-sided test statistics for all estimators against the baseline is given in Table 4, with the null hypothesis stating the equality of Sharpe ratios.

For the MLP and the CNN models, none of the Sharpe ratio differentials are found to be significant. The LSTM model shows significance at the 5% level for the risk targets up to 7% volatility and significance at the 10% level for the risk target of 9.5% volatility, with no significance for higher volatilities. Remarkably, the inferiority of the RANDOM estimator also only shows significance at the 10% level. One should note that even though the RANDOM model has larger test statistics, the LSTM model shows stronger significance for the lowest three risk levels. This is due to the fact that the stated test statistic represents the HAC standard error by which the Sharpe ratio differential is normalized to determine the statistical significance. More specifically, the p-value of the two-sided test is given by $p = 2 * \Phi(-\left|\frac{\widehat{\Delta SR}}{\sigma_{\widehat{\Delta SR}}}\right|)$, where $\Phi(.)$ is the cumulative standard normal distribution function, $\widehat{\Delta SR}$ is the estimated non-annualized Sharpe ratio differential and $\sigma_{\widehat{\Delta SR}}$ is the HAC standard error of this differential. As the LSTM model has the largest Sharpe ratio differential in this lower risk region, it achieves higher significance even though the HAC standard error is slightly smaller.

In light of these results, only a partial support for Hypothesis 1 can be established. While the neural network estimators do show higher Sharpe ratios across most risk targets, few of these performance differentials withstand a test for statistical significance.

4.1.4. Trade Volume

While no trading costs were considered in this case, it is still worthwhile to asses the trade volume associated with these strategies in order to evaluate how drastically the trading volume differs between them and potentially by how much it would need to be reduced to make them viable. Since the portfolios are rebalanced every day, the trade volume is given by the sum of absolute weight differences in the weight vector before and after the rebalancing. At the beginning of day t, the weights $\mathbf{w}_{t,bod}$ are given by the optimized weights of day $t-1$, to which the portfolio was traded. During day t, the weights change through the different returns on the assets, thus at the end of the day, their new weight is determined by their growth, normalized to one by a division through the sum of adjusted portfolio weights for all assets. Formally, for n assets, the end of day weights are given by

$$\mathbf{w}_{t,eod} = \frac{\mathbf{w}_{t,bod} \odot (\mathbf{1} + \mathbf{r}_t)}{\sum_{\nu=1}^{n} w_{\nu t,bod}(1 + r_{\nu t})}, \tag{36}$$

and trade volume at day t is thus being defined as the sum of absolute differences between the end of day weights and the optimal portfolio weights for that day, or formally

$$\text{tv}_t = \sum_{\nu=1}^{n} \left| w_{\nu t,eod} - w_{\nu t,optimized} \right| . \tag{37}$$

One should note that the trade volume defined by Equation 37 is defined in the portfolio weight space, thus making it comparable across time when the portfolio value changes. A switch from a portfolio completely allocated in the risk-free asset to a portfolio completely comprised of the market asset would therefore correspond to a trade volume of 2.0. Table 5 shows the average yearly trade volumes for all estimators and risk targets. Since the EWMA estimator by construction only adjusts its estimates in a smoothed manner, it shows the lowest trade volume across all estimators. Yet still, it too would not be a viable strategy even at low trade costs. While the CNN network shows approximately a doubling, the MLP and especially the LSTM network show a manifold increase in trade volume, with the latter even resulting in higher trade volumes than the RANDOM model. The slight edge in performance of the MLP and LSTM models thus comes in conjunction with an extreme trading volumes which would make them unprofitable even for very low trading costs. These volumes are also indicative of very unstable predictions with little clustering. Thus, in the following, two ways to reduce the trade volume in the presence of these unstable estimates shall be discussed.

4.2. Monthly Rebalancing

The first and perhaps most obvious way to reduce trading costs is to reduce the rebalancing frequency. This represents a trade-off between trading noise in the daily data and forgoing potential profits within this noise. For the neural networks, this changes the way they are being trained since they now do not minimize the error based on the next day's, but rather the next month's (21 days) return. The optimization is then repeated for these models, yielding new optimized weights. Next, the portfolio returns are computed for all risk targets, but the portfolios are only traded towards the optimized weights on the first day of each month.

Table 3: Annualized portfolio Sharpe ratios
(daily rebalancing, no TC)

Vola Target	EWMA	MLP	CNN	LSTM	RANDOM
2.0%	0.624	*0.842*	0.658	**0.942**	0.360
4.5%	0.624	*0.841*	0.659	**0.942**	0.360
7.0%	0.621	*0.838*	0.656	**0.936**	0.353
9.5%	0.611	*0.812*	0.644	**0.894**	0.344
12.0%	0.631	*0.800*	0.654	**0.865**	0.357
14.5%	0.645	*0.778*	0.640	**0.834**	0.366
17.0%	0.645	*0.746*	0.619	**0.798**	0.368
19.5%	0.646	*0.721*	0.606	**0.775**	0.366

Bold and italic table entries represent the highest and second highest Sharpe ratios per risk target, respectively.

Table 4: Ledoit-Wolf test for equality of Sharpe ratios against baseline estimator

Vola Target	MLP	CNN	LSTM	RANDOM
2.0%	0.0088	0.0076	*0.0096***	**0.0100***
4.5%	0.0088	0.0076	*0.0097***	**0.0100***
7.0%	0.0088	0.0076	*0.0097***	**0.0100***
9.5%	0.0088	0.0078	*0.0097**	**0.0101***
12.0%	0.0088	0.0080	*0.0098*	**0.0103***
14.5%	0.0089	0.0084	*0.0100*	**0.0105***
17.0%	0.0089	0.0087	*0.0102*	**0.0107***
19.5%	0.0089	0.0088	*0.0104*	**0.0108***

Bold and italic table entries represent the highest and second highest absolute test statistics per risk target, respectively.
$^{***}p < 0.01$, $^{**}p < 0.05$, $^{*}p < 0.1$ for a two-sided test.

Table 5: Average yearly portfolio trade volume
(daily rebalancing)

Vola Target	EWMA	MLP	CNN	LSTM	RANDOM
2.0%	**6.84**	29.75	*10.10*	42.20	40.42
4.5%	**15.39**	66.96	*22.73*	94.99	90.94
7.0%	**23.62**	103.51	*35.00*	147.04	140.77
9.5%	**30.01**	133.68	*45.26*	191.41	182.82
12.0%	**33.16**	152.85	*52.06*	220.65	210.17
14.5%	34**.82**	163.51	*57.12*	236.82	225.10
17.0%	**36.00**	170.29	*61.08*	246.75	234.08
19.5%	**36.65**	174.67	*63.73*	253.08	239.76

Bold and italic table entries represent the lowest and second lowest trade volume per risk target, respectively.

4.2.1. Predictive Accuracy

A first insight can again be obtained by assessing the predictive accuracy of the monthly return estimates. Table 6 provides an overview of the error metrics of these estimates.

First, it should be noted that these error metrics cannot be compared directly to those of the daily forecasts since the monthly returns are generally different from daily returns in their distributions. Nevertheless, looking at the relative error differences, the results are generally similar to the errors on the daily return forecasts. For the risk-free asset, the baseline estimator provides the best estimates, followed by the CNN model. In contrast, the market asset is again best predicted by the MLP and LSTM models. For this asset, the CNN also performs well, with only the EWMA estimator lagging behind the neural networks. Performing the Diebold-Mariano test against the

Table 6: MSE and MAE of monthly return predictions

	MSE		MAE	
	RF	Mkt	RF	Mkt
EWMA	**2.38e-07**	8.47e-04	**2.97e-04**	2.20e-02
MLP	3.57e-06***	**1.43e-06*****	1.39e-03***	**7.62e-04*****
CNN	*2.54e-07*	8.11e-05***	*3.52e-04****	*3.32e-03****
LSTM	5.97e-07***	*4.22e-05****	6.42e-04***	4.19e-03***
RANDOM	1.17e-05***	1.06e-01***	2.72e-03***	2.54e-01***

Bold and italic table entries represent the lowest and second lowest error per column, respectively.

Asterisks refer to the p-value of the Diebold-Mariano test statistics against the baseline estimator with $^{***}p < 0.01$, $^{**}p < 0.05$, $^{*}p < 0.1$ for a two-sided test.

baseline estimator also indicates a statistical significance of all error differentials with the exception of the only small edge of the EWMA against the CNN model when comparing the MSE of the risk-free asset. In the table, the significance of the Diebold-Mariano test statistics is also indicated by the asterisks. One should note that these significance indicators refer to an edge of the baseline estimator against the neural networks on the risk-free asset, and vice versa for the market asset. However, as was the case for the daily returns, the overall error on the risk-free asset is much smaller, potentially suggesting a superiority for the neural networks in portfolio performances, which are analyzed next.

4.2.2. Risk-Return Characteristics

A first assessment of the risk-return characteristics of the portfolios can again be obtained by plotting the returns against their volatility, as shown in Figure 16.

Again, the EWMA estimator is able to reduce the portfolio volatility the most, however, only accompanied by lower returns compared to the neural networks. The LSTM network provides higher returns but also higher volatility compared to the baseline estimator, again extending the "realized frontier" of the EWMA model. The MLP and CNN models provide similar risk-return characteristics in this setting, with both models showing both higher risks and returns compared to the EWMA model. These models, unlike the CNN, achieve a better risk-return trade-off compared to the EWMA and LSTM models. Notably, in the monthly rebalancing case, all portfolios are much closer to the line connecting the risk-free and the market asset compared to the daily setting, indicating an outperformance against the market asset as measured by the Sharpe ratio would be unlikely. Keeping in mind that these results were achieved without considering trade costs, the trade volumes of the portfolios are analyzed in the following.

4.2.3. Trade Volume

As can be seen in Table 7, the reduction of trade frequency is extremely effective in reducing the average trade volume of the portfolios. All neural networks show a lower trade volume than the baseline estimator in this setting, potentially also indicating a more stable estimate for monthly returns by these models. Since these trade volumes are in a range where strategies could be profitable when realistic trade costs are considered, the remainder of this section shows results net of trading costs. In fact, trade costs have changed significantly during the observation period and thus assuming them to be constant would not be a good approximation. Jones (2002) assembled average trading costs for stocks in the Dow Jones during the 20th century. Pillared on the presented time series of trading costs, a linear interpolation was performed for the time span from 1980-2000, starting with costs of 90 basis points in the year 1980 and dropping to 20 basis points in the year 2000. Thereafter, the trading costs were conservatively estimated to remain at 20 basis points until 2019. More specifically, these figures represent the relative one-way trading costs consisting of half of the average bid-ask spread plus the exchange commissions. Thus, the return on each day where trading occurs is reduced by the trade volume weighted trading costs. When considering trading costs, the portfolios move vertically in the risk-return space as can be seen comparing Figure 16 to Figure 17. The vertical movement can only be approximated in Table 7 due to the potentially different distribution of trade volumes during the observation period.

Due to its small trading volumes, the MLP achieves the highest returns when trading costs are considered, with only a narrow gap to the CNN model. However, the return distance to the LSTM and EWMA models widens significantly, which due to their similar trade volume continue to be close to each other. For the first time, the random model even underperforms the risk-free asset, illustrating the detrimental effect trading costs can have to unsubstantiated trading strategies. When instead of the

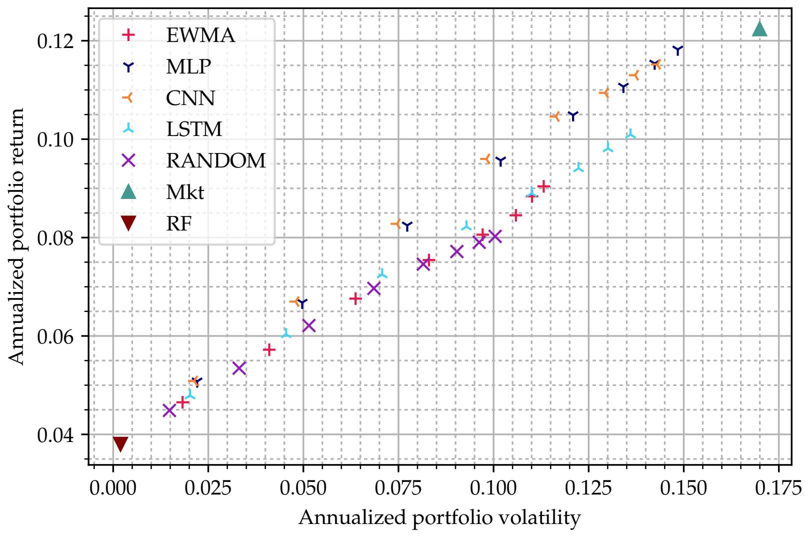

Figure 16: Portfolio performances in μ-σ-space (monthly rebalancing, no TC)

Table 7: Average yearly portfolio trade volume (monthly rebalancing)

Vola Target	EWMA	MLP	CNN	LSTM	RANDOM
2.0%	1.47	**0.53**	*0.71*	1.18	1.93
4.5%	3.32	**1.20**	*1.61*	2.67	4.34
7.0%	5.10	**1.80**	*2.43*	4.08	6.71
9.5%	6.41	**1.93**	*2.78*	5.06	8.76
12.0%	7.07	**1.58**	*2.68*	5.42	10.03
14.5%	7.29	**1.12**	*2.41*	5.42	10.64
17.0%	7.43	**0.85**	*2.30*	5.41	10.96
19.5%	7.50	**0.63**	*2.21*	5.35	11.17

Bold and italic table entries represent the lowest and second lowest trade volume per risk target, respectively.

portfolio volatility, the maximum drawdown is selected as a risk measure, very similar interpretations ensue, with the most notable difference being a slightly wider gap between the MLP and CNN models. For reference, this chart is presented in the Appendix.

4.2.4. Portfolio Sharpe Ratios

Finally, the Sharpe ratios of the portfolios are considered, with an overview given in Table 8.

Unsurprising when recalling the μ-σ plot, the MLP and CNN models achieve the highest Sharpe ratios of up to 0.529 for the MLP and 0.472 for the CNN. The Sharpe ratios of the EWMA and LSTM model, on the other hand, only reach 0.212 and 0.251, respectively. Considering the LSTM model had the highest Sharpe ratios and trade volumes in the daily rebalance setting, it is evident that this model sustained the largest reduction in portfolio returns. Potentially, this model gained its high returns through many small adjustments to the portfolio, which was severely restricted by the switch to a monthly rebalancing schedule. In contrast to the daily rebalancing case, the execution of the Ledoit-Wolf test suggests that the Sharpe ratio differentials of the MLP and CNN models against the baseline estimator are mostly significant at the 1% level, as indicated by the asterisks in the table. Overall, the reduction of trading frequency works well for the MLP and CNN models, providing support for Hypothesis 2, as these models outperform the baseline estimator even in the presence of trading costs. However, the LSTM model does not support this hypothesis under these circumstances but also does not perform worse than the baseline estimator.

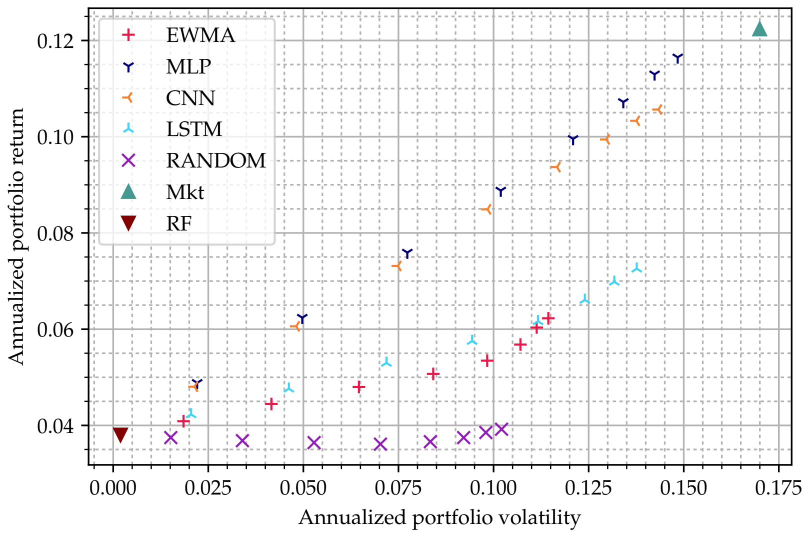

Figure 17: Portfolio performances in μ-σ-space (monthly rebalancing, with TC)

Table 8: Annualized portfolio Sharpe ratios (monthly rebalancing, with TC)

Vola Target	EWMA	MLP	CNN	LSTM	RANDOM
2.0%	0.155	**0.492*****	*0.471****	0.208	−0.036
4.5%	0.155	**0.491*****	*0.469****	0.208	−0.035
7.0%	0.154	**0.490*****	*0.469****	0.208	−0.030
9.5%	0.151	**0.498*****	*0.476****	0.207	−0.027
12.0%	0.157	**0.509*****	*0.477****	0.211	−0.017
14.5%	0.175	**0.516*****	*0.474****	0.226	−0.006
17.0%	0.201	**0.527*****	*0.475***	0.241	0.005
19.5%	0.212	**0.529*****	*0.472***	0.251	0.012

Bold and italic table entries represent the highest and second highest Sharpe ratios per risk target, respectively.
Asterisks refer to the p-value of the Ledoit-Wolf test statistics against the baseline estimator with $^{***}p < 0.01$, $^{**}p < 0.05$, $^{*}p < 0.1$ for a two-sided test.

4.3. Weight smoothing

As previously established, the low signal-to-noise ratio of historic returns results in a low certainty and high error rate in the forecasts. Similar to the underlying principle of exponentially smoothing previous returns, one could also conceptualize the smoothing of the weights resulting from the portfolio optimizations, such that the weights only change gradually for repeatedly deviating weights. Ideally, this would remove many of the noisy trades while retaining the beneficial shifts in the portfolio weights. One could either apply this smoothing to the weights optimized according to the daily or monthly forecasts or to the forecasts themselves, with the former approach being employed in the analysis. For this, similar to the EWMA estimator, the weights were smoothed in an exponential manner with a smoothing parameter of $\lambda = 0.94$. When smoothing over the optimized weights, the average lag of the weights in the resulting composition clearly is greater than one, and thus, the half-life and the center of mass of the chosen smoothing parameter were considered to find a return estimate better reflecting this property. These lie at 17 and 24 days, respectively, and thus the monthly predictions provide a good fit and the portfolio weights optimized using these forecasts were used. As no fixed trading costs are considered, the portfolios are traded towards these smoothed weights on a daily basis.

4.3.1. Trade volume

In order to assess the effectiveness of this methodology, the trade volumes are examined and presented in

Table 9.

Compared to the monthly rebalancing setting in Table 7, the smoothing of the weights results in a very similar and even slightly higher reduction in trade volumes. In particular, the rank of the models is unchanged, with the MLP model still achieving the lowest trade volume, followed by the CNN, LSTM and eventually the EWMA and RANDOM models. In light of these promising decreases in trade volumes and therefore corresponding trade costs, the analysis can now be focused on the risk-adjusted returns of the portfolios.

4.3.2. Risk-Return Characteristics

Again, the portfolios are plotted in the μ-σ-space for the first assessment of performance. Figure 18 provides this overview.

Like for the trade volumes, the ranking is again unchanged compared to the monthly rebalancing setting. Especially the MLP and CNN models are almost indistinguishable when comparing the two approaches. The LSTM and EWMA, on the other hand, perform better both in terms of risk as well as return when the smoothing method is chosen. The LSTM model can even widen its return margin against the EWMA model, indicating that more of the returns from the unsmoothed daily rebalancing case can be retained compared to monthly rebalancing, whilst showing less overall trade volume. Choosing the maximum drawdown as a risk metric again shows very similar results, implying an approximate proportionality between these measures in both the smoothed and monthly rebalancing settings. The corresponding figure can also be found in the Appendix.

4.3.3. Portfolio Sharpe Ratios

To quantify these implications of the further reductions in trade volumes, the Sharpe ratios of the portfolios in a setting with trade costs are considered next, as presented in Table 10.

Given the similarity in the risk-return figures, it is unsurprising that the Sharpe ratios between the monthly rebalancing and the smoothed rebalancing show a comparable dependency. The MLP model again leads in this metric, closely followed by the CNN model. According to the Ledoit-Wolf test, the outperformance against the EWMA model is slightly less significant in this setting due to the slightly increased Sharpe ratio of the baseline estimator. Up to an annualized volatility target of 12%, the Sharpe ratio differentials are still significant at the 1% level. The higher risk categories remain significant at the 5% level, with solely the 19.5% volatility target of the CNN model showing only significance at the 10% level. The increased Sharpe ratios of the LSTM model, on the other hand, are not enough to constitute significant outperformance against the EWMA estimator as measured by the Ledoit-Wolf test. Conversely, the increased Sharpe ratios of the baseline estimator itself now exhibit an outperformance against the RANDOM model, which is statistically significant at the 1% level. The weight smoothing approach thus underlines the findings of the monthly rebalancing setting and provide further support for Hypothesis 2 by showing significant outperformance of the MLP and CNN model against the baseline. Furthermore, the LSTM model now not only matches but also outperforms the EWMA model, however without the corresponding degree of statistical significance.

4.4. Sensitivity analysis

While the previous sections have given insights into the performance of the employed neural network estimators, it is important to check these results for robustness. Therefore, this section analyzes the sensitivity of these results towards hyperparameter variation, universe expansion, recalibration, and rerun variations.

4.4.1. Hyperparameter variation

Neural networks generally consist of a multitude of model parameters and hyperparameters. As described in Chapter 3, many theoretical and empirical factors were considered when choosing these hyperparameters. This, however, does not guarantee that these hyperparameters are ideal for the specific data set. A variation in these hyperparameters can have a significant impact on model performance, which can be utilized by specifying parts of the data as a validation data set and fine-tuning the hyperparameters based on the results obtained on this data. Alternatively, one can change these hyperparameters after the analysis to assess whether the choices made are stable across a wide range of model configurations or if the results are particularly (un-)favorable only for the narrow set of chosen hyperparameters. Since each hyperparameter can be adapted individually, all possible combinations of these adaptations could theoretically be evaluated. However, each new combination requires a retraining of the estimators and renewed optimization. Therefore, this approach is computationally very costly.

As a narrower approach, for each of the neural networks, both a smaller as well as a larger configuration in terms of layers and nodes was tested. For the larger configurations also a higher dropout probability of 0.6 was used, whereas the smaller configurations only used a dropout probability of 0.4. Finally, the batch size was increased (decreased) from 32 to 64 (16) in the larger (smaller) configuration. The final configurations are also schematically represented in the Appendix. The results of these repeated analyses are depicted in Figure 19. For each of the neural networks, the smaller and lighter markers aptly represent the performances of the smaller models, with the larger and darker markers indicating the larger models. Notably, for the LSTM and CNN models the scaling of the models seemingly did not affect the performance substantially. On the other hand, the MLP model also showed similar performance for the restricted model, but a worse performance for the larger

Table 9: Average yearly portfolio trade volume
(smoothed daily rebalancing)

Vola Target	EWMA	MLP	CNN	LSTM	RANDOM
2.0%	1.01	**0.55**	*0.62*	0.82	1.63
4.5%	2.25	**1.08**	*1.26*	1.76	3.67
7.0%	3.45	**1.44**	*1.77*	2.62	5.68
9.5%	4.42	**1.45**	*1.96*	3.24	7.39
12.0%	5.02	**1.18**	*1.93*	3.54	8.53
14.5%	5.30	**0.86**	*1.82*	3.62	9.15
17.0%	5.46	**0.65**	*1.79*	3.65	9.53
19.5%	5.56	**0.48**	*1.77*	3.65	9.75

Bold and italic table entries represent the lowest and second lowest trade volume per risk target, respectively.

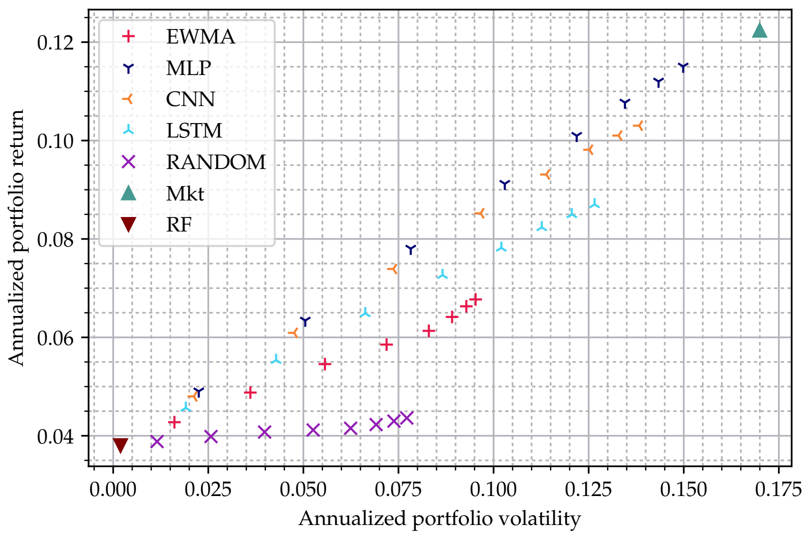

Figure 18: Portfolio performances in μ-σ-space
(smoothed daily rebalancing, with TC)

Table 10: Annualized portfolio Sharpe ratios
(smoothed daily rebalancing, with TC)

Vola Target	EWMA	MLP	CNN	LSTM	RANDOM
2.0%	0.296	**0.491*****	*0.472****	0.402	0.071***
4.5%	0.299	**0.503*****	*0.481****	0.405	0.071***
7.0%	0.297	**0.512*****	*0.487****	0.405	0.069***
9.5%	0.285	**0.517*****	*0.488****	0.400	0.060***
12.0%	0.281	**0.517*****	*0.484****	0.394	0.056***
14.5%	0.293	**0.518****	*0.480***	0.393	0.062***
17.0%	0.305	**0.516****	*0.474***	0.390	0.067***
19.5%	0.312	**0.514****	*0.470**	0.387	0.072***

Bold and italic table entries represent the highest and second highest Sharpe ratios per risk target, respectively.
Asterisks refer to the p-value of the Ledoit-Wolf test statistics against the baseline estimator with $^{***}p < 0.01$, $^{**}p < 0.05$, $^{*}p < 0.1$ for a two-sided test.

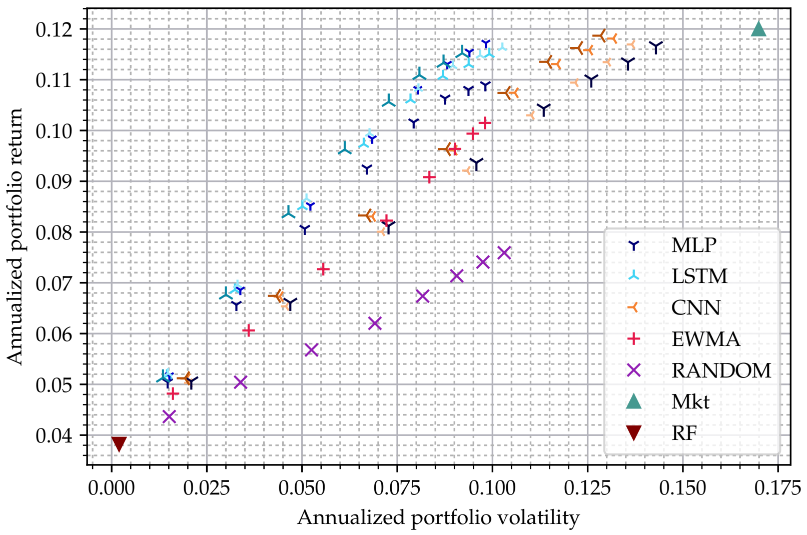

Figure 19: Portfolio performances in μ-σ-space
(small and large models, daily rebalancing, no TC)

model. While the larger model did, in fact, show higher returns, these came at the cost of over-proportionally higher volatilities, thus reducing the overall performance from a Sharpe ratio perspective. A potential explanation for the different performance characteristics for this model could be overfitting, as the nodes per layer were doubled, thereby quadratically increasing the parameters per fully connected layer, as apparent when looking at the construction of the weight matrix (compare the sample weight matrix in Equation 5). The addition of another fully connected layer could further have contributed to the overfitting of the model. Overall, it can be concluded that for the data set at hand, the LSTM and CNN models seem robust against changes in the model architecture, with the MLP model showing inferior performance for larger models. Thus, when implementing an MLP model, special attention should be paid to the choice of hyperparameters, and the usage of parts of the data set to validate hyperparameter choices seems advantageous.

4.4.2. Universe expansion

While the presented analysis considered a two-asset case, the implementation can easily be expanded to a multi-asset case. Thus, the two previously excluded assets representing the *high minus low (HML)* and *small minus big (SMB)* factors are added to the estimation and subsequent optimization. This provides further valuable insights, as unlike the risk-free asset, these factor assets are not completely uncorrelated to the market and therefore, can provide a better proxy for a realistic application of portfolio optimization. Figure 20 shows the performances of the portfolios optimized on the extended universe in the μ-σ-space. It is apparent that both the HML and SMB factor assets did underperform even the risk-free asset during the testing period. For approximately half of the volatility of the market asset, the HML return is similar to the risk-free return, and the SMB effectively has no return at all. These full period measures do not necessarily hold true for every sub-period, and outperformance can thus be achieved by shifting the portfolio weights towards assets with good risk-return trade-offs at any given day. In fact, all neural networks estimators seem to be better in this anticipatory shifting of portfolio weights, as the outperformance against the baseline estimator increases substantially. Furthermore, all neural networks achieve higher absolute returns compared to the market asset at the upper end of the risk targets, which was not achieved in the two-asset case. The portfolios' volatilities, on the other hand, are not affected substantially, resulting in greatly improved Sharpe ratios for these portfolios, as indicated by the slope of the "realized frontiers".

Overall, the expansion of the asset universe does not undermine the results of the main analysis based on the two-asset case, but rather provide further support for the usefulness of the neural network estimators. Additionally, the remarkable performance improvements obtained through the addition of two assets which on their own underperformed during the training period opens up further research questions like the inclusion of short assets (i.e., assets which track the inverse performance of a benchmark or index), which are also expected to underperform in the long run in this setting due to the positive expected return of the market, but can provide excess returns when held during times of financial distress. Furthermore, these results justify the expan-

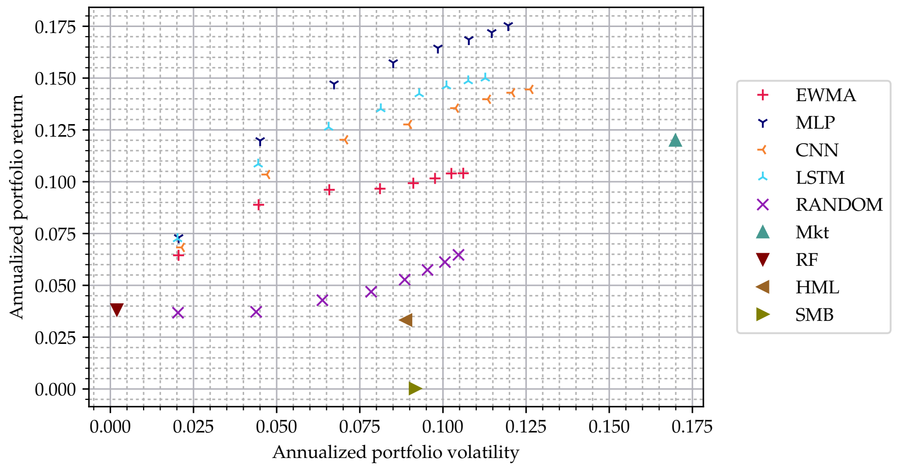

Figure 20: Portfolio performances in μ-σ-space (all assets, daily rebalancing, no TC)

sion of the analysis across a broader set of asset classes, thereby providing the optimization access to different risk-return profiles, which are more or less advantageous at any given moment. To achieve optimal performance, the portfolio composition, therefore, needs to be adapted over time, a task at which the incorporation of neural network estimators seemingly can be beneficial.

4.4.3. Recalibration

Since capital markets are constantly evolving, even a well-performing model trained on any given data set would become outdated at some point. Therefore, models need to be updated to reflect the information contained in more recent data. Neural networks can potentially deal with this phenomenon by being retrained on an expanding data set, also referred to as recalibration.

Theoretically, one could update the model parameters incrementally using a single stochastic gradient descent step whenever a single new observation is available. This is referred to as *incremental* or *online learning*. It is in so far advantageous as only the gradient for the most recent sample has to be computed, thus making it computationally more efficient. A more comprehensive overview of this technique is given by Gepperth and Hammer (2016).

On the other hand, completely retraining the model from a random initialization using the extended data set is referred to as *offline learning*. This method has the advantage that it is often easier to implement and monitor. However, due to the random initialization and the non-convex error function, it is also possible to reach a different local minimum during the next optimization, potentially causing a more drastic change in model behavior after each retraining. It is also possible to recalibrate the models at less frequent intervals, for example, once per month or once per year.

To assess the possible improvements that can be gained from this technique, a single recalibration step is added to the analysis. After the first half of the test data set is predicted, these observations are added to the training data set. Next, the model is retrained from a random initialization on the extended data, again preventing information leakage by introducing a gap between the (extended) training data and the (remaining) test data. Finally, a prediction on the second half of the test set is performed. The performance of the recalibrated and non-recalibrated models are plotted in Figure 21. Since the recalibration is only relevant for the neural networks, the EWMA and RANDOM estimators are omitted from the figure. The added larger and darker markers refer to the recalibrated models. The overall effect of recalibrating the model is not consistent, with the LSTM model showing only very minor differences, the MLP model showing a substantial improvement in returns accompanied by an under-proportional increase in volatility and the CNN even showing a small reduction in returns at similar volatilitie. Possibly, the different patterns recognized by the models in the additional data points were not equally recurring in the following testing period. Moreover, one should consider whether an extension of the training window is necessarily the ideal approach, or whether a shifting of the training window would be more efficient as this would remove the oldest data which potentially is least relevant for the prediction.

An alternative approach would be to extend the window, but weight each sample according to its recency, thereby overweighing newer observations compared to older ones. An overview of this approach is given by Morantz, Whalen, and Zhang (2008). While different weighting schemes can be conceptualized, following the argumentation of the EWMA estimator itself, an exponential decay of the importance of previous samples could be used as a starting point. In conclusion, the recalibration of the model can substantially affect the performance of the model in the long run but is not a

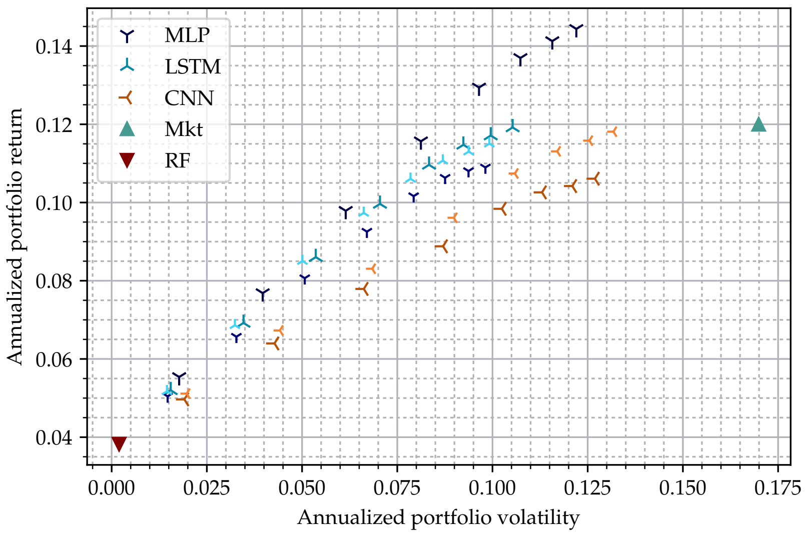

Figure 21: Portfolio performances in μ-σ-space (recalibrated, daily rebalancing, no TC)

guaranteed way to improve performance. The different introduced approaches need to be carefully evaluated when implementing a neural network, especially when considering the computational burden of frequently recalibrating large models.

4.4.4. Prediction consistency

Since the gradient descent algorithm starts with random initialization of the model, it is possible that different local minima are obtained as the objective function is often non-convex. While Du, Lee, Li, Wang, and Zhai (2018) show that overparameterized networks can find the global minimum of the error function, Choromanska, Henaff, Mathieu, Arous, and LeCun (2014) and Goodfellow et al. (2016) conclude that it is generally more likely to obtain local minima close to the global minimum. Furthermore, at the point at which the training process is stopped, as defined by the number of epochs, different model parameters can follow from different paths towards the minimum as a result of the random initialization. Repeating the analysis thus gives insights into how different the obtained optimization results are. Additionally, repeating the analysis supports the reproducibility of results. While commonly a fixed random seed is used in the analysis, this does not guarantee that the outcomes based on this seed are representative for any starting values. Furthermore, it is not easily possible to fix all occurrences of randomness in the interplay of different programming libraries used. Thus, the main analysis with daily rebalancing was repeated three times. Figure 22 gives a first overview of the results, with again the EWMA and RANDOM estimators being omitted. Compared to the initial analysis, the two additional repetitions are again presented with smaller and lighter as well as larger and darker markers. The LSTM model shows the smallest rerun variation across the samples, with both repeated "realized frontiers" closely tracking the initial performances. For the MLP model, one repetition shows higher returns for similar volatilities, whereas the other shows higher returns at higher volatilities, with the overall results being similar. For the CNN model, on the other hand, both repetitions show up to two percentage points lower returns at similar volatilities, marking the largest deviation across the models. Most importantly, the overall impression of model performances remains valid when comparing multiple runs of model training, especially when considering the sensitivity of portfolio optimizations to the estimation of the expected asset returns. This is especially pronounced when the case of daily rebalancing based on single day return predictions is considered where forecasts are particularly unstable. Thus, the analysis of rerun consistency was also performed for the daily smoothed portfolios based on monthly predicted returns. As apparent from Figure 23, the smoothing in combination with the substantially more stable monthly returns, results in almost unaltered portfolio performances across multiple runs.

Overall, the repetition of the analysis provides support for the main findings by corroborating that the observed portfolio performances based on the neural network estimates were not coincidentally obtained from an exceptional manifestation of otherwise clearly worse-performing models. However, for practical applications, variation between runs would obviously pose a greater problem. In this case, one could consider choosing either the return forecasts or the optimized weights based

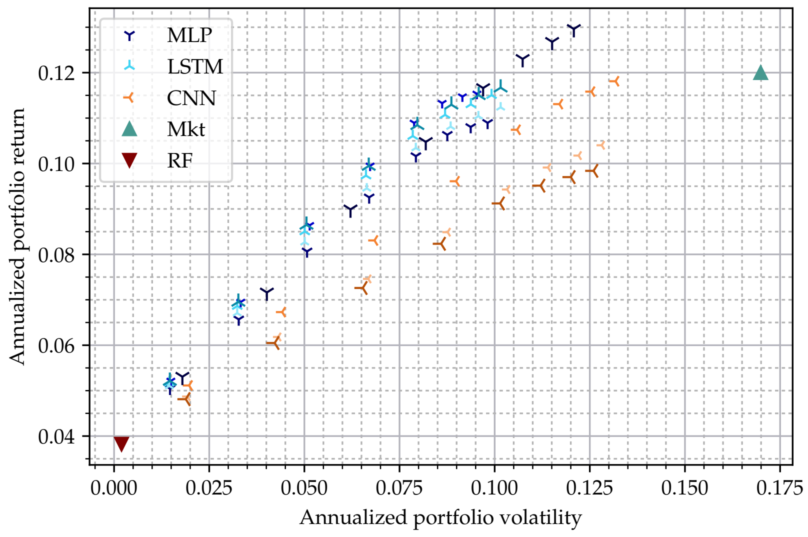

Figure 22: Portfolio performances in μ-σ-space (rerun consistency, daily rebalancing, no TC)

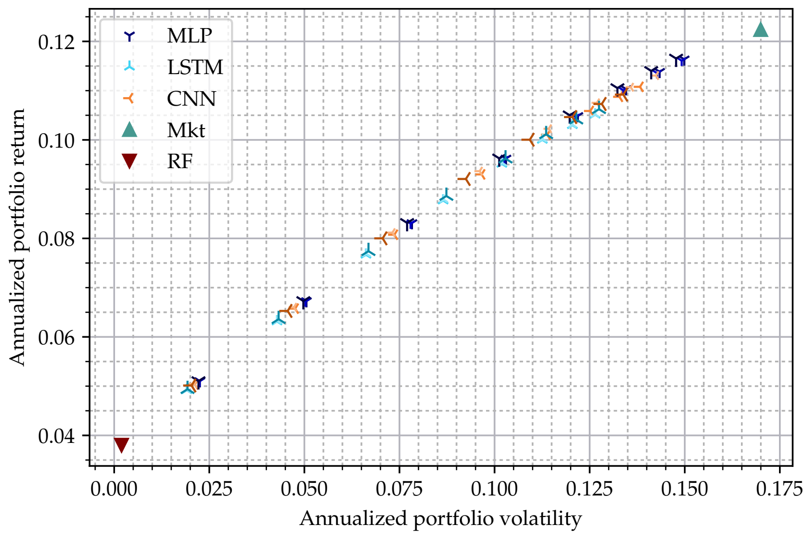

Figure 23: Portfolio performances in μ-σ-space (rerun consistency, smoothed daily rebalancing, no TC)

on the mean of multiple runs, which would, however, again sharply increase the computational demands of the implementation.

4.5. Discussion

As could be shown in the previous sections of this thesis and the resulting partial acceptance of the formulated hypotheses, neural networks were found to be beneficial in a portfolio selection framework for the data set under examination. However, several challenges arise with the generalization of these results.

Firstly, certain common assumptions in academia employed in this thesis do not entirely reflect the real-world dynamics of financial markets. One of the most common deviations is that portfolios can be optimized after observing the closing prices in real-time, and the resulting allocations are also obtained based on these very prices. Arguably, this is rather a result of data availability and more specifically, the lack of intraday prices, whose presence would allow modeling a time gap between optimization and trading prices. However, this most likely does not impact the fundamental interpretability of results, as price differences between optimization and execution prices are not one-sided and likely provide netting effects over time. A more severe impact likely arises due to the negligence of tax effects, as potential savings from tax deferrals are not realized when sell trades on assets with taxable gains are executed. Yet, the baseline model also exhibits this negligence and in fact, boasts even higher trading volumes in the monthly and smoothed rebalancing settings compared to the neural network estimators, thus potentially even strengthening the neural networks' edge. In order to avoid such trades, a buy and hold strategy could also be implemented and indeed would be a viable option to compare against the neural networks in future analyses, especially since the market asset provided the highest absolute return in many settings.

Secondly, implementations of neural networks, especially in the analysis of time series, often are subject to explicit and implicit research biases. Hence, special care was taken to avoid common pitfalls within the process. This includes the prevention of information leakage through complete separation of training and testing data for both feature engineering and data scaling. At the same time, the model hyperparameters were chosen based on academic consensus and best practices rather than repeated iterations on the same data set, which would immediately invalidate the generalization power.

Thirdly, while it lies beyond the scope of this analysis, the additional model complexity brought on by the neural networks can potentially lead to behavioral consequences when implemented in a real-time environment. One example could be a reduced model confidence in times of underperformance, possibly leading to adverse decision-making. This is especially true for neural networks, as their mapping process of inputs and outputs is poorly understood and can be counterintuitive at times (e.g. Szegedy et al., 2013).

Finally, a higher degree of reliability on the results can only be obtained by repeating the analysis for additional data sets reflecting other asset classes, markets and time periods, thereby assessing whether the properties of the neural networks in a portfolio optimization framework retain their validity across the multitude of possible applications found in financial data.

5. Conclusion

After a brief introduction of the dynamic portfolio optimization framework, three types of neural networks with potentially advantageous properties for the estimation of expected asset returns are presented: the Multilayer Perceptron, Convolutional Neural Network, and Long Short-Term Memory Network. The usefulness of these neural networks is then evaluated by using their asset return estimates within a dynamic portfolio optimization framework and an asset universe consisting of a risk-free and a market asset. This thesis finds evidence that in the analyzed data set the neural networks were able to outperform a more traditional EWMA estimator in most settings. In a daily rebalancing scenario and utilizing daily return estimates, the outperformance was mostly insignificant, and the neural network estimates would have resulted in infeasible trade volumes, thus providing only partial support for the corresponding hypothesis. Using monthly return estimates and a likewise monthly rebalancing frequency sharply reduced the trade volumes of the neural networks, resulting in a mostly significant outperformance when historically adjusted trading costs are considered. Applying a smoothing function to the optimized portfolio weights and again using a daily rebalancing frequency results in even lower trade volumes, thus further strengthening the outperformance of the neural networks and yielding an increase in Sharpe ratios of up to 0.2 against the EWMA baseline. Therefore, support is presented for the hypothesis associated with outperformance in the scenarios aiming at reduced trading volumes. The results were further validated through the assessment of model sensitivities towards hyperparameter variations, recalibrations, and repeated executions. Moreover, the analysis of an asset universe extended beyond two assets has shown remarkable results, substantiating the future analysis of other combinations of asset classes, market regions, and time periods.

Besides these naturally emerging research questions, the incorporation of neural network research into the portfolio optimization framework opens a multitude of additional research opportunities. Related to this thesis, weighting schemes for individual observations during the training phase could be considered in order to increase the relevance of more recent samples. Furthermore, instead of modeling each asset's time series individually, returns of other assets could be included as additional features to implicitly reflect the dependency structure between asset returns in the model. Neural networks are not restricted to the use of historic returns either, enabling the inclusion of fundamental or otherwise relevant data into the estimation. Similarly, neural networks can directly be incorporated in the estimation of asset covariances or, taking it one step further, the optimal portfolio weights (and thus performance) could directly be targeted by a neural network. Thereby the moment estimation and optimization steps can be combined in a single architecture, providing a transition to reinforcement learning models. Whenever neural networks are applied in the context of financial data, however, one needs to pay particular attention to the unique challenges arising from these models, such that a compromise of the result validity through repeated analysis on a single data set or information leakage can be prevented. As reinforced by this thesis, financial markets by their very nature continue to provide a challenging environment for reliable forecasting. Depending on the implementation, advanced models such as neural networks might be able to provide improvements in academic analyses, but one always needs to keep in mind the inherent limitations of forecasts in this setting. For practitioners, the potential adverse effects of additional model complexity should not be underestimated, and if implemented, research into the traceability of the neural network estimates is demanded.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from http://tensorflow.org/ (Software available from tensorflow.org)

Alpaydin, E. (2010). *Introduction to machine learning*. Cambridge: MIT Press.

Antonacci, G. (2014). *Dual Momentum Investing: An Innovative Approach for Higher Returns with Lower Risk*. New York: McGraw-Hill Education.

Asness, C., Frazzini, A., Israel, R., & Moskowitz, T. (2014, 9). Fact, Fiction, and Momentum Investing. *The Journal of Portfolio Management*, *40*(5), 75–92. Retrieved from http://jpm.pm-research.com/lookup/doi/10.3905/jpm.2014.40.5.075 doi: 10.3905/jpm.2014.40.5.075

Azoff, E. M. (1994). *Neural network time series forecasting of financial markets* (1st ed.). New York: John Wiley & Sons.

Bartlett, M. S. (1946). On the Theoretical Specification and Sampling Properties of Autocorrelated Time-Series. *Supplement to the Journal of the Royal Statistical Society*, *8*(1), 27–41. Retrieved from https://www.jstor.org/stable/2983611?origin=crossref doi: 10.2307/2983611

Baz, J., Granger, N. M., Harvey, C. R., Le Roux, N., & Rattray, S. (2015). Dissecting Investment Strategies in the Cross Section and Time Series. *SSRN Electronic Journal*. Retrieved from http://www.ssrn.com/abstract=2695101 doi: 10.2139/ssrn.2695101

Bengio, Y. (2009). Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning*, *2*(1), 1–127. Retrieved from http://www.nowpublishers.com/article/Details/MAL-006 doi: 10.1561/2200000006

Bengio, Y. (2012). Practical Recommendations for Gradient-Based Training of Deep Architectures. (arXiv:1206.5533v2 [cs.LG]). Retrieved from http://deeplearning.net/software/pylearn2

Bengio, Y., Simard, P., & Frasconi, P. (1994, 3). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166. Retrieved from https://ieeexplore.ieee.org/document/279181/ doi: 10.1109/72.279181

Bergerson, K., & Wunsch, D. C. (1991). A commodity trading model based on a neural network-expert system hybrid. In *Ijcnn-91-seattle international joint conference on neural networks* (Vol. i, pp. 289–293). IEEE. Retrieved from http://ieeexplore.ieee.org/document/155192/ doi: 10.1109/IJCNN.1991.155192

Berry, M. J. A., & Linoff, G. (1997). *Data Mining Techniques*. New York: John Wiley & Sons.

Best, M. J., & Grauer, R. R. (1991, 4). On the Sensitivity of Mean-Variance-Efficient Portfolios to Changes in Asset Means: Some Analytical and Computational Results. *Review of Financial Studies*, *4*(2), 315–342. Retrieved from https://academic.oup.com/rfs/article-lookup/doi/10.1093/rfs/4.2.315 doi: 10.1093/rfs/4.2.315

Black, F., & Litterman, R. (1992, 9). Global Portfolio Optimization. *Financial Analysts Journal*, *48*(5), 28–43. Retrieved from https://www.tandfonline.com/doi/full/10.2469/faj.v48.n5.28 doi: 10.2469/faj.v48.n5.28

Blum, A. (1992). *Neural Networks in C++*. New York: John Wiley & Sons.

Borovykh, A., Bohte, S., & Oosterlee, C. W. (2017, 3). Conditional Time Series Forecasting with Convolutional Neural Networks. (arXiv:1703.04691v5 [stat.ML]). Retrieved from http://arxiv.org/abs/1703.04691

Brent, R. P. (1973). Algorithms for Minimization Without Derivatives. In *Algorithms for minimization without derivatives* (chap. 3-4). Englewood Cliffs: Prentice-Hall.

Brown, D. B., & Smith, J. E. (2011, 10). Dynamic Portfolio Optimization with Transaction Costs: Heuristics and Dual Bounds. *Management Science*, *57*(10), 1752–1770. Retrieved from http://pubsonline.informs.org/doi/abs/10.1287/mnsc.1110.1377 doi: 10.1287/mnsc.1110.1377

Chen, A.-S., Leung, M. T., & Daouk, H. (2003, 5). Application of neural networks to an emerging financial market: forecasting and trading the Taiwan Stock Index. *Computers & Operations Research*, *30*(6), 901–923. Retrieved from https://linkinghub.elsevier.com/retrieve/pii/S0305054802000370 doi: 10.1016/S0305-0548(02)00037-0

Chiappori, P.-A., & Paiella, M. (2011, 12). Relative risk aversion is constant: evidence from panel data. *Journal of the European Economic Association*, *9*(6), 1021–1052. Retrieved from https://academic.oup.com/jeea/article-lookup/doi/10.1111/j.1542-4774.2011.01046.x doi: 10.1111/j.1542-4774.2011.01046.x

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014, 6). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. (arXiv:1406.1078v3 [cs.CL]). Retrieved from http://arxiv.org/abs/1406.1078

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., & LeCun, Y. (2014, 11). The Loss Surfaces of Multilayer Networks. (arXiv:1412.0233v3 [cs.LG]). Retrieved from http://arxiv.org/abs/1412.0233

Ciresan, D., Meier, U., & Schmidhuber, J. (2012, 6). Multi-column deep neural networks for image classification. In *2012 ieee conference on computer vision and pattern recognition* (pp. 3642–3649). IEEE. Retrieved from http://ieeexplore.ieee.org/document/6248110/ doi: 10.1109/CVPR.2012.6248110

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011, 3). Natural Language Processing (almost) from Scratch. (arXiv:1103.0398v1 [cs.LG]). Retrieved from http://arxiv.org/abs/1103.0398

Cootner, P. H. (1964). *The random character of stock market prices*. Cambridge: MIT Press.

Cybenko, G. (1989, 12). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, *2*(4), 303–314. Retrieved from http://link.springer.com/10.1007/BF02551274 doi: 10.1007/BF02551274

Deboeck, G. J. (1994). *Trading on the edge: neural, genetic, and fuzzy systems for chaotic financial markets* (Vol. 39). New York: John Wiley & Sons.

Diebold, F. X. (2015, 1). Comparing Predictive Accuracy, Twenty Years Later: A Personal Perspective on the Use and Abuse of Diebold–Mariano Tests. *Journal of Business & Economic Statistics*, *33*(1), 1–9. Retrieved from http://www.tandfonline.com/doi/abs/10.1080/07350015.2014.983236 doi: 10.1080/07350015.2014.983236

Diebold, F. X., & Mariano, R. S. (1995, 7). Comparing Predictive Accuracy. *Journal of Business & Economic Statistics*, *13*(3), 253–263. Retrieved from http://www.tandfonline.com/doi/abs/10.1080/07350015.1995.10524599 doi: 10.1080/07350015.1995.10524599

Di Persio, L., & Honchar, O. (2016). Artificial neural networks architectures for stock price prediction: Comparisons and applications. *International Journal of Circuits, Systems and Signal Processing*, *10*, 403–413. Retrieved from http://www.naun.org/main/NAUN/circuitssystemssignal/2016/b482005-303.pdf

Du, S. S., Lee, J. D., Li, H., Wang, L., & Zhai, X. (2018, 11). Gradient Descent Finds Global Minima of Deep Neural Networks. (arXiv:1811.03804v4 [cs.LG]). Retrieved from http://arxiv.org/abs/1811.03804

Dumas, B., & Luciano, E. (1991, 6). An Exact Solution to a Dynamic Portfolio Choice Problem under Transactions Costs. *The Journal of Finance*, *46*(2), 577. Retrieved from https://www.jstor.org/stable/2328837?origin=crossref doi: 10.2307/2328837

Dybvig, P. H. (1984, 3). Short Sales Restrictions and Kinks on the Mean Variance Frontier. *The Journal of Finance*, *39*(1), 239–244. Retrieved from http://doi.wiley.com/10.1111/j.1540-6261.1984.tb03871.x doi: 10.1111/j.1540-6261.1984.tb03871.x

Egmont-Petersen, M., de Ridder, D., & Handels, H. (2002, 10). Image processing with neural networks—a review. *Pattern Recognition*, *35*(10), 2279–2301. Retrieved from https://linkinghub.elsevier.com/retrieve/pii/S0031320301001789 doi: 10.1016/S0031-3203(01)00178-9

Elman, J. L. (1990, 6). Finding structure in time. *Cognitive Science*, *14*(2), 179–211. Retrieved from http://doi.wiley.com/10.1016/0364

-0213(90)90002-E doi: 10.1016/0364-0213(90)90002-E

Elton, E. J., & Gruber, M. J. (1974). On the Optimality of Some Multiperiod Portfolio Selection Criteria. *The Journal of Business*, *47*(2), 231–243. doi: 10.1086/295633

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017, 2). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, *542*(7639), 115–118. Retrieved from http://www.nature.com/articles/nature21056 doi: 10.1038/nature21056

Fama, E. F. (1970, 5). Efficient Capital Markets: A Review of Theory and Empirical Work. *The Journal of Finance*, *25*(2), 383–417. Retrieved from https://www.jstor.org/stable/2325486?origin=crossref doi: 10.2307/2325486

Fama, E. F., & French, K. R. (1993, 2). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, *33*(1), 3–56. Retrieved from https://linkinghub.elsevier.com/retrieve/pii/0304405X93900235 doi: 10.1016/0304-405X(93)90023-5

Fama, E. F., & French, K. R. (2019). *Kenneth r. french - data library.* Retrieved 2019-08-08, from https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html ([online] Available at: https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html [Accessed 2019-08-08])

Gal, Y., & Ghahramani, Z. (2015, 12). A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. (arXiv:1512.05287v5 [stat.ML]). Retrieved from http://arxiv.org/abs/1512.05287

Gârleanu, N., & Pedersen, L. H. (2018, 8). Efficiently Inefficient Markets for Assets and Asset Management. *The Journal of Finance*, *73*(4), 1663–1712. Retrieved from http://doi.wiley.com/10.1111/jofi.12696 doi: 10.1111/jofi.12696

Gately, E. (1995). *Neural networks for financial forecasting*. New York: John Wiley & Sons.

Gepperth, A., & Hammer, B. (2016). Incremental learning algorithms and applications. *ESANN 2016 - 24th European Symposium on Artificial Neural Networks*(April), 357–368.

Gers, F. A., & Schmidhuber, J. (2000). Recurrent nets that time and count. In *Proceedings of the ieee-inns-enns international joint conference on neural networks. ijcnn 2000. neural computing: New challenges and perspectives for the new millennium* (pp. 189–194). IEEE. Retrieved from http://ieeexplore.ieee.org/document/861302/ doi: 10.1109/IJCNN.2000.861302

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, & M. Dudík (Eds.), *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (Vol. 15, pp. 315–323). Fort Lauderdale: PMLR. Retrieved from http://proceedings.mlr.press/v15/glorot11a.html

Goetzmann, W. N., & Huang, S. (2018, 12). Momentum in Imperial Russia. *Journal of Financial Economics*, *130*(3), 579–591. Retrieved from https://linkinghub.elsevier.com/retrieve/pii/S0304405X18301843 doi: 10.1016/j.jfineco.2018.07.008

Goldberg, Y. (2016, 11). A Primer on Neural Network Models for Natural Language Processing. *Journal of Artificial Intelligence Research*, *57*, 345–420. Retrieved from https://jair.org/index.php/jair/article/view/11030 doi: 10.1613/jair.4992

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. Cambridge: MIT Press.

Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H., & Schmidhuber, J. (2009, 5). A Novel Connectionist System for Unconstrained Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *31*(5), 855–868. Retrieved from http://ieeexplore.ieee.org/document/4531750/ doi: 10.1109/TPAMI.2008.137

Graves, A., Mohamed, A.-r., & Hinton, G. (2013, 3). Speech Recognition with Deep Recurrent Neural Networks. (arXiv:1303.5778v1 [cs.NE]). Retrieved from http://arxiv.org/abs/1303.5778

Grefenstette, E., Blunsom, P., de Freitas, N., & Hermann, K. M. (2014, 4). A Deep Architecture for Semantic Parsing. (arXiv:1404.7296v1 [cs.CL]). Retrieved from http://arxiv.org/abs/1404.7296

Hahnloser, R. H. R., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., & Seung, H. S. (2000, 6). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, *405*(6789), 947–951. Retrieved from http://www.nature.com/articles/35016072 doi: 10.1038/35016072

Harvey, D., Leybourne, S., & Newbold, P. (1997, 6). Testing the equality of prediction mean squared errors. *International Journal of Forecasting*, *13*(2), 281–291. Retrieved from https://linkinghub.elsevier.com/retrieve/pii/S0169207096007194 doi: 10.1016/S0169-2070(96)00719-4

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. New York: Springer. Retrieved from http://link.springer.com/10.1007/978-0-387-84858-7 doi: 10.1007/978-0-387-84858-7

Hayou, S., Doucet, A., & Rousseau, J. (2018, 5). On the Selection of Initialization and Activation Function for Deep Neural Networks. (arXiv:1805.08266v2 [stat.ML]). Retrieved from http://arxiv.org/abs/1805.08266

Hill, T., O'Connor, M., & Remus, W. (1996, 7). Neural Network Models for Time Series Forecasts. *Management Science*, *42*(7), 1082–1092. Retrieved from http://pubsonline.informs.org/doi/abs/10.1287/mnsc.42.7.1082 doi: 10.1287/mnsc.42.7.1082

Hinton, G., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012, 7). Improving neural networks by preventing co-adaptation of feature detectors. (arXiv:1207.0580v1 [cs.NE]). Retrieved from http://arxiv.org/abs/1207.0580

Hochreiter, S. (1991). *Untersuchungen zu dynamischen neuronalen Netzen [Investigations of dynamic neural nets]* (Unpublished doctoral dissertation). Technical University of Munich.

Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. A field guide to dynamical recurrent neural networks. IEEE Press.

Hochreiter, S., & Schmidhuber, J. (1997, 11). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. Retrieved from http://www.mitpressjournals.org/doi/10.1162/neco.1997.9.8.1735 doi: 10.1162/neco.1997.9.8.1735

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, *4*(2), 251–257. Retrieved from https://linkinghub.elsevier.com/retrieve/pii/089360809190009T doi: 10.1016/0893-6080(91)90009-T

Hurst, B., Ooi, Y. H., & Pedersen, L. H. (2017). A Century of Evidence on Trend-Following Investing. *SSRN Electronic Journal*. Retrieved from https://www.ssrn.com/abstract=2993026 doi: 10.2139/ssrn.2993026

Ioffe, S., & Szegedy, C. (2015, 2). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. (arXiv:1502.03167v3 [cs.LG]). Retrieved from http://arxiv.org/abs/1502.03167

Jegadeesh, N., & Titman, S. (1993, 3). Returns to Buying Winners and Selling Losers: Implications for Stock Market Efficiency. *The Journal of Finance*, *48*(1), 65–91. Retrieved from http://doi.wiley.com/10.1111/j.1540-6261.1993.tb04702.x doi: 10.1111/j.1540-6261.1993.tb04702.x

Jones, C. M. (2002). A century of stock market liquidity and trading costs. *SSRN Electronic Journal*.

Jordan, M. I. (1997). Serial Order: A Parallel Distributed Processing Approach. In *Advances in psychology, volume 121* (pp. 471–495). Retrieved from https://linkinghub.elsevier.com/retrieve/pii/S0166411597801112 doi: 10.1016/S0166-4115(97)80111-2

J.P. Morgan. (1996). *J.p. morgan/reuters riskmetricstm- technical document.* New York: J.P. Morgan.

Kaastra, I., & Boyd, M. (1996, 4). Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, *10*(3), 215–236. Retrieved from http://linkinghub.elsevier.com/retrieve/pii/0925231295000399 doi: 10.1016/0925-2312(95)00039-9

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. In *Proceedings of the 52nd annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 655–665). Stroudsburg: Association for Computational Linguistics. Retrieved from http://aclweb.org/anthology/P14-1062 doi: 10.3115/v1/P14-1062

Kang, L., Ye, P., Li, Y., & Doermann, D. (2014, 6). Convolutional Neural Networks for No-Reference Image Quality Assessment. In *2014 ieee conference on computer vision and pattern recognition* (pp. 1733–1740). IEEE. Retrieved from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6909620 doi: 10.1109/CVPR.2014.224

Kaufman, S., Rosset, S., & Perlich, C. (2011). Leakage in data mining. In *Proceedings of the 17th acm sigkdd international conference on knowledge discovery and data mining - kdd '11* (p. 556). New York: ACM Press. Retrieved from http://dl.acm.org/citation.cfm?doid=2020408.2020496 doi: 10.1145/2020408.2020496

Kim, Y. (2014, 8). Convolutional Neural Networks for Sentence Classification. (arXiv:1408.5882v2 [cs.CL]). Retrieved from http://arxiv.org/abs/1408.5882

Kingma, D. P., & Ba, J. (2014, 12). Adam: A Method for Stochastic Optimization. (arXiv:1412.6980v9 [cs.LG]). Retrieved from http://arxiv.org/abs/1412.6980

Krauss, C., Do, X. A., & Huck, N. (2017). Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500. *European Journal of Operational Research*, *259*(2), 689–702. Retrieved from http://dx.doi.org/10.1016/j.ejor.2016.10.031 doi: 10.1016/j.ejor.2016.10.031

Laptev, N., Yosinski, J., Li, L. E., & Smyl, S. (2017). Time-series extreme event forecasting with neural networks at uber. *International Conference on Machine Learning*, *34*, 1–5.

Lawrence, S., Giles, C., Ah Chung Tsoi, & Back, A. (1997). Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, *8*(1), 98–113. Retrieved from http://ieeexplore.ieee.org/document/554195/ doi: 10.1109/72.554195

LeCun, Y., Bengio, Y., & Hinton, G. (2015, 5). Deep learning. *Nature*, *521*(7553), 436–444. Retrieved from http://www.nature.com/articles/nature14539 doi: 10.1038/nature14539

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., & Jackel, L. D. (1989, 12). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*(4), 541–551. Retrieved from http://www.mitpressjournals.org/doi/10.1162/neco.1989.1.4.541 doi: 10.1162/neco.1989.1.4.541

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., & Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems* (pp. 396–404).

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. Retrieved from http://ieeexplore.ieee.org/document/726791/ doi: 10.1109/5.726791

Ledoit, O., & Wolf, M. (2003, 12). Improved estimation of the covariance matrix of stock returns with an application to portfolio selection. *Journal of Empirical Finance*, *10*(5), 603–621. Retrieved from https://linkinghub.elsevier.com/retrieve/pii/S0927539803000070 doi: 10.1016/S0927-5398(03)00007-0

Ledoit, O., & Wolf, M. (2004, 2). A well-conditioned estimator for large-dimensional covariance matrices. *Journal of Multivariate Analysis*, *88*(2), 365–411. Retrieved from https://linkinghub.elsevier.com/retrieve/pii/S0047259X03000964 doi: 10.1016/S0047-259X(03)00096-4

Ledoit, O., & Wolf, M. (2008, 12). Robust performance hypothesis testing with the Sharpe ratio. *Journal of Empirical Finance*, *15*(5), 850–859. Retrieved from https://linkinghub.elsevier.com/retrieve/pii/S0927539808000182 doi: 10.1016/j.jempfin.2008.03.002

Lempérière, Y., Deremble, C., Seager, P., Potters, M., & Bouchaud, J. P. (2014, 4). Two centuries of trend following. (arXiv:1404.3274v1 [q-fin.PM]). Retrieved from http://arxiv.org/abs/1404.3274

Li, D., & Ng, W.-L. (2000, 7). Optimal Dynamic Portfolio Selection: Multiperiod Mean-Variance Formulation. *Mathematical Finance*, *10*(3), 387–406. Retrieved from http://doi.wiley.com/10.1111/1467-9965.00100 doi: 10.1111/1467-9965.00100

Li, F.-F. (2019). *Stanford University CS231n: Optimization.* ([online] Available at: http://cs231n.github.io/optimization-1/ [Accessed 2019-09-08])

Liew, J. K., & Mayster, B. (2017, 12). Forecasting ETFs with Machine Learning Algorithms. *The Journal of Alternative Investments*, *20*(3), 58–78. Retrieved from http://jai.iijournals.com/lookup/doi/10.3905/jai.2018.20.3.058 doi: 10.3905/jai.2018.20.3.058

Lim, B., Zohren, S., & Roberts, S. (2019). Enhancing Time Series Momentum Strategies Using Deep Neural Networks. (arXiv:1904.04912v1 [stat.ML]). Retrieved from http://arxiv.org/abs/1904.04912

Livni, R., Shalev-Shwartz, S., & Shamir, O. (2014, 10). On the Computational Efficiency of Training Neural Networks. (arXiv:1410.1141v2 [cs.LG]). Retrieved from http://arxiv.org/abs/1410.1141

Lopez, J. A., & Walter, C. A. (2002). Evaluating Covariance Matrix Forecasts in a Value-at-Risk Framework. *SSRN Electronic Journal*. Retrieved from http://www.ssrn.com/abstract=305279 doi: 10.2139/ssrn.305279

Markowitz, H. (1952, 3). Portfolio Selection. *The Journal of Finance*, *7*(1), 77. Retrieved from https://www.jstor.org/stable/2975974?origin=crossref doi: 10.2307/2975974

Masters, D., & Luschi, C. (2018, 4). Revisiting Small Batch Training for Deep Neural Networks. (arXiv:1804.07612v1 [cs.LG]). Retrieved from http://arxiv.org/abs/1804.07612

McCulloch, W. S., & Pitts, W. (1943, 12). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133. Retrieved from http://link.springer.com/10.1007/BF02478259 doi: 10.1007/BF02478259

Merton, R. C. (1972, 9). An Analytic Derivation of the Efficient Portfolio Frontier. *The Journal of Financial and Quantitative Analysis*, *7*(4), 1851. Retrieved from https://www.jstor.org/stable/2329621?origin=crossref doi: 10.2307/2329621

Michaud, R. O. (1989, 1). The Markowitz Optimization Enigma: Is 'Optimized' Optimal? *Financial Analysts Journal*, *45*(1), 31–42. Retrieved from https://www.tandfonline.com/doi/full/10.2469/faj.v45.n1.31 doi: 10.2469/faj.v45.n1.31

Müller, A. C., & Guido, S. (2016). *Introduction to Machine Learning with Python*. Sebastopol: OŔeilly Media. Retrieved from https://www.oreilly.com/library/view/introduction-to-machine/9781449369880/

Morantz, B., Whalen, T., & Zhang, G. P. (2008). Neural Network Time Series Forecasting Using Recency Weighting. In *Encyclopedia of decision making and decision support technologies* (pp. 661–667). Hershey: IGI Global. Retrieved from http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-59904-843-7.ch074 doi: 10.4018/978-1-59904-843-7.ch074

Mossin, J. (1968). Optimal Multiperiod Portfolio Policies. *The Journal of Business*, *41*(2), 215–229.

Mozer, M. C. (1989). A Focused Backpropagation Algorithm for Temporal Pattern Recognition. *Complex Systems*, *3*, 349- 381.

Ng, A. (2019a). *Stanford University CS229: Lecture notes.* Retrieved 2019-09-08, from http://cs229.stanford.edu/notes/cs229-notes1.pdf ([online] Available at: http://cs229.stanford.edu/notes/cs229-notes1.pdf [Accessed 2019-09-08])

Ng, A. (2019b). *Stanford University CS229: Splitting into train, dev and test sets.* Retrieved 2019-09-08, from https://cs230-stanford.github.io/train-dev-test-split.html ([online] Available at: https://cs230-stanford.github.io/train-dev-test-split.html [Accessed 2019-09-08])

Nickolls, J., Buck, I., Garland, M., & Skadron, K. (2008, 3). Scalable parallel programming with CUDA. *Queue - GPU Computing*, *6*(2), 40–53. Retrieved from http://portal.acm.org/citation.cfm?doid=1365490.1365500 doi: 10.1145/1365490.1365500

OpenAI. (2018). *OpenAI Five.* Retrieved from https://blog.openai.com/openai-five/ ([online] Available at: https://blog.openai.com/openai-five/ [Accessed 2019-09-08])

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. ([online] Available at: https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf [Accessed 2019-09-08])

Robinson, A. J., & Fallside, F. (1987). *The utility driven dynamic error propagation network*. University of Cambridge, Department of Engineering Cambridge.

Ross, S. A. (1977, 3). The Capital Asset Pricing Model (CAPM), Short-Sale Restrictions and Related Issues. *The Journal of Finance*, *32*(1), 177. Retrieved from https://www.jstor.org/stable/2326912?origin=crossref doi: 10.2307/2326912

Ruder, S. (2016, 9). An overview of gradient descent optimization algorithms. (arXiv:1609.04747v2 [cs.LG]). Retrieved from http://arxiv.org/abs/1609.04747

Samuelson, P. A. (1969, 8). Lifetime Portfolio Selection By Dynamic Stochastic Programming. *The Review of Economics and Statistics*, *51*(3), 239. Retrieved from https://www.jstor.org/stable/1926559?origin=crossref doi: 10.2307/1926559

Schmidhuber, J. (2015, 1). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117. Retrieved from https://linkinghub.elsevier.com/retrieve/pii/S0893608014002135 doi: 10.1016/j.neunet.2014.09.003

Severini, T. A. (2017). *Introduction to statistical methods for financial models*. Boca Raton: CRC Press, Taylor & Francis Group.

Sharpe, W. F. (1966, 1). Mutual Fund Performance. *The Journal of Business*, *39*(1), 119. Retrieved from https://www.jstor.org/stable/2351741 doi: 10.1086/294846

Sharpe, W. F. (1994, 10). The Sharpe Ratio. *The Journal of Portfolio Management*, *21*(1), 49–58. Retrieved from http://jpm.pm-research.com/lookup/doi/10.3905/jpm.1994.409501 doi: 10.3905/jpm.1994.409501

Smith, K. V. (1967, 9). A Transition Model for Portfolio Revision. *The Journal of Finance*, *22*(3), 425. Retrieved from https://www.jstor.org/stable/2978895?origin=crossref doi: 10.2307/2978895

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958. Retrieved from http://jmlr.org/papers/v15/srivastava14a.html

Sun, C., Shrivastava, A., Singh, S., & Gupta, A. (2017, 7). Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. (arXiv:1707.02968v2 [cs.CV]). Retrieved from http://arxiv.org/abs/1707.02968

Swinkels, L. (2004, 8). Momentum investing: A survey. *Journal of Asset Management*, *5*(2), 120–143. Retrieved from http://link.springer.com/10.1057/palgrave.jam.2240133 doi: 10.1057/palgrave.jam.2240133

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013, 12). Intriguing properties of neural networks. (arXiv:1312.6199v4 [cs.CV]). Retrieved from http://arxiv.org/abs/1312.6199

Theodoridis, S., & Koutroumbas, K. (2009). *Pattern recognition*. Burlington: Academic Press.

Tino, P., Schittenkopf, C., & Dorffner, G. (2001, 7). Financial volatility trading using recurrent neural networks. *IEEE Transactions on Neural Networks*, *12*(4), 865–874. Retrieved from http://ieeexplore.ieee.org/document/935096/ doi: 10.1109/72.935096

Wakker, P. P. (2008, 12). Explaining the characteristics of the power (CRRA) utility family. *Health Economics*, *17*(12), 1329–1344. Retrieved from http://doi.wiley.com/10.1002/hec.1331 doi: 10.1002/hec.1331

Walter, C. A., & Lopez, J. A. (2000, 2). Is Implied Correlation Worth Calculating? *The Journal of Derivatives*, *7*(3), 65–81. Retrieved from http://jod.pm-research.com/lookup/doi/10.3905/jod.2000.319125 doi: 10.3905/jod.2000.319125

Werbos, P. J. (1988, 1). Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, *1*(4), 339–356. Retrieved from https://linkinghub.elsevier.com/retrieve/pii/089360808890007X doi: 10.1016/0893-6080(88)90007-X

Würtz, D., Ellis, A., Chalabi, Y., Packages, R., Chalabi, Y., Chen, W., & Ellis, A. (2009). *Portfolio optimization with r/rmetrics rmetrics, association & finance online.* ([online] Available at: https://www.rmetrics.org/downloads/9783906041018-fPortfolio.pdf [Accessed 2019-09-08])