university of groningen / faculty of economics and business

A comparative analysis of neural networks in financial return forecasting

Master Thesis

Name: Max van de Ven
Student number: 2964449

Supervisor: Walter Kohl

9th of January 2020

University of Groningen

Faculty of Economics and Business

Nettelbosje 2, Groningen

The Netherlands

Abstract

This research is a comparative analysis of neural networks in financial return forecastability. Linear regression models are widely used to forecast returns. These models, however, often fail to capture abstract and non-linear relationships inherent financial time series. Artificial neural networks (ANN) seem to be particularly useful to bridge this gap. Inspired by a simplification of neurons in a brain, ANNs are non-linear statistical models that can theoretically approximate any continuous function. In this study, we compare the performance of the feed-forward neural network (FNN) as well as two types of recurrent neural networks (RNN), among which the gated recurrent unit (GRU) and the long short term memory (LSTM). Each network is trained on an asset-by-asset basis using an extensive dataset consisting of macro-economic and market data. By implementing a rolling-window scheme, we capture periodically changing predictor relationships and repeatedly forecast the returns one day ahead. The performance of each network is evaluated using the mean absolute scaled error (MASE), directional accuracy, pairwise Diebold-Mariano tests as well as a long-short risk parity portfolio constructed using the acquired forecasts. The results demonstrate that the RNNs tend to outperform FNNs. The magnitude of out performance, however, is asset-dependent. Additionally, the depth of each network type, nor the unit architecture of RNNs, tend not to alter the forecasting performance drastically. Finally, the network-based risk-parity portfolios demonstrate strong economic benefits, but raises questions for future research.

Table of content

Chapter 1

Introduction

The popularity of artificial neural networks is ever increasing because of their ability to effectively model data with very limited assumptions on the underlying distribution. The financial market is a domain that is immensely hard to model, in particular using classical statistical models. These traditional models often require an educated guess on the underlying function, before using the data to estimate its parameters. In contrast, artificial neural networks use their data directly to model the underlying function. This allows networks to capture complex and abstract relations which traditional models fail to find, giving it the theoretical underpinning as 'universal function approximator' (Hornik et al., 1989; Cybenko, 1989). For this reason, artificial neural networks are widely used in a variety of applications, such as prediction and forecasting.

The main objective of this research is to investigate the legitimacy of the efficient market hypothesis (or, abbreviated, "EMH") by means of artificial neural networks. In particular, we try to determine which network type and architecture is best equipped to exploit possible inefficiencies. We select a set of candidate neural networks that are potentially well suited to forecast asset returns. These include the feed-forward neural network and various recurrent networks, among which the gated recurrent unit (GRU) and long short term memory (LTSM). Taken in mind that highly noisy data from financial sources makes it hard to perform reliable experiments, we consider different model architectures and aim to fully optimize them before comparison. The forecasting power of each model will be compared using various metrics, as well as their performance on a long-short portfolio, which will be optimized using the acquired forecasts.

It should be noted that this is not an exhaustive analysis of all methods. For example, we exclude sparsely connected network architectures because of their arbitrarily chosen design. Nevertheless, our research is designed to be representative of predictive possibilities of the conventional toolkit of artificial neural networks.

1.1 Research questions

In particular, we will limit our research to the following set of questions:

1. Given a particular feature setup, are certain neural network types better equipped to predict future returns than others?
2. How does the depth of the various neural networks affect its predictive power?
3. Is it possible to build a successful trading strategy based on the predictions of each network?

The research will therefore provide us with interdisciplinary results covering modern machine learning and traditional finance empirics, such as portfolio theory and the efficient market hypothesis.

1.2 Related work

Various comparative work has been done on the intersection between machine learning and return forecastability. Gu, Kelly and Xiu (2018) investigate and compare the performance of a wide variety of machine learning methods in forecasting returns primarily using company-level characteristics. Their comparative analysis shows that the return forecasting ability of feed-forward neural networks outperforms classical linear models as well as other machine learning methods, such as random forests and generalized linear models. This result has been attributed to the ability of feed-forward neural networks to capture complex interactions among predictor variables, which other methods failed to identify. Abe and Nakayama (2018) validate these results and, additionally, find that deeper feed-forward networks tend to outperform their shallow counterpart. Our thesis serves as a direct extension on their work by investigating the forecasting abilities of different types of artificial neural networks, among which the recurrent neural network, with various levels of depth.

We distinguish ourselves by applying a computationally demanding rolling window scheme, where a window of t days is used to forecast the return of the $(t + 1)$st day. This allows us to accurately capture periodically changing predictor relationships. In addition, instead of grouping financial assets together like Gu et al. (2018), we train networks and generate forecasts for each financial asset separately. This allows us to distinguish between the heterogeneous predictor relationships of each financial asset under consideration. Furthermore, instead of company-specific data, this research uses an extensive predictor set consisting of both macroeconomic data, such as consumer confidence and inflation, and market data such as yield curves, foreign exchange rates and momentum. Lastly, we apply a distinct long-short portfolio approach to access the comprehensive performance of each network.

1.3 Outline

This thesis is organized as follows. In Chapter 2, we discuss the empirical models, which form the basis of our comparative analysis, and the main architecture of the networks. In Chapter 3, we discuss the data and the methodology for training and selecting the various networks. In addition, we introduce the metrics used to evaluate the forecasting performance of each network. In Chapter 4, the results are presented, followed by an analysis. Finally, we conclude and discuss our limitations in Chapter 5.

Chapter 2

Empirical models

In this section, we elaborate on the neural networks employed in this study. First, we will explain the functionality of the feed-forward neural network. Then, we will discuss recurrent neural networks, including the simple recurrent neural network, the long short term memory and gated rectified unit. Lastly, we elaborate on the network selection and architecture.

2.1 Feed-forward neural network

Artificial neural networks (ANNs) are mathematical simplifications of our central nervous system. These networks consist of interconnected groups of artificial neurons (or, synonymously, "nodes"). In feed-forward neural networks, these nodes are organized in the form of layers. The information moves progressively from the input layer, through the hidden layer (if any), to the output layer.

Figure 1: Feed-forward network without hidden layer

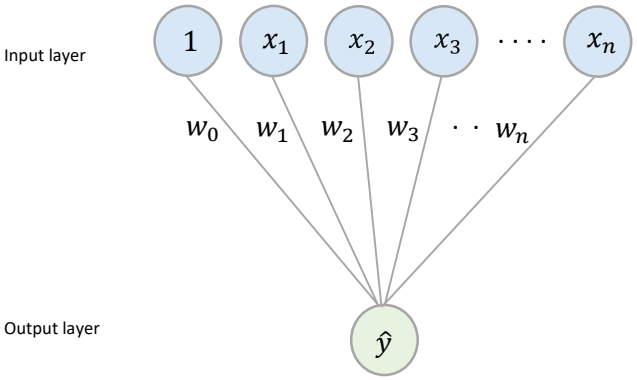


Fig. 1 displays a feed-forward network without a hidden layer. The input layer is a vector defined as $X = [x_0, x_1, \dots, x_n]'$. The first element of this vector x_0 is the bias node which is conveniently set to one. The other elements x_j for $j \in \{1,2, \dots, n\}$ are predictor values. Each line connecting one of the input units to an output node is assigned a weight, given by the weight vector $\omega = [\mathrm{w}_0, \mathrm{w}_1, \dots, \mathrm{w}_n]'$. These weights amplify or attenuate the input signals. The weighted signal (or, synonymously, "activation value") is therefore calculated as follows:

$$A = \sum_{i=0}^{n} x_i w_i \iff \sum_{i=1}^{n} x_i w_i + w_0 \iff \omega \cdot X \tag{2.1}$$

where $x_0 = 1$ and, consequently, w_0 is the bias term (or, synonymously, “intercept"). This bias term is analogous to the off-set of an ordinary least squares regression and allows for the shifting of the activation function.

Based on the weighted signal of equation (2.1), the activation function determines whether or not the output node is being activated. This is expressed as follows:

$$\hat{y} = \varphi(A) \iff \varphi(\omega \cdot X) \tag{2.2}$$

where $\varphi(\cdot)$ is an activation function. Note that, in case of a linear (or, synonymously, “identity”) activation, the simple neural network results in a linear regression model: $\hat{y} = \sum_{i=1}^{n} w_i x_i + w_0$. Usually, however, non-linear activation functions are used to generate non-linear mappings from inputs to output.

Figure 2: Feed-forward network with one hidden layer

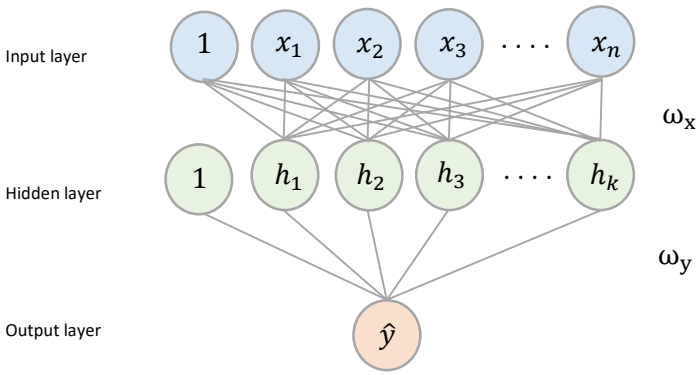


Fig. 2 displays a feed-forward neural network with one hidden layer. Adding hidden layers typically allows the model to capture more abstract and complex relationships between inputs and outputs. Similar to equation (2.2), the nodes in the hidden layer linearly extract information from the input layer. Then, each node applies an activation function to its weighted input. Finally, the activations are aggregated into an ultimate forecast output, as such:

$$\hat{y} = \varphi\left(\sum_{j=0}^{k} w_{j,y}\, h_j\right) \;\Leftrightarrow\; \varphi\left(\omega_{\mathrm{y}} \cdot \mathrm{H}\right) \tag{2.3}$$

where

$$\mathrm{h_j} = \varphi\left(\sum_{i=0}^{n} w_{ij,x}\, x_i\right), \qquad H = \varphi(\omega_{\mathrm{x}} \cdot \mathrm{X}) \tag{2.4}$$

The appropriate weight parameters are set by means of a *learning algorithm*. This also holds for the to-be discussed networks. We will come back to this in Section 2.5 and 2.6.

2.2 Recurrent neural network

Usually lagged predictor variables are included in feed-forward neural networks to capture time dependencies inherent financial data. Unfortunately, the appropriate number of lags is typically unknown. Consequently, a lagged dependent structure in a feed-forward network might be unable to accurately explain the behaviour of the predicted variable. To circumvent this problem, researchers have come up with various networks that have internal feedback loops. These are called recurrent neural networks (RNN) and are based on the seminal work by Rumelhart, Hinton and William (1986). RNNs can be thought of as a sequence of feed forward networks with the property that output at one time step serves as a supplementary input the next time step.

2.2.1 Simple recurrent neural network

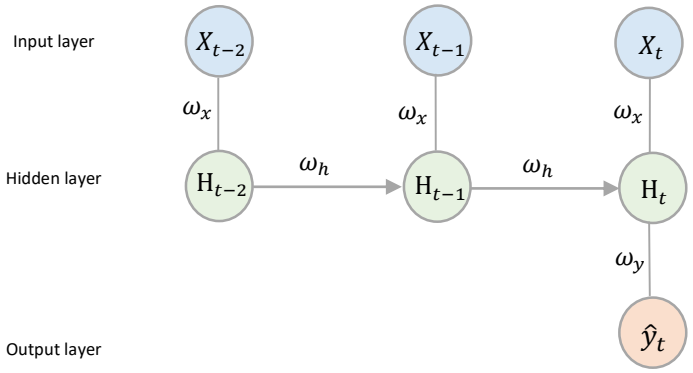
Simple RNNs model time dependencies (or, synonymously, "memory") by maintaining a hidden state. This is displayed in *Fig.* 3. The state of the hidden layer at the current time instant H_t uses the hidden state at the previous time instant H_{t-1} as a supplementary input. This enables the model to capture potential temporal relationships. Mathematically, this is expressed as follows:

$$\mathrm{H}_t = \varphi(\omega_h \cdot \mathrm{H}_{t-1} + \omega_x \cdot \mathrm{X_t}) \tag{2.5}$$

where $\mathrm{H_t}$ and $\mathrm{H_{t-1}}$ are vectors of the current and previous state of the hidden layer, X_t is the current input vector and, similar to the feed-forward network, the weight matrices ω_x and ω_h control the degree of significance to accord to both the present input and the past hidden state. Ultimately, the output forecast can be calculated as:

$$\hat{y}_t = \varphi(\omega_y \cdot \mathrm{H}_t) \tag{2.6}$$

Figure 3: Simple RNN with one hidden layer



The simple RNN is plagued with a couple of problems. Firstly, the performance of the simple RNN is strongly tied to the length of the sequence. The selected amount of time steps determine how far back in time the model goes to find time dependencies. For instance, the simple RNN displayed in *Fig*. 3 has three time steps. Once the time steps become too long, however, errors further back in time pass through an elongated chain of derivatives when applying the *learning algorithm* (*see* Section 2.6; Appendix 4). This so-called vanishing gradient problem causes long-term dependencies to be hidden by the effect of short-term dependencies. Secondly, it is worth noting that simple RNNs gradually accumulate information over time. This, however, is sub-optimal in the domain of financial time-series analysis which deals with highly non-stationary data. Alternately, we would like to forget periods where the distribution of the data strongly deviates from the current distribution. To achieve this, and circumvent the long-time dependency problem, this research focusses on a couple of modified RNNs, including the l*ong short term memory* and *gated recurrent units*.

2.2.2 Long Short Term Memory

The long short term memory (LSTM) is a recurrent neural network introduced by Hochreiter and Schmidhuber (1997). LTSM networks consists of a memory cell, called the carry state, which flows through the sequence of networks. The carry state is carefully regulated by structures called gates, which control the extent to which information flows in and out of the carry state. In contrast to simple RNNs, this carry dataflow allows the network to capture autoregressive structures of arbitrary lengths.

Similar to the simple RNN, the input at the current time step, X_t, and the hidden state of the previous time step, H_{t-1}, are fed into the LTSM. Unlike regular nodes, a LTSM unit consist of the following system of equations:

$$\begin{cases} f_t = \sigma_g(\omega_{f,h} \cdot H_{t-1} + \omega_{f,x} \cdot X_t) & (2.7) \\ i_t = \sigma_g(\omega_{i,h} \cdot H_{t-1} + \omega_{i,x} \cdot X_t) & (2.8) \\ o_t = \sigma_g(\omega_{o,h} \cdot H_{t-1} + \omega_{o,x} \cdot X_t) & (2.9) \\ \tilde{C}_t = \varphi\left(\omega_{c,h} \cdot H_{t-1} + \omega_{c,x} \cdot X_t\right) & (2.10) \\ C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t & (2.11) \\ H_t = o_t \circ \varphi(C_t) & (2.12) \end{cases}$$

where f_t, i_t and o_t are the forget, input and output gate, respectively, $\tilde{C}_t$ is the candidate memory, C_t is the carry state, H_t the hidden state at the current time instant, σ_g is the sigmoid activation function and the operator $\circ$ denotes the entry wise product (or, synonymously, "Hadamard product").[1] Note that the gates and candidate memory are constructed by feeding the input through a distinct fully connected layer before applying an activation function.

Taken in mind that $\sigma_g(x) \in [0,1]$, the gates modulate the flow of information inside the unit by means of three entry wise products. Firstly, the entry wise product of the previous carry state C_{t-1} with the forget gate f_t in equation (2.11) is a way to remove irrelevant information from the carry dataflow. Secondly, the entry wise product of the input gate i_t with the vector of candidate values $\tilde{C}_t$ in equation (2.11) updates the carry dataflow with new information. Finally, the entry wise product of the output gate o_t with the activated new carry state $\varphi(C_t)$ in equation (2.12) regulates what part of the carry dataflow is used to create the current hidden state, H_t.

[1] The sigmoid activation function is defined as $\sigma_g(x) = \frac{1}{1+e^{-x}}$ which squishes the elements of the gate vectors between 0 ("irrelevant") and 1 ("relevant").

2.2.3 Gated Recurrent Units

Cho, et al. (2014) developed an alternative to the LTSM called the gated rectified unit (GRU). Similar to LSTM units, the GRU has gates that regulate the flow of information inside the unit, however, without having a distinct memory cell. The GRU unit consists of the following system of equations:

$$\begin{cases} z_t = \sigma_g\left(\omega_{z,h} \cdot H_{t-1} + \omega_{z,x} \cdot X_t\right) & (2.13) \\ r_t = \sigma_g\left(\omega_{r,h} \cdot H_{t-1} + \omega_{r,x} \cdot X_t\right) & (2.14) \\ \widetilde{H}_t = \varphi(\omega_{h,h} \cdot (H_{t-1} \circ \mathrm{r_t}) + \omega_{h,x} \cdot X_t) & (2.15) \\ H_t = z_t \circ \mathrm{H_{t-1}} + (1 - z_t) \circ \widetilde{H}_t & (2.16) \end{cases}$$

where z_t and r_t are the update and reset gate, respectively, and $\widetilde{H}_t$ is the candidate hidden state. Again, the gates and candidate hidden state are constructed by feeding the input through a distinct fully connected layer before applying an activation function.

Similar to the LTSM unit, the gates regulate the flow of temporal information by means of entry wise products. Firstly, the candidate hidden state $\widetilde{H}_t$ of equation (2.15) is shaped by the entry wise product of the reset gate r_t with the previous hidden state H_{t-1}. Whenever the elements of the reset vector r_t are close to one, we retrieve the conventional hidden state of equation (2.5). However, for elements of r_t that are close to zero, the pre-existing memory 'resets'. This effect therefore resembles the forget gate of the LTSM unit. Secondly, the hidden state H_t of equation (2.16) is a linear interpolation of the prior hidden state $\mathrm{H_{t-1}}$ and candidate hidden state $\widetilde{H}_t$, controlled by the update gate z_t. Whenever an element of the update vector z_t equals one, the prior hidden state is preserved while effectually disregarding any new information of the current time step. On the other hand, for elements of the update vector close to zero, the hidden state is updated with new information from the candidate state $\widetilde{H}_t$. The update gate can therefore be regarded as a combination of the input gate and forget gate of the LTSM unit.

There are a few advantages of using GRU units over to LSTM units. Firstly, GRU units regulate the intertemporal information transfer utilizing only two gates - instead of three gates in LSTM units. This reduces training time as there are fewer to be estimated parameters. Secondly, GRU units have shown to exhibit better performance than LTSM units in certain applications outside of the financial realm (Chung et al., 2014).[2]

[2] It is worth noting, however, that the literature has not reached a consens on this matter. For instance, Greff et al. (2017) demonstrate that none of the RNN variants improve upon the standard LSTM units significantly.

2.3 Neurons and layers

The amount of neurons in each layer, the number of hidden layers and the connectedness of each layer are essential parts of the overall architecture and strongly tied to the performance of the network.

Firstly, the number of neurons in each hidden layer is closely related to under and overfitting (Heaton, 2015). Overfitting occurs when the neural network has too many degrees of freedom, allowing the model to capture relationship based on the noise or randomness inherent the training data. Similarly, underfitting occurs when there are too few neurons in the hidden layers to adequately capture abstract and complex relationships in a data set. Both scenarios generally result in poor generalization performance of the model. For this reason being, we run a search algorithm to find the optimal number of neurons for the first hidden layer (*see* Section 2.9). To economize on computational cost, the amount of neurons in hidden layers thereafter (if any) are determined by a popular rule-of-thumb called the geometric pyramid rule (Masters, 1993).

Moreover, the number of hidden layers in the network affects its ability to capture complex and abstract relationships. It has been shown that deeper networks can achieve the same performance with considerably fewer parameters (e.g. Cohen et al., 2015; Eldan and Shamir, 2016). However, training a deeper neural network is challenging for a few reasons. Firstly, deeper networks overfit with diminished effort because of their ability to capture abstract relationships with more ease. Secondly, the learning algorithm of deeper networks requires the multiplication of an elongated chain of derivatives, potentially resulting in exploding or vanishing gradients (*see* Appendix 4). Finally, the multidimensional error plane becomes increasingly non-convex as network depth increases (*see* Section 2.6). Hence, to infer the trade-off of network depth in the return forecasting problem, we compare the performance of several network depths.

Lastly, it is worth noting that all architectures are fully connected, implying that each node receives inputs from all nodes in the earlier layer.

2.4 Activation function

Based on the weighted signal, the activation function determines whether or not the output node is being activated. As a neural network without an activation function can be regarded as a linear regression model, the purpose of the activation function is to introduce non-linearity. The most popular activation in the literature is known as the rectified linear unit developed by Hahnloser et al. (2000), calculated as:

$$Relu\ (X) = \max\ (0, X) = \begin{cases} X & if\ X > 0 \\ 0 & else \end{cases} \tag{2.17}$$

This activation function tends to enhance the performance of many deep networks, both in the sense of computational efficiency and accuracy (e.g. Hahnloser et al., 2000; Jarrett et al., 2009; Nair and Hinton, 2010; Glorot et al., 2011).[3] We adopt the same activation function across all hidden nodes, unless explicitly stated otherwise in earlier sections.

2.5 Objective function

To evaluate the performance of the networks, we introduce an error function (or, synonymously, "objective function"). The traditional error function for regression problems is the mean-squared error. The mean-squared error measures the average of the squared difference between the known return $\gamma_{i,t+1}$ and the forecasted return $\hat{\gamma}_{i,t+1}(\omega)$ of financial asset i at time t. It is defined as:

$$\mathcal{L}(\omega) = \frac{1}{T}\sum_{t=1}^{T}\left(\hat{\gamma}_{i,t+1}(\omega) - \gamma_{i,t+1}\right)^2 \tag{2.18}$$

2.6 Parameter optimization

2.6.1 Gradient descent

The main problem is to find the weight parameters of the neural network ω that minimize the value of the error function $\mathcal{L}(\omega)$. This can be achieved by applying optimization theory. Given that $\mathcal{L}(\omega)$ is a differentiable and continuous function of ω, we can calculate the gradient $\nabla\mathcal{L}(\omega)$ as:

$$\nabla\mathcal{L}(\omega) = \left[\frac{\partial\mathcal{L}(\omega)}{\partial w_1}, \dots, \frac{\partial\mathcal{L}(\omega)}{\partial w_j}\right]' \tag{2.19}$$

[3] The low computational cost stems from the fact that the activation function is easily differentiable, which reduces the time it takes to recursively approximate the gradient (*see* Section 2.6; Appendix 4).

were $\partial\mathcal{L}(\omega)/\partial w_j$ is the partial derivative of $\mathcal{L}(\omega)$ w.r.t. the j-th weight parameter. The gradient is evaluated using an algorithm called back propagation (*see* Appendix 4).

The gradient $\nabla\mathcal{L}(\omega)$ is a vector that points in the direction of the greatest ascent. We can therefore minimize the error function by repeatedly taking steps in the opposite direction of the gradient. This algorithm is called *gradient descent* and is expressed as follows:

$$\omega_{\tau+1} = \omega_\tau - \eta\nabla\mathcal{L}(\omega_\tau) \tag{2.20}$$

where τ is the iteration and η is a hyper parameter known as the learning rate that determines the velocity of the descent. After an adequate amount of iterations, the algorithm will reach a point ω_τ where $\nabla\mathcal{L}(\omega_\tau) = 0$.[4] This is called a stationary point, representing a local extreme or saddle point. Generally, loss functions of neural networks are highly non-convex with multiple local extrema and saddle points.

2.6.2 Stochastic gradient descent

Two problems rise to the surface when applying true gradient descent. Firstly, the error function $\mathcal{L}(\omega)$ and the associated gradient $\nabla\mathcal{L}(\omega)$ are calculated with respect to the entire training dataset $\mathcal{D}$ at every iteration. This is incredibly time consuming when working with large datasets. Secondly, local minima are rare in high dimensional non-convex error surfaces while saddle points generically proliferate (Dauphin et al., 2014). That is, as dimensionality increases, the chance that all the directions around a stationary point have a positive curvature decreases exponentially. This therefore requires an algorithm that abstains from stranding in saddle points.

To economize on the computational cost and address saddle points, we implement a modification of the algorithm called *stochastic gradient descent* (Bottou et al., 1998). As displayed in *Fig* 4, stochastic gradient descent uses a random subset of the training dataset (or, synonymously, "batch") to approximate the true gradient $\nabla\mathcal{L}(\omega)$ at every iteration. This is expressed as follows:

$$\omega_{\tau+1} = \omega_\tau - \eta\nabla\mathcal{L}(\omega_\tau, b_\tau) \tag{2.21}$$

where $\nabla\mathcal{L}(\omega_\tau, b_\tau)$ is the gradient of the loss function evaluated on batch b_τ from the training set $\mathcal{D}$.[5]

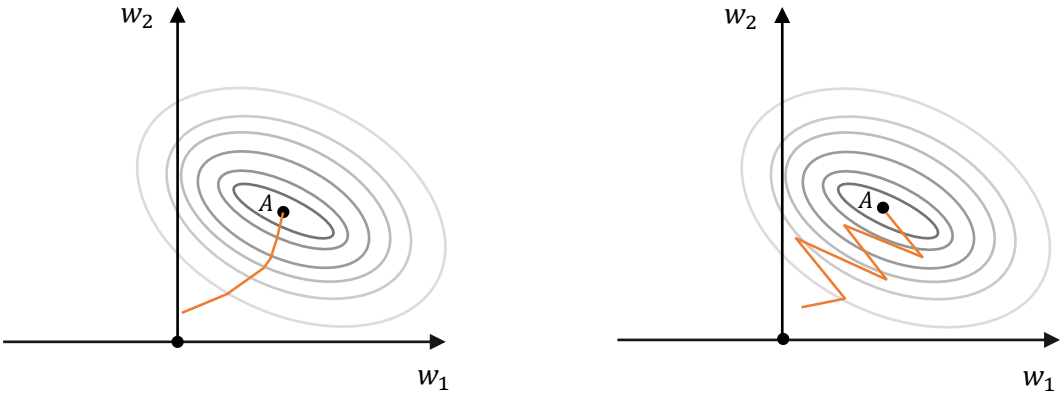
[4] Ignoring the effects of the learning rate for the time being.

[5] The temporal order of the financial time series is pivotal as the next data point could could be influenced by the previous one. Thus, even though the batch is randomly selected at each iteration, we do not allow for

If the batch-size $\mathcal{B}$ is equal to 1, the true gradient is approximated by single training point at each iteration. This might lead to very noisy gradients which are poor approximations of the true gradient – and lead us to the wrong direction. Meanwhile, if $\mathcal{B}$ consist of the entire dataset, we are back at square one. Hence, a compromise is reached when the batch size is in the midst of these extremes. Here, the approximation sacrifices accuracy for massive acceleration of the recursive process. While, at the same time, reasonable levels of noise in the sample gradient enables it to escape saddle points (Dauphin et al., 2014; Ge et al., 2015).

Both the learning rate η and the batch-size $\mathcal{B}$ are hyper parameters which are optimized using a search algorithm (*see* Section 2.9).

Figure 4: Comparison of true- and stochastic gradient descent



Note: this figure shows a simplified example of true (left) and stochastic (right) gradient descent. Shown are the contours of a loss function $\mathcal{L}(\omega)$ with $\omega = [w_1, w_2]'$, which is minimized at point A. The noise of the gradient approximation causes the trend to alter from the direction of the greatest ascent at each iteration – allowing it to escape saddle points.

2.7 Optimizers

The performance of the gradient descent algorithm strongly depends on the decided upon learning rate. If the size of the learning rate is too small, convergence is very time-consuming. On the other hand, if the size of the learning rate is too large, the algorithm might overshoot and oscillate around the minima. Thus, a corrective measure is to fine-tune the learning rate at various epochs of the algorithm.

shuffling. Taken in mind that our predictor set consists of, among others, quarterly released predictor values, this requires a batch-size which is sufficiently large to capture signals.

2.7.1 AdaGrad

The adaptive gradient algorithm (AdaGrad) developed by Duchi, Hazan and Singer (2011) is a mutation of the stochastic gradient descent with a parameter-specific learning rate. The per-parameter update of the AdaGrad algorithm is as follows:

$$\omega_{\tau+1,j} = \omega_{\tau,j} - \eta \frac{1}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,j}^2}} g_{\tau,j} \tag{2.22}$$

where $g_\tau = \nabla \mathcal{L}(\omega_\tau)$, the gradient, and $g_{\tau,j} = \partial \mathcal{L}(\omega_\tau)/\partial w_j$, the partial derivative, at iteration τ. The denominator accumulates the squared value (or, synonymously, "$\ell 2$ norm") of previous derivatives over all elapsed iterations, t. The learning rate η is then scaled by the inverse of this magnitude. Consequently, parameters that experience small (large) updates receive larger (smaller) learning rates. In addition, the accumulation of derivatives in the denominator reduces the learning rate over time (Zeiler, 2012). Both effects tend to improve convergence performance over standard stochastic gradient descent. Particularly in situations where data is sparse and sparse parameters have more explanatory power (Gupta, Bengio and Weston, 2014).

2.7.2 RMSProp

In the AdaGrad method the denominator accumulates the squared gradients from all elapsed iterations, causing it to grow through-out the entire training period. After many iterations this results in an infinitesimal learning rate - potentially before actually reaching a minima. The RMSprop optimizer is a method that aims to curtail this expeditious, consistently decreasing learning rate (Tieleman and Hinton, 2012). Instead of accumulating all past squared gradients, the accumulation in RMSprop is an exponentially decaying average of the squared gradients. This allows learning to continue even after many iterations have elapsed. The average depends on the current gradient and the previous average:

$$\mathbb{E}[g^2]_{\tau,j} = \lambda\, \mathbb{E}[g^2]_{\tau-1,j} + (1-\lambda)\, g_{\tau,j}^2 \tag{2.23}$$

where λ is the decay constant, which is set equal to its default value $\lambda = 0.9$. This parameter determines the length of the gradient window that truly influences the learning rate. Similar to equation (2.22), the per-parameter updated is then calculated as follows:

$$\omega_{\tau+1,\mathrm{j}} = \omega_{\tau,j} - \eta \frac{1}{\sqrt{\mathbb{E}[g^2]_{t,j}}} g_{\tau,j} \tag{2.24}$$

2.7.3 ADAM

The most prominently used optimizer is the adaptive moment estimation (ADAM) developed by Kingma and Ba (2014). The ADAM optimizer keeps track of running averages of both the first and second order moment of the gradients, as such:

$$\mathbb{E}[g]_{\tau,j} = \lambda_1 \, \mathbb{E}[g]_{\tau-1,j} + (1 - \lambda_1) \, g_{\tau,j} \tag{2.25}$$

$$\mathbb{E}[g^2]_{\tau,j} = \lambda_2 \, \mathbb{E}[g^2]_{\tau-1,j} + (1 - \lambda_2) \, g_{\tau,j}^2 \tag{2.26}$$

Taken in mind that the initial values of $\mathbb{E}[g]_{\tau,j}$ and $\mathbb{E}[g^2]_{\tau,j}$ are zero, they will be biased towards zero when the decay constants are large. To correct for this, the mean and variance are calculated as follows:

$$\widehat{\mathbb{E}[g]_{\tau,J}} = \frac{\mathbb{E}[g]_{\tau,j}}{1 - \lambda_1} \tag{2.27}$$

$$\widehat{\mathbb{E}[g^2]_{\tau,J}} = \frac{\mathbb{E}[g^2]_{\tau,j}}{1 - \lambda_2} \tag{2.28}$$

Similar to equation (2.22) and (2.24), the per-parameter updated is then calculated as follows:

$$\omega_{\tau+1,j} = \omega_{\tau,j} - \eta \frac{1}{\sqrt{\widehat{\mathbb{E}[g^2]_{\tau,J}}}} \widehat{\mathbb{E}[g]_{\tau,J}} \tag{2.29}$$

Clearly, the learning rate is scaled by the inverse of the squared gradients just like RMSprop. Additionally, the optimizer takes advantage of momentum by using moving average of the gradient instead of gradient itself. Following common practice, the hyper parameters λ_1 and λ_2 are set to its default values of 0.9 and 0.99, respectively.

2.7.4 Remarks

It is not entirely clear whether widely-used adaptive methods like ADAM, Adagrad and RMSProp actually stimulate out-of-sample performance. Even though many adaptive optimizers tend to enhance training performance, Wilson et al. (2017) and Keskar et al. (2017) demonstrate that adaptive methods generalize significantly worse compared to vanilla stochastic gradient. Taken this into account, we implement a search algorithm to determine the optimal optimization algorithm (*see* Section 2.9).

2.8 Regularization

Training the model is equivalent to minimizing the loss function, which assures that the model fits the training set as well as possible. This, however, also allows the model over fit and learn the idiosyncrasies of the training set. As a consequence, the model might not generalize well. That is, it will perform poorly when we apply the model on data it has never seen before to generate forecasts. To prevent this from occurring, we put constraints on the complexity of the neural networks. These constraints aim at making the model generalize better.

2.8.1 Weight regularization

A typical way to mitigate over fitting is by forcing the weight parameters of the network to take on small values.[6] This is called weight regularization, and is accomplished by adding to the loss function a penalty term associated with having large weights:

$$\mathcal{L}_p(\omega, \lambda_1, \lambda_2) = \mathcal{L}(\omega) + \phi(\omega, \lambda_1, \lambda_2) \tag{2.30}$$

There are various choices for the penalty term $\phi(\omega, \lambda_1, \lambda_2)$. This paper employs the elastic net penalty, which takes the form:

$$\phi(\omega, \lambda_1, \lambda_2) = \lambda_1 \sum_{j=1}^{P} |\omega_j| + \lambda_2 \sum_{j=1}^{P} \omega_j^2 \tag{2.31}$$

The elastic net is composed of two popular regularizers whose penalty strength is determined by the two hyper parameters λ_1 and λ_2. The case were $\lambda_1 \neq 0$ and $\lambda_2 = 0$ corresponds to the *lasso regression* and uses the $\ell 1$-norm (or, synonymously, "absolute value penalty"). The $\ell 1$-norm shrinks the weights by a constant factor and forces a subset of weight parameters to exactly zero. In this sense, it compresses the network to a smaller number of high-importance connections. The case were $\lambda_1 = 0$ and $\lambda_2 \neq 0$ corresponds to *ridge regression* and uses the $\ell 2$-norm (or, synonymously, "squared value penalty"). The $\ell 2$-norm shrinks the weights by a factor proportional to ω_j, preventing weights from becoming disproportionately large in magnitude. However, it does not impose sparsity like the $\ell 1$-norm. If both $\lambda_1 \neq 0$ and $\lambda_2 \neq 0$, the elastic net favours simpler models through both shrinkage and sparsity. Both hyper parameters λ_1 and λ_2 are optimized using a search algorithm (*see* Section 2.9).

[6] In a nutshell, if weight parameters are smaller, the outcome of the neural network has more stability. In particular, small changes of the input will result in small changes of the output. This makes it harder for the network to learn the noise inherent the training dataset.

2.8.2 Ensemble training

We aim to improve the results of each neural network by adopting an ensemble method. In particular, we start the optimization algorithm at multiple random locations and construct an ultimate forecast by averaging forecasts over all randomly initialized networks.[7] Given the stochastic nature of the optimization process, different weight initializations tend to bring about different forecasts. By averaging forecasts over all randomly initiated networks, the ensemble approach reduces prediction variance. In turn, it may provide us with a better approximation of the true unknown function (Dietterich, 2000).

2.8.3 Early stopping

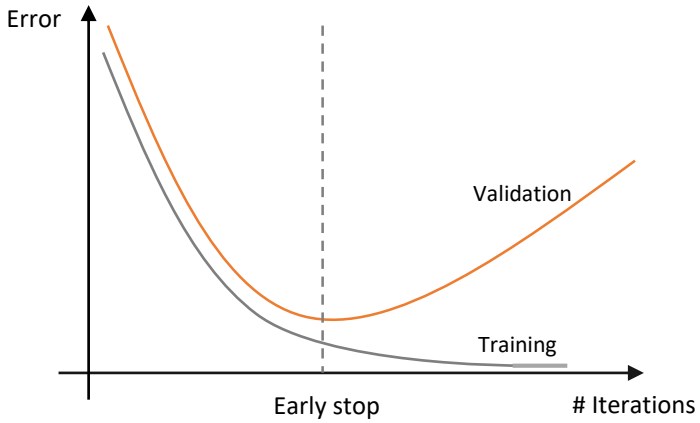
Early stopping is a regularization algorithm which prevents the network from over fitting. The weight parameters are iteratively adjusted with the ultimate objective of minimizing the loss function over the training sample. After each adjustment, the predictions of the network are evaluated over a fixed set of examples not from the training set - the validation set. The error on the validation set calculated using equation (2.18) serves as a proxy for the generalization error. This provides us with an indication of when over fitting has started. The optimization is discontinued when the validation errors starts to increase, as displayed in *Fig*. 5.

[7] To intiailize weights randomly, we use the He initializer (He et al., 2015). This is an extension of the Xavier initializer (Glorot and Bengio, 2010) particularly designed for ReLu activation. This assures faster convergence and prevents activations from vanishing or exploding. The He initializer draws sample weights from a truncated normal distribution:

$$N\left(0, \sqrt{\frac{2}{n^{(l-1)}}}\right)$$

were $n^{(l-1)}$ is the number of incoming network connections.

Figure 5: Early stopping



In practice, the validation error does not mature as smoothly. Instead, the validation error tends to evolve in a highly non-convex manner. Thus, to prevent the call back from stopping to early, we define a patience. This is an interval of iterations over which we allow the validation error to see no improvement, without triggering the early stopping mechanism.

2.9 Hyper parameter tuning

Hyper parameter tuning amounts to searching for optimal hyper parameters that tend to produce satisfactory out-of-sample forecasts. The literature typically employs Grid Search in order to do so. This is an exhaustive search through a manually specified subset of the hyper parameter, in which all hyper parameter combinations are evaluated on a held-out validation set. This algorithm, however, suffers from the curse of dimensionality making it computationally intensive. To enhance efficiency, we adopt the Random Search algorithm proposed by Bergstra and Bengio (2012). Instead of an exhaustive enumeration, this algorithm selects various random parameter combinations. The random combinations are drawn from pre-defined marginal distributions, which allows for the inclusion of prior knowledge. When not all parameters are equally relevant in optimization, it has been shown to outperform Grid Search using the same amount of resources (Bergstra and Bengio, 2012). The hyper parameter setting is summarized in Appendix 2.

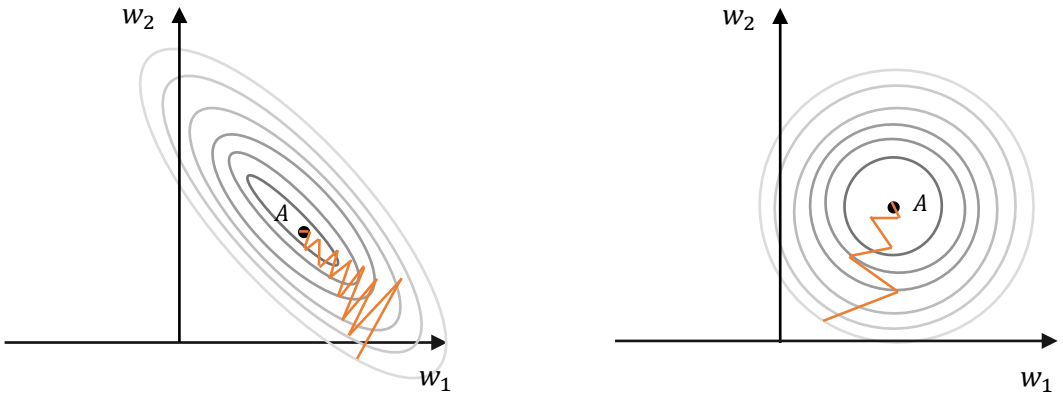
2.10 Scaling

A common data pre-processing step is feature scaling. Before feeding the input through the network, the range of values that each feature can attain is normalized. The main objective of rescaling is to enhance optimization. In particular, when employing gradient descent, the weight updates are dependent on the values of the features itself. As a result, certain weights update faster than others solely because of scaling differences. This might lead to oscillation around a minimum and elongation of the optimization procedure, as shown in *Fig*. 6. Feature scaling could therefore enhance convergence speed, particularly if the scale of feature values are dissimilar. The most widely implemented normalization method is called standardization. This method assures that the feature values have zero-mean and unit-variance, as such:

$$x_{j,t}{}' = \frac{x_{j,t} - \bar{x}_j}{\sigma_j} \tag{2.32}$$

Scaling by removing the mean is particularly interesting for our purpose, as it forces features to never be always positive, allowing us to remove any bias in the model.

Figure 6: Data normalization



Note: the figure shows non-normalized (left) and normalized (right) features. Shown are the stylized contours of the cost function $\mathcal{L}(\omega)$. By normalizing the features, the performance of the learning algorithm becomes less dependent on the weight initialization; enhancing learning.

2.10.1 Batch normalization

Developed by Sergey and Szegedy (2015), batch normalization is a method to stabilize the distributions of hidden layer inputs. This is achieved by introducing an additional network layer that cross-sectionally standardizes the activations values over each batch. Then, the standardized values are scaled and shifted based on trainable parameters to maintain modelling flexibility. This standardization is applied before the non-linearity of the preceding layer.

Santurkar et al. (2019) show that batch normalization causes a reparameterization of the underlying optimization. This has two main effects. Firstly, it increases the smoothness of the high dimensional non-convex error plane. This enhances the stability of gradient descent–based training algorithm as it becomes less prone to exploding or vanishing gradients. Secondly, it tends to make gradients more reliable and predictive. In particular, the computed gradient direction remains a fairly accurate estimate of the actual gradient direction, even after taking a larger step in that direction. Both effects make training significantly faster and less sensitive to the choice of hyper parameters. [8, 9]

[8] The findings are based on the improved Lipschitzness of the loss function and the gradient, respectively, after applying batch normalization.

[9] Santurkar et al. (2019) also show that the 'covariate shift' has no effect on performance by injecting noise in each layer after batch normalization. This invalidates Sergey and Szegedy's (2015) original motivation for batch normalization.

Chapter 3

Methodology

3.1 Data

We compare the performance and forecasting ability of three different types of networks with three different levels of depth, yielding nine different models. This is achieved using 5 financial assets $A_1, \dots, A_5$, as shown in Appendix 3. After selecting a specific model, we train network weights and generate forecasts for each financial asset separately. This allows us to distinguish between the heterogeneous predictor relationships of the financial assets under consideration.

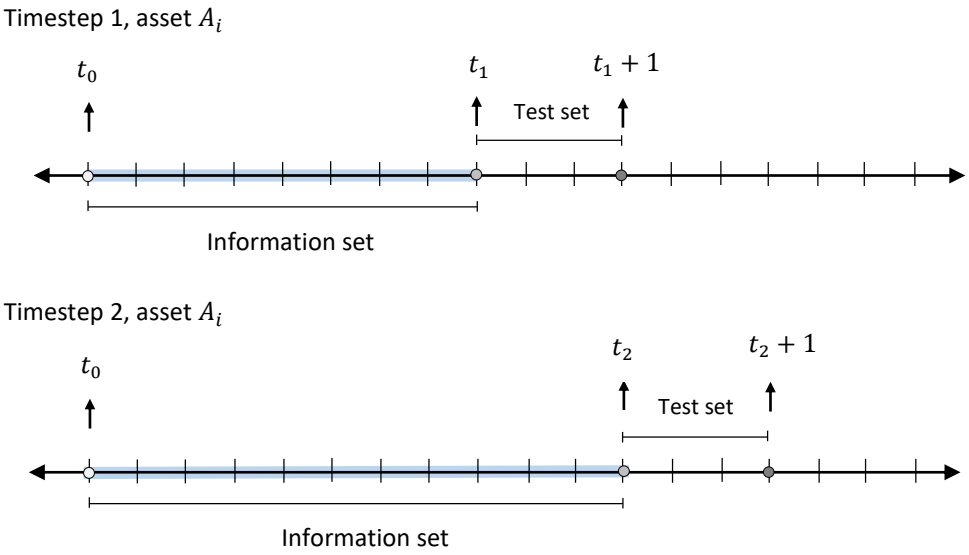
Our dataset reaches from April 1, 2009 to June 27, 2019, encompassing 2,578 business days for each financial asset under consideration. We select 44 predictor variables $x_{1,i,t}, \dots, x_{44,i,t}$ to forecast the return of asset A_i one day ahead, $y_{i,t+1}$. In particular, the predictor set consist of macroeconomic data such as output growth, money supply and inflation, and market data such as yield curves, foreign exchange rates and momentum. As shown in Appendix 1, these predictor variables have shown significant forecasting abilities in prior research. However, it is worth noting that return predictability of a variable tends to decline substantially after the publication of the predictor (McLean and Pontiff, 2016). To partially address this issue, our main focus lies on recently detected predictor variables. In addition, it should be pointed out that macro-economic data releases are often subject to revision. As a result, unsophisticated use of the data can lead to serious look-ahead bias. To prevent this from happening, point in time estimates are collected were possible. When unable to distinguish between original and revised data, we follow Gu et al. (2019) and delay the weekly characteristics by one week, monthly characteristics by one month and quarterly characteristics by three months. Another issue is missing data points, which we replace with the monthly cross-sectional median for each financial asset.

The implementation is done using Google's Tensorflow library, which is a highly scalable and flexible machine learning framework (*see* Abadi et al., 2015). The current surge in training deep neural networks for financial applications is, at least partly, motivated by recent important contributions to this open source community.

3.2 Rolling window scheme

We train the networks and generate the out-of-sample forecasts by implementing a rolling window scheme, as displayed in *Fig*. 7. The combination of the training and validation set is called the *information set*. The training set consist of the earliest 97 percent of the information set, while the validation set consist of the remaining 3 percent. For each time step, the information set consist of all the datapoints available up to that moment and the next data point is used as a test set. That is, we repeatedly try to predict one day ahead using a network shaped by all the information available to date. As a result, the information set increases by one data point after each time step. This procedure is repeated until we reach the end of the dataset.

Figure 7: Rolling window scheme



Taken into account the low signal-to-noise ratio of financial data, we need an adequate amount of training data to allow the network to capture signals. Therefore, the initial information set consists of 2,518 datapoints, approximately 9 business years. The length of the information set increases the chance of exposure to both up- and downturns of the market, potentially improving signal detection. Given the size of the original dataset, the rolling window scheme results in 60 partly-overlapping information sets and 60 non-overlapping test sets for each financial asset.

This particular rolling window scheme has been implemented for two reason. Firstly, the rolling window scheme allows us to periodically adapt the predictor weights. Taken in mind that the predicting power of signals change over time, this produces more robust forecasts. Secondly, it tends to replicate modelling behaviour of investors and portfolio managers alike.[10]

3.2.1 Training

The data that goes into the network are the standardized predictor variables $x_{1,i,t}, \dots, x_{44,i,t}$ over the training set $\mathcal{D}$, the optimal hyper parameters and the randomly initialized weights. Using these inputs, the network predicts future returns $\hat{y}_{t+1,i}$ over the training set, which are fed into the penalized loss function. The gradient of the loss function is then approximated through back propagation and the weights are adjusted according to the selected optimizer. After each weight adjustment, the network generates predictions on the held-out validation set. In accordance to the early stopping algorithm, training stops once the validation error starts to increase and the patience requirement is met. Once training stops, the fully trained network is used to generate an out-of-sample prediction $\hat{y}_{t+1,i}$ over the test set. Following the ensemble approach, these steps are repeated multiple times per timestep before averaging the results together into an ultimate forecast. Afterwards, the data point of the test set is added to the information set and the entire procedure repeated.

3.2.2 Hyper parameters

Before conducting the earlier discussed training procedure, we search for the optimal hyper parameter combination by means of cross-validation. The data that goes into the network are the predictor variables over the training set $\mathcal{D}$, randomly initialized weights and an initial guess for the hyper parameter choice. The network weights are then adjusted recursively in accordance to the learning algorithm discussed earlier. After each weight adjustment, the network also generates predictions over the held-out validation set. Training stops as soon as the early stopping mechanism is triggered. The lowest level of validation error and the corresponding hyper parameters are then stored. Next, the procedure is repeated with a new set of hyper parameters, drawn randomly from pre-defined marginal distributions (see Appendix 2). After evaluating a fixed amount of hyper parameter combinations (or, synonymously, “grid iterations”), we select the one that produces the lowest validation error.

[10] One potential bias stems from the fact that the training set inflates as the as the rolling period grows. In turn, forecasts made further into the future may be of higher quality than those at earlier rolling periods. However, since this bias is present in all model forecasts, we expect it not to influence the quality of the model comparison.

We expect the optimal hyper parameters choice to be rather stable between two successive time steps. For this reason, the hyper parameter are optimized only once every 10 timesteps. This allows us to increase the amount of grid iterations, and therefore the accuracy of the optimal parameter choice, at no computational cost.

3.3 Performance evaluation

3.3.1 Forecasting accuracy

As an informal measure to evaluate the predictive performance of networks for individual asset return forecasts, we calculate the mean absolute scaled error (MASE) as proposed by Hyndman and Koehler (2006). The MASE has advantageous properties compared to other measures, such as the root-mean-square deviation, for calculating forecasting accuracy (*see* Franses, 2016). These include, among others, scale invariance, symmetry and interpretability. It is calculated as follows:

$$MASE = \frac{\frac{1}{J} \cdot \sum_{t=1}^{J} \left| y_{i,t+1} - \hat{y}_{i,t+1} \right|}{\frac{1}{T-1} \cdot \sum_{t=2}^{T} \left| y_{i,t+1} - y_{i,t} \right|} \tag{3.1}$$

where the numerator is the forecast error over the out-of-sample period J, and the denominator is the mean absolute error of a naive random-walk without a drift over the information set T. That is, it measures the relative reduction in error compared to a naive one-step forecast method. When comparing forecasting methods, the method with the lowest MASE is the preferred model.

3.3.2 Directional accuracy

Predicting stock market returns accurately is incredibly challenging because of the high noise-to-signal ratio inherent financial data. It is generally presumed that predicting the sign is easier than predicting the actual return. Taken in mind that the direction of the market may already be sufficient to determine the investment position on an asset, we will pay close attention to the percentage of correct directional forecasts of each network.

3.3.3 Diebold-Mariano test

To make a statistical pairwise comparisons of the generated network forecasts, we implement the Diebold and Mariano (1995) test for differences in out-of-sample forecasting accuracy between two methods. In order to test the null hypothesis that method (1) and (2) have the same accuracy, Diebold-Mariano use the following test statistic:

$$DM = \frac{\bar{d}}{\sqrt{\frac{\left[y_0 + 2\sum_{k=1}^{h-1} y_k\right]}{n}}} \tag{3.3}$$

where $d_t = \left(e_t^{(1)}\right)^2 - \left(e_t^{(2)}\right)^2$ is called the loss-differential, $e_t^{(2)}$ and $e_t^{(1)}$ denote the prediction error of the return at time t for two different ANNs over the testing set, h is the forecast horizon and y_k is the autcovariance at lag k, defined as: [11]

$$y_k = \frac{1}{T}\sum_{t=k+1}^{T}(d_t - \bar{d})(d_{t-k} - \bar{d}) \tag{3.4}$$

The Diebold-Mariano test statistic requires only that the loss differential be covariance stationary (Diebold and Mariano, 1995). For this reason, we examine the sample autocorrelations of the loss differentials and test for unit roots through the augmented Dickey-Fuller test.

Harvey, Leybourne and Newbold (1997) find that the Diebold-Mariano test tends to reject the null hypothesis too often for small samples. They state that one can obtain improved properties for small sample testing by making a bias correction to the Diebold-Mariano test statistic, and comparing this corrected statistic with a student t-distribution, instead of the standard normal distribution. The bias corrected statistic is calculated as follows:

$$HLN = \sqrt{\frac{T + 1 - 2h + h(h-1)}{T}}\,DM \tag{3.5}$$

[11] The loss-differentials could be serially correlated because of a variety of reasons. The most obvious one being sub-optimal forecasting. To prevent this from hampering the test statistic, the standard error in the DM statistic is calculated robustly.

Taken in mind the relative small size of our out-of-sample window, the adjusted Diebold-Mariano test statistic provides us with more reliable p-values for model comparison.

3.3.4 Portfolio performance

Given that forecasts of each network are asset-specific, the out-of-sample performance of a portfolio could provide us with an additional evaluation of the total network. The performance of portfolios tend to be of broader economic interest, as they constitute the vehicles most widely held by investors. In turn, allowing us to demonstrate economic significance.

Following Baltas et al. (2013), we implement a portfolio selection method based on risk-budgetting. In specific, the portfolio weights are allocated in such a way that an asset's contribution to overall portfolio risk, measured in volatility, is proportional to a certain asset-specific score, s_i. After equating this asset-specific score to the return forecast of a distinct network, $\hat{y}_{t+1,i}$, the objective can be defined as:

$$w_{i,t} \cdot MCR_{i,t} \propto \hat{y}_{i,t+1}, \qquad \forall i \tag{3.6}$$

where $MCR_{i,t}$ is the marginal contribution to risk of an asset, $\delta\sigma_P/\delta w_{t,i}$. To allow for short selling, the sign of the asset-specific score must agree with $w_{t,i}$. This assures that the position is entirely determined by the sign of the scores, as such:

$$w_{i,t} \cdot MCR_{i,t} \propto \left|\hat{y}_{i,t+1}\right|, \qquad \forall i \tag{3.7}$$

As shown in Appendix 5, this objective can be transformed into the following constrained optimization objective:

$$\begin{aligned} \underset{\boldsymbol{w}_t}{\text{maximize:}} \quad & \textstyle\sum_{i=1}^{N} \left|\mathrm{s_{i,t}}\right| \cdot \log\left(\left|w_{i,t}\right|\right) \\ \text{subject to:} \quad & \sigma_{p,t} \equiv \sqrt{\boldsymbol{w_t^T} \cdot \boldsymbol{\Sigma}_t \cdot \boldsymbol{w}_t} < \sigma_{TGT} \\ & w_{i,t} > 0 \; if \; s_{i,t} > 0 \\ & w_{i,t} < 0 \; if \; s_{i,t} < 0 \end{aligned} \tag{3.8}$$

were $\sigma_{p,t}(\boldsymbol{w})$ is the volatility of the portfolio, σ_{TGT} is the volatility target, $\sum_t$ is the rolling variance-covariance matrix over the previous 255 business days and $\boldsymbol{w}_t$ is a $A\ x\ 1$ vector of portfolio weights. The yearly target volatility is set to 10 per cent, corresponding to a monthly volatility of 2.88 per cent. This seems like a sensible volatility target for a medium-risk investor.

The weights are re-optimized on a daily basis until reaching the end of the out-of-sample set. After each optimization procedure, the weight vector is rescaled by its sum, assuring the portfolio weights sum up to 1. Delaying the implementation of this "fully-invested" constraint enhances computational efficiency without altering the final outcome in terms of the risk-parity objective (Baltas, 2013).

3.3.4.1 Sharpe ratio

The resulting portfolios are evaluated by looking at performance metrics at the end of the time horizon. As a traditional performance measure, we implement the Sharpe Ratio (SR) which is defined as follows:

$$SR = \frac{\mathbb{E}(R)}{\sqrt{\sigma^2}} \tag{3.9}$$

where R is the out-of-sample return of the portfolio and σ^2 is the out-of-sample variance of the returns. The ratio therefore measures the average out-of-sample portfolio return per unit of deviation.

Taken in mind that the SR is based on mean-variance theory, it is only useful for normally distributed returns. In particular, the SR provides an inaccurate picture of risk when the return distribution is asymmetric and leptokurtic, with fatter and wider tails than the normal distribution. For this reason, we also examine the skewness and kurtosis of the out-of-sample return series.

3.3.4.2 Jobson and Korkie test

In order to evaluate the difference in Sharpe ratios statistically, we use the test statistic proposed by Jobson and Korkie (1981) after making the corrections suggested by Memmel (2003). Let $SR^{(1)}$ and $SR^{(2)}$ be two Sharpe ratios generated by two portfolios based on network (1) and (2), respectively. To test whether the difference $SR^{(1)} - SR^{(2)}$ is statistically significant, the adjusted Jobson-Korkie statistic is given by:

$$Z_{JK} = \frac{SR^{(1)} - SR^{(2)}}{\sqrt{\vartheta}} \tag{3.10}$$

which is asymptotically distributed as a standard normal when the sample size is large, with mean zero and variance ϑ:

$$\vartheta = \frac{1}{T}\left(2(1-\rho_{1,2}) + \frac{1}{2}\left(\left(SR^{(1)}\right)^2 + \left(SR^{(2)}\right)^2 - 2SR^{(1)}SR^{(2)}\rho_{1,2}^2\right)\right) \tag{3.11}$$

where $\rho_{1,2}$ is the correlation coefficient between the distinct portfolio returns and T is the number of out-of-sample returns. Clearly, the test assumes joint normality of the underlying process of portfolio returns. Only in that case we can express the difference in SR as a function of the first and the second empirical moments of the two portfolio return series.

3.3.4.3 Maximum drawdown

The maximum drawdown is an indicator of downside risk, calculated as the maximum decline from a historical peak over a certain time period. It is calculated as follows:

$$MMD(T) = \max_{\mathcal{T}\in(0,T)}\left[\max_{t\in(0,\mathcal{T})} X(t) - X(\mathcal{T})\right] \tag{3.12}$$

where T is the out-of-sample time period and $X(\cdot)$ is the cumulative return over a time period. The smaller the MMD, the better. Clearly, the power of MMD as a risk measure stems from the fact that it is independent of the empirical distribution at hand.

3.3.4.4 Benchmark

The performance metrics of portfolios corresponding to distinct networks are compared to an equally weighted portfolio. This equally weighted portfolio serves as a naive benchmark, because of its easy implementation, widespread use and out-of-sample performance.[12]

[12] Equally weighted portfolios tend to out-perform mean-variance optimized portfolios in certain out-of-sample cases (e.g. DeMiguel, Garlappi and Uppal, 2009).

Chapter 4

Results

4.1 Forecasting accuracy

Table 1 shows the mean absolute scaled errors of the forecasts for each financial asset. We compare nine different networks in total, including the FNN, LSTM and GRU with various levels of depth. When comparing forecasting methods, the method with the lowest MASE is the preferred model. In fact, scores larger than one indicate that in-sample one-step predictions from the naive method perform better than the forecast values under consideration.

Table 1: MASE scores of daily out-of-sample returns

Model	BND	VNQ	VOE	VOT	VTI
1L FNN	5.30	1.45	1.78	1.93	1.78
2L FNN	6.54	1.31	1.45	1.58	1.69
3L FNN	6.34	1.17	0.96	1.17	1.37
1L LTSM	0.76	0.55	0.63	0.57	0.77
2L LTSM	0.71	0.53	0.58	0.58	0.69
3L LTSM	0.70	0.47	0.55	0.57	0.56
1L GRU	0.71	0.53	0.58	0.65	0.58
2L GRU	0.65	0.49	0.55	0.57	0.56
3L GRU	0.66	0.48	0.56	0.57	0.57

Note: This table shows the mean absolute scaled errors of forecasts generated using nine different networks, including the FNN, LSTM and GRU with various levels of depth, for each financial asset. The financial assets include the vanguard total bond market ('BND'), total stock market ('VTI'), real estate ('VNQ'), mid-cap value ('VOE'), and mid-cap growth ('VOT') exchange traded funds. Lower MASE scores indicate stronger predictive accuracy. Scores higher than one indicate that the out-of-sample forecasts of the network under performs the one-step naive benchmark over the in-sample period.

Firstly, it can be observed that RNNs tend to outperform FNNs by a large margin in terms of mean absolute scaled errors. The plurality of FNNs have MASE scores larger than one, indicating that their performance is below par. In contrast, both the GRU and LSTM tend to outperform the naive benchmark by a large margin, with MASE scores ranging from 0.47 to 0.77. This provides us with preliminary evidence that implementing temporal connections between hidden layers improves the networks forecasting abilities.

Furthermore, it can be noted that a clear distinction between LSTM and GRU is absent. In particular, for each level of depth, the differences in MASE scores between the two units is neglectable and inconsistent. This provides preliminary evidence that both units show similar performance in financial return forecasting applications.

Secondly, it can be observed that the benefits of “deep" learning are present, but limited. Most of the RNNs and FNNs with two and three layers have lower MASE scores compared to its one-layered counterparts. This indicates that, by introducing additional hidden layers to our model, we can detect an improvement of performance. In turn, validating the hypothesis that financial time-series exhibit highly non-linear dependencies and require deep network architectures.

However, it is worth noting that the observed benefits of “deep” learning are substantially higher for FNNs compared to RNNs. This dichotomy has two possible explanations. One potential explanation stems from the fact that the incremental degrees of freedom for each additional hidden layers is larger for RNNs compared to FNNs. As mentioned earlier, this has three effects. It makes RNNs extra prone to both overfitting and vanishing gradients, and its loss function extra non-convex. Overall, this causes the cost of an extra layer to be substantially higher for RNNs compared to FNNs. Another potential explanation stems from the fact that the benefits of capturing abstract and non-linear temporal dependencies are limited.

Lastly, it can be observed that the forecasting accuracy of each network is not identical across financial assets. Clearly, the financial assets with real estate, value, growth and total market exposure (being ‘VNQ’, ‘VOE’, ‘VOT’ and ‘VTI’, respectively) have similar MASE scores for each network under consideration. This provides reassurance that the predictor set has similar forecasting power across these financial assets. However, the MASU scores of ‘BND’ seem to be strongly incompatible with other assets, particularly for feed-forward networks. This could indicate that the predictor set of bond market returns does not fully align with that of other assets. Notably, after including temporal relationships, the disparity disappears to a certain extent.

4.2 Directional accuracy

Table 2 displays the percentage of correct direction of change forecasts for all considered models. This includes the FNN, LSTM and GRU model where a sign function is wrapped around the output of the network. The scores of the binomial test are shown with (*), (**) and (***) for 10, 5 and 1 per cent significance, respectively. It is worth noting that, as we test multiple hypothesis at the same time, the chance of observing flukes and, therefore, the likelihood of incorrectly rejecting the null hypothesis (or, synonymously, "Type I errors") increases. To counteract this, we implement the conservative Bonferroni correction and divide the significance levels by the number of comparisons. Bold fonts indicate the highest percentage of correct directional forecasts for each network.

Table 2: Directional prediction, percentage correct

Model	BND	VNQ	VOE	VOT	VTI	Total
1L FNN	44.6 %	55.0 %	**56.7 %**	40.7 %	48.3 %	49.2 %
2L FNN	48.2 %	46.7 %	43.3 %	49.2 %	**56.7 %**	48.8 %
3L FNN	51.8 %	43.3 %	46.7 %	42.4 %	**53.3 %**	47.5 %
1L LSTM	50.0 %	50.0 %	56.7 %	**64.4 %**	48.3 %	53.9 %
2L LSTM	41.1 %	50.0 %	58.3 %	**61.0 %**	60.0 %	54.2 %
3L LSTM	42.9 %	51.7 %	58.3 %	**59.3 %**	58.3 %	54.2 %
1L GRU	42.9 %	36.7 %	**56.7 %**	47.5 %	53.3 %	47.5 %
2L GRU	42.9 %	53.3 %	55.0 %	57.6 %	**60.0 %**	53.6 %
3L GRU	44.6 %	58.3 %	56.7 %	**59.3 %**	58.3 %	55.5 %
Total	40.7 % *****	49.3 %	54.3 %	53.5 %	**55.2 % ***	

Note: This table shows the percentage of correct directional forecasts for each financial asset generated using nine different networks, including the FNN, LSTM and GRU with various levels of depth. The financial assets include the vanguard total bond market ('BND'), total stock market ('VTI'), real estate ('VNQ'), mid-cap value ('VOE'), and mid-cap growth ('VOT') exchange traded funds. (*), (**) and (***) indicate significance at 10%, 5% and 1% level, respectively, for two-tailed binomial tests with $H_0: p = 0.5$. The significance levels are corrected for 5-way comparison for the marginal column totals and 9-way comparison for all other elements of the table.

Firstly, it should be noted that most networks perform better than what one would expect in a purely random setting. In particular, 27 out of the 45 networks predict the correct sign with a probability of 50 per cent or higher. This result is mainly driven by the strong performance of recurrent neural networks. In fact, out of all considered recurrent neural networks, 22 out of 30 predict the correct sign with a probability of 50 per cent or higher. This again provides us with suggestive evidence that RNNs tend to perform better than FNNs in return forecasting applications. However, taken in mind that the

majority of these sign predictions yield no statistical significance after making a Bonferroni correction, we lack assurance that these results are driven by model performance instead of pure randomness.

Secondly, the marginal row totals demonstrate the percentage of total correctly classified signs for each network. It can be observed that the LSTM units outperform the FNN networks in terms direction of change forecasting irrespective of network depth, with marginal row totals ranging from 53.9 to 54.2 per cent. With the exception of one-layered networks, FNNs also tend to fall behind the performance of GRU units. Strikingly, the performance of the one-layered GRU is much worse, with a marginal row total of 47.5 per cent. This deviation comes unanticipated, taken in mind that the MASE scores of GRU and LSTM are similar.

In addition, the marginal row totals demonstrate the overall effects of depth on each distinct network. In line with the results of the previous section, the forecasting performance is weakly related to depth for the LSTM units, with marginal row totals increasing from 53.9 to 54.2 per cent. Likewise, GRU units seem to better predict future signs after adding additional hidden layers, with percentages steadily increasing from 47.5 to 55.5 per cent. Surprisingly, the benefits of depth for FNNs as observed in the previous section seem to have disappeared entirely. In fact, the marginal row totals of FNNs actually decrease with depth, from 49.2 to 47.5 per cent. This indicates that, as depth increases, the mean absolute error for each sign that is still being correctly classified decreases.

Moreover, the marginal column totals demonstrate the percentage of total correctly classified signs for each asset. In can be observed that the sign predictions of the assets with equity exposure, being 'VOE', 'VOT' and 'VTI', are quite accurate, with column totals ranging from 54.3% to 55.2%. In fact, the predictions of 'VTI' yield statistical significance at 10 per cent level, after making a conservative Bonferroni adjustment for 5-way comparison. Depending on the predictor variables used to exploit inefficiencies, this suggests that the weak and/or semi-strong form of the efficient market hypothesis does not hold.

Lastly, a strong imbalance can be observed between the financial asset with bond exposure (being 'BND'), with a column total of 40.7 per cent, and other assets. This substantiates the earlier point that the predictor set does a poor job explaining bond market performance.

4.3 Diebold-Mariano test

Table 3 to 7 demonstrate a comparison of out-of-sample forecasting performance for each distinct financial asset and method using the adjusted Diebold-Mariano test. The asterisks (*), (**), and (**) indicate that the difference in prediction errors is significant at 10, 5 and 1 percent level after applying the conservative Bonferroni correction for 8-way comparison. Bold fonts indicate the difference is significant at 5 per cent or higher using standard calculated probabilities. Our sign convention is that a positive DM test statistic indicates the column model outperforms the row model.

Table 3: Comparison of daily out-of-sample forecast of 'BND' using adjusted Diebold-Mariano test

Model	FNN2	FNN3	LSTM1	LSTM2	LSTM3	GRU1	GRU2	GRU3
FNN1	-1.97	- 0.91	**3.63** ***	**3.69** ***	**3.68** ***	**3.69** ***	**3.70** ***	**3.70** ***
FNN2		- 0.12	**5.57** ***	**5.65** *	**5.62** ***	**5.62** ***	**5.67** ***	**5.67** ***
FNN3			**2.72** *	**2.72** *	**2.71** *	**2.73** *	**2.73** *	**2.73** *
LSTM1				1.30	0.46	0.87	**2.53**	**3.08** **
LSTM2					0.71	0.57	**2.37**	**2.36**
LSTM3						0.23	1.22	1.28
GRU1							1.45	1.78
GRU2								1.33

Table 4: Comparison of daily out-of-sample forecast of 'VNQ' using adjusted Diebold-Mariano test

Model	FNN2	FNN3	LSTM1	LSTM2	LSTM3	GRU1	GRU2	GRU3
FNN1	0.86	0.71	**4.16** ***	**4.18** ***	**4.01** ***	**3.88** ***	**3.98** ***	**4.06** ***
FNN2		0.42	**3.99** ***	**4.06** ***	**4.02** ***	**4.05** ***	**4.06** ***	**4.03** ***
FNN3			**2.34**	**2.41**	**2.52**	**2.55**	**2.54**	**2.52**
LSTM1				0.07	1.05	0.23	0.95	1.34
LSTM2					1.27	0.43	1.66	1.83
LSTM3						- 1.18	0.05	1.07
GRU1							1.54	1.52
GRU2								1.36

Table 5: Comparison of daily out-of-sample forecast of 'VOE' using adjusted Diebold-Mariano test

Model	FNN2	FNN3	LSTM1	LSTM2	LSTM3	GRU1	GRU2	GRU3
FNN1	1.39	**2.53**	**2.92** **	**3.06** **	**3.07** **	**3.10** **	**3.03** **	**3.01** **
FNN2		1.45	**2.97** **	**3.63** ***	**3.62** ***	**3.58** ***	**3.61** ***	**3.61*****
FNN3			**2.46**	**8.90** ***	**11.1** ***	1.26	**2.87** *	**2.90** *
LSTM1				1.44	1.76	1.26	1.32	1.46
LSTM2					1.83	- 0.02	- 0.24	0.04
LSTM3						- 0.46	- 0.89	- 0.69
GRU1							- 0.06	0.03
GRU2								0.27

Table 6: Comparison of daily out-of-sample forecast of 'VOT' using adjusted Diebold-Mariano test

Model	FNN2	FNN3	LSTM1	LSTM2	LSTM3	GRU1	GRU2	GRU3
FNN1	**4.28** ***	**2.73** *	**6.48** ***	**6.83** ***	**6.92** ***	**6.74** ***	**6.81** ***	**6.79** ***
FNN2		1.71	**18.5** ***	**16.8** ***	**16.7** ***	**15.3** ***	**15.3** ***	**15.4** ***
FNN3			**2.54**	**2.43**	**2.42**	**2.26**	**2.48**	**2.49**
LSTM1				0.61	0.78	- 0.44	0.87	0.99
LSTM2					1.50	**- 4.45** ***	0.43	0.86
LSTM3						**- 4.60** ***	- 0.49	- 0.26
GRU1							**2.72** *	**3.99** ***
GRU2								1.25

Table 7: Comparison of daily out-of-sample forecast of 'VTI' using adjusted Diebold-Mariano test

Model	FNN2	FNN3	LSTM1	LSTM2	LSTM3	GRU1	GRU2	GRU3
FNN1	0.93	1.51	**3.52** ***	**3.98** ***	**4.03** ***	**4.21** ***	**4.24** ***	**4.26** ***
FNN2		1.22	**3.72** ***	**4.37** ***	**4.40** ***	**4.39** ***	**4.65** ***	**4.60** ***
FNN3			**4.22** ***	**2.29**	**2.25**	**3.49** ***	**3.50** ***	**3.43** ***
LSTM1				0.12	0.13	1.96	1.99	1.88
LSTM2					0.38	1.21	1.51	1.49
LSTM3						1.14	1.41	1.40
GRU1							0.68	0.67
GRU2								- 0.29

Note: Table 3 to 7 report the adjusted Diebold-Mariano test statistics for pairwise comparisons of a row model versus a column model for each financial asset. The financial assets include the vanguard total bond market ('BND'), total stock market ('VTI'), real estate ('VNQ'), mid-cap value ('VOE'), and mid-cap growth ('VOT') exchange traded funds. A positive sign of the DM statistic indicates that the column model outperforms the row model. (*), (**) and (***) indicate significance at 10%, 5% and 1% level, respectively, for individual tests after applying the conservative Bonferroni adjustment for 8-way comparisons. Bold font indicates the difference is significant at 5% level or better using standard calculated probabilities.

Firstly, it can be observed in Tables 3 to 7 that recurrent neural networks tend to outperform FNNs. The signs of the loss differentials indicate that the squared prediction errors of FFNs are higher than those of the recurrent networks, irrespective of network depth. The bold fonts indicate that, before applying the Bonferroni correction, the plurality of these differences are at least statistical significant at 5 per cent level. This aligns with the strong difference in MASU scores and direction of change forecasts between FNNs and RNNs as observed in Section 4.1 and 4.2.

The main effect of applying the Bonferroni correction is that RNNs lose their significance over the three-layered FNNs for various instances. This could indicate that RNNs perform better than FNNs in most cases because of their initial higher levels of complexity, instead of their ability to capture

relationships unattainable by FNNs.[13] In that case, ceterus paribus, as the depth and complexity of the feed-forward network increases, the forecast errors becomes statistically indifferent from RNNs. This is exactly what we observe in Table 4 and 6.

However, we expect the performance of RNNs to be, at least partly, driven by their ability to capture inter temporal relationships. There are two reasons for this line of thought. Firstly, for each network and time step, the width of the first hidden layer is fully optimized through a computationally demanding search algorithm. In this way, the network selection procedure assures an adequate amount of network complexity. Secondly, it is unlikely that our predictor set is flawless and consists of all the correct lagged predictor variables.

Thus, another explanation for the disparity could stem from the fact that the benefits of incorporating internal feedback loops in the network are asset-specific. This makes intuitive sense, as the investor base of each financial asset is heterogeneous and, therefore, respond to temporal information differently. This highlights the importance to generate forecasts for each financial asset separately.

Moreover, the Bonferroni correction could simply be too conservative, as observed by Moran (2003). If this holds for our analysis, we find strong statistical evidence that recurrent networks outperform the plain vanilla feed-forward networks, irrespective of depth.

Secondly, the tables demonstrate inconsistent evidence regarding the performance of LSTM units compared to GRU units. The sign of the DM test statistic in Table 3, 4 and 7 indicate that GRU units tend to outperform LSTM units, irrespective of network depth. In contrast, Table 5 and 6 demonstrate evidence in favour of the exact opposite. Overall, the inconsistency and the absence of statistical significance unables us generalize Chung et al.'s (2014) findings that GRU unit performs better than LSTM units in the application to the financial realm.

One potential explanation for this inconsistency stems from the differences in architecture between GRU and LSTM units. In particular, the way in which each type of gated network exposes its memory. While GRUs expose their entire memory at each time step, LSTM units apply an output gate before exposing only part of their memory. No theoretical explanation is available that explains why this

[13] Note that LSTM and GRU units have four and three times the weight parameters of a FNN given the same number of units, respectively.

benefits one asset over another. We therefore fail to conclude that the benefits of particular recurrent network types are asset-specific.

Thirdly, the tables demonstrate whether the benefits of "depth" are statistically significant for each network under consideration. The DM statistics observed between LSTM units in Table 3 to 7 indicate that the effects of "deep" learning are positive. However, the loss differentials yield no statistical significance. This is non-surprising given the neglectable benefits of depth for LSTM units observed in Section 4.1 and 4.2. Similarly, the positive loss differentials for FNNs and GRUs in Table 4 to 7 indicate that additional hidden layers allow the network to better capture future returns. Nevertheless, with the exception of Table 6, the difference in prediction errors are not statistically significant. Again, the inconsistency and the absence of statistical significance unables us to conclude that deeper networks generally outperform shallow networks in financial return forecasting applications.

Finally, as demonstrated in Appendix 6, the plurality of Augmented Dicky Fuller tests demonstrate significance at 5 and 1 per cent. This allows us to reject the presence of unit roots and provides evidence in favour of covariance stationarity. Some tests, however, demonstrate significance at levels over, but close to, the 10 per cent boundary. In other words, it can be observed that a couple of loss differentials are not precisely stationary, just as surely *no* financial time series is likely precisely stationary. However, all results can be regarded as a close approximation to the DM assumption, making the test statistics reliable.

4.4 Portfolio performance

Table 8 demonstrates the out-of-sample performance of ten different portfolios, including the EWM and the network-specific portfolios. As described in Section 3.3.4, the portfolios are constructed based on the acquired forecasts of each network. To prevent the noise inherent bond return forecasts from injecting randomness into the performance statistics, we only consider portfolio generated using the equity and real estate assets.[14] The transaction costs are not taken into account.

Firstly, it can be observed from Table 8 that the distributions of the out-of-sample portfolio return series are far from normal. The portfolio returns are highly skewed with fatter and wider tails than the normal distribution. This makes the Sharpe ratio an unreliable performance metric and,

[14] The noise inherent the bond forecasts, as observed in Section 4.1 and 4.2, causes very unstable performance metrics w.r.t risk preferences. This can be observed in Appendix 7. As a result, making us unable to draw any conclusions regarding the overall performance of each network. For this reason, we decided to only focus on portfolios generated using 'VTI', 'VOT', 'VNQ' and 'VOE'.

simultaneously, invalidates the assumption of bivariate normality underlying the JB statistic. For this reason, we shall focus on the cumulative returns (CR), the maximum draw down (MMD) and the ratio of the two.

Table 8: Portfolio performance with $\sigma_{TGT} = 10\%$

Portfolio	CR	SR	Z_{JB}	Skewness	Kurtosis	MMD	CR/MMD
1LSTM	-0.11	- 0.04	- 0.53	1.8	20.0	0.38	-0.3
2LSTM	**0.44**	**0.10**	0.35	7.3	52.3	0.12	**3.7**
3LSTM	**0.11**	**0.13**	0.69	1.8	10.1	0.07	**1.6**
1GRU	-0.23	- 0.17	- 1.61	-5.1	29.5	0.29	-0.8
2GRU	**0.12**	**0.12**	0.59	3.8	22.7	0.06	**2.0**
3GRU	**0.10**	**0.11**	0.63	1.8	9.8	0.07	**1.4**
1FNN	**0.11**	0.04	- 0.05	5.7	39.3	0.16	**0.7**
2FNN	**1.20**	**0.15**	0.62	6.6	44.5	0.15	**8.0**
3FNN	-0.23	- 0.11	- 0.89	-5.7	38.3	0.32	-0.7
EWM	0.02	0.05	n/a	-0.4	-0.02	0.04	0.5

Note: this table demonstrate the out-of-sample performance metrics for each network-based portfolio. Demonstrated are the cumulative out-of-sample return in percentages, the Sharpe ratio, the Jobson and Korkie test between the EWM and each distinct network-based portfolio, higher moments as well as the maximum draw down in terms of CRs. (*), (**) and (***) indicate significance at 10%, 5% and 1% level, respectively, for individual tests after making a conservative Bonferroni correction for 10-way comparison. Bold fonts indicate that the portfolio metric outperforms the EWM benchmark.

Secondly, it is worth noting that the plurality of network-based portfolios produce positive performance metrics. Both the two- and three layered RNN portfolios produce CRs ranging between 0.10 and 0.44 per cent, respectively. Likewise, the one- and two-layered FNN portfolio generate CRs of 0.10 and 1.20 per cent, respectively. These CRs tend to be substantially higher than that of the equally weighted benchmark. Even after adjusting the CRs for maximum draw down, it can be observed that the majority of portfolios strongly outperform the naive benchmark. This demonstrates the economic significance of network-based portfolios.

Yet, the disparity between portfolio metrics and the earlier analyzed performance measures raise questions. To start, it comes as a surprise that the two-layered FNN portfolio outperforms all other portfolios, with a risk adjusted cumulative return of 8.0. The RNNs generally performed much better compared to their feed forward counterparts based on MASE, directional accuracy and DM statistics. In terms of portfolio metrics, however, this seems not to hold true. Likewise, the dependency of portfolio performance on depth also comes as a surprise, taken in mind neglectable effects observed in earlier sections – particularly for RNNs.

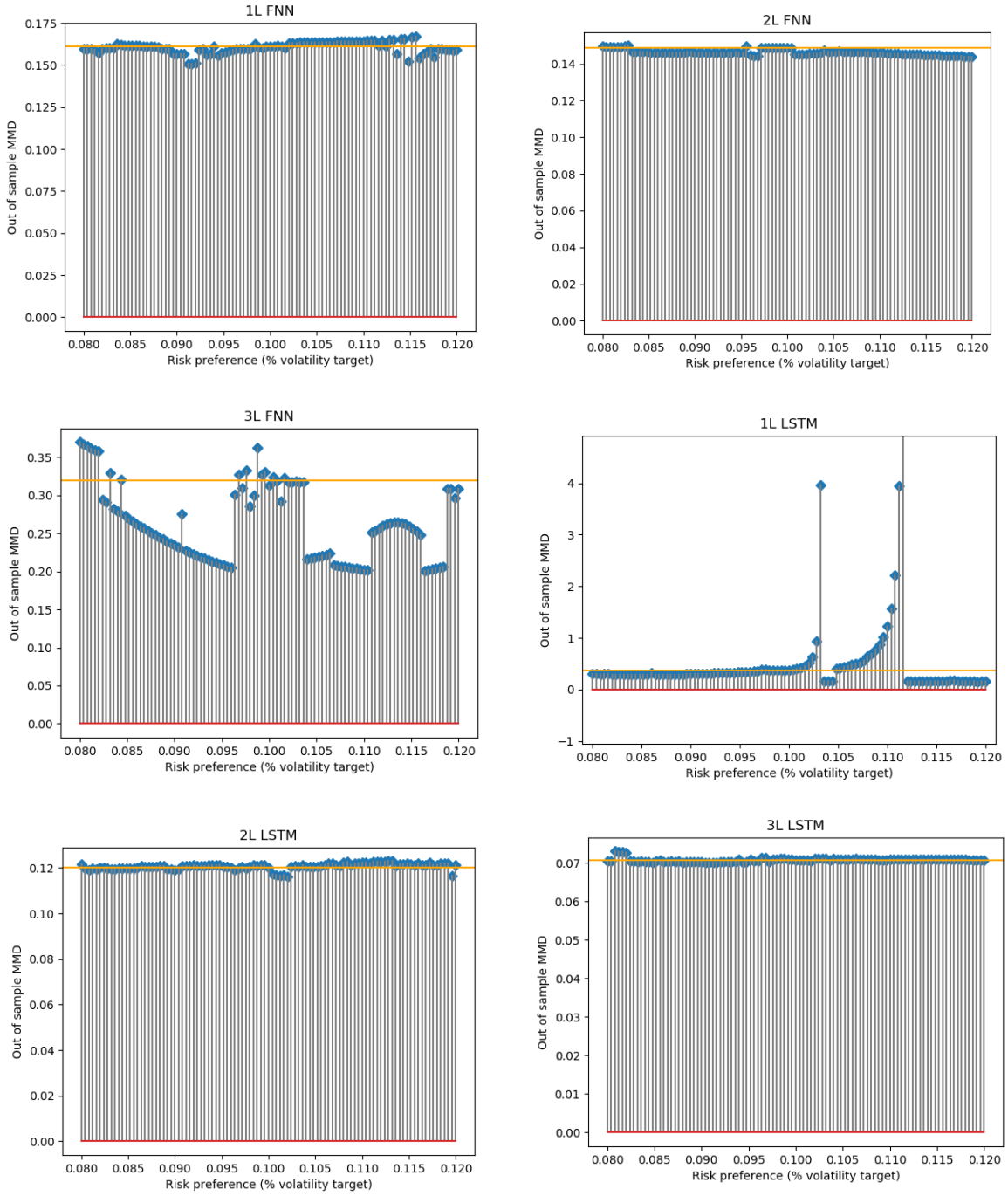
Overall, the inconsistency of the portfolio performance metrics unable us to conclude that one network-based portfolio generally outperforms the other. The discrepancy between the portfolio metrics and the MASU, direction of change and DM statistics could indicate that the portfolio approach captures information unattainable by traditional metrics. Another potential reason for the disparity stems from the fact that the portfolio metrics are very sensitive to risk preferences. Lastly, the results could be a twist of fate stemming from a small data sample. The final hypothesis cannot easily tested without enlarging our dataset.

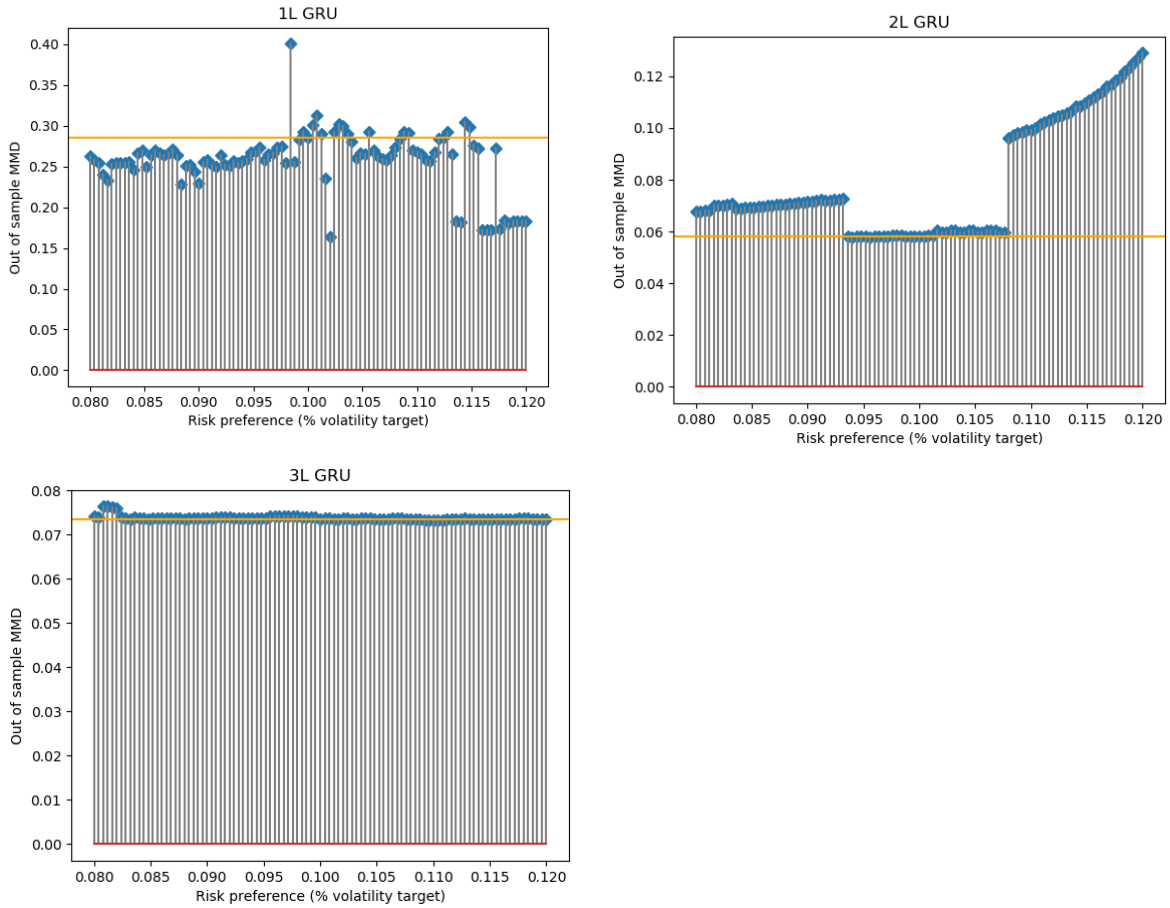
4.4.1 Portfolio robustness

To determine whether the portfolio metrics portray an inaccurate image of performance due to its sensitivity to risk preferences, we conduct a robustness check. In particular, we calculate out-of-sample MMDs for hundred portfolios with risk preferences (i.e. volatility targets) uniformly distributed between 8 and 12 per cent. The results are demonstrated in *Fig*. 8.

It can be observed the risk measure is rather stable for feed-forward neural networks and, with the exception of an outlier, also for the plurality of RNNs. In turn, invalidating the notion that the performance metrics are driven by differences in risk preferences.

Figure 8: Portfolio robustness with respect to risk preferences





Note: this figure demonstrates the out-of-sample MMDs of all the considered network-based portfolios based on hunderd different risk preferences, uniformly distributed between 8 and 12 percent. The horizontal orange line indicates the MMD corresponding to $\sigma_{TGT} = 10\%$, as portrayed in Table 8.

Chapter 5

Conclusion and limitations

5.1 Conclusions

This research aims to answer three propositions:

1. Given a particular feature setup, are certain neural network types better equipped to predict future returns than others?
2. How does the depth of the various neural networks affect its predictive power?
3. Is it possible to build a successful trading strategy based on the predictions of the network?

5.1.1 Given a particular feature setup, are certain neural network types better equipped to predict future returns than others?

From our results it is clear that recurrent neural networks perform better than the vanilla feed-forward network in the application of financial return forecasting. Both the MASU scores and direction of change forecasts are substantially higher for the LSTM and GRU units compared to FNNs. In addition, the positive DM statistics demonstrates that the squared error of RNNs is substantially lower than those of FNNs. For many financial assets, the DM statistics yielded statistical significance after a conservative Bonferonni adjustment. For some financial assets, however, we found no significance difference between the three-layered FNN and RNNs. As the exception proves the rule, we therefore state that the benefits of internal feedback loops are asset-specific.

We fail to provide evidence that the performance of RNNs is driven by their ability to capture inter temporal relationships, instead of their higher inherent levels of complexity. Nevertheless, we argue that the network selection procedure largely assures that the FNN is adequately complex. Furthermore, we indicate that it is highly unlikely that all lagged relationships are captured solely using our predictor set. For the above reasons, we conclude that the excess in performance of recurrent neural networks is, at least partly, driven by their ability to capture inter temporal relationships.

Combining the two findings, we conclude that the benefits of capturing inter temporal relationships with recurrent neural networks are asset-specific. This makes intuitive sense, as the investor base of each asset is heterogeneous.

Lastly, our results demonstrate no noteworthy difference in performance between the GRU and LSTM units. In terms of both the MASU scores and direction of change forecasts, the two types of RNNs performed similar. While some DM statistics between the GRU and LSTM units were statistically significant, the results remained inconsistent. A clear explanation as to why the GRU or LSTM architecture should perform better for certain assets over others remains to be found. Until then, we simply fail to conclude that one is better than the other in the application of return forecasting.

5.1.2 How does the depth of the various neural networks affect its ability to forecast future returns?

Our result suggests that the benefits of depth on each network are absent. The effect of depth on MASU scores and direction of change forecasts were positive, but neglectable for LSTM and GRU units. In contrast, deeper FNNs performed substantially better in terms of MASU, but weaker in terms of direction of change. For each network, the significance levels of the DM statistics were inconsistent and mostly absent. Thus, in contrast to Abe and Nakayama (2018), we fail to find statistical evidence that deeper networks generally perform better than shallow networks in forecasting financial returns. The disparity could stem from the fact that our research optimizes the width of each network through a search algorithm, while Abe and Nakayama (2018) use pre-defined architectures. The latter could undermine the levels of complexity a network can achieve in a shallow state. Overall, our findings invalidate the hypothesis that financial time-series exhibit highly abstract and non-linear dependencies and require deep network architectures.

5.1.3 Is it possible to build a successful trading strategy based on the predictions of the network?

Our results find evidence against the efficient market hypothesis. After aggregating the sign predictions of all considered networks, it has been shown that the correct sign of 'VTI' and 'VOT' was forecasted with reasonable accuracy. In fact, the direction of change forecasts of 'VTI' yielded statistical significance. It is unclear what predictors allow for this forecastability, hence, we are only able to conclude that we find evidence against the weak and/or semi-strong form of market efficiency.

To exploit these inefficiencies, we translated the forecasts in a long-short risk parity portfolio. We observed that deeper RNN portfolios outperform the naive benchmark by a large margin. Surprisingly, shallow FNNs portfolios also produced maximum draw down adjusted cumulative returns higher than the equally weighted benchmark. In fact, the two-layered FNN portfolio outperform all others portfolios. The latter came as a surprise taken in mind that the traditional performance metrics, such as MASE, direction of change and DM statistics, were significantly worse for FNNs compared to RNNs. We tested and invalidated the hypothesis that this inconsistency stems from sensitivity to risk-preference. For this reason, other research is still required to thoroughly understand the discrepancy and analyse the full potential of portfolio approaches to evaluate network performance.

5.2 Limitations

In this section we will elaborate on some limitations of our research and present potential improvements which could provide a basis for future research.

5.2.1 Hyper parameter optimization

One limitation arises due to the fact that the design and the training procedure of each network requires making presumably arbitrary choices regarding the learning rate, penalty terms, number of nodes and layers, training and validation set, and so forth. These choices are critical for the performance of the network, yet no infallible method exists to determine them. By implementing a computationally demanding search algorithm, we attempt to optimize the networks parameters. Given the immense search space, however, it is highly unlikely that we fully achieved this goal. To further optimize the network selection procedure, different kinds of hyper parameter search algorithms, such as gradient-based, Bayesian or evolutionary optimization, could have been implemented.

5.2.2 Training algorithm

Another limitation stems from the fact that training a network has a large solution space in a highly non-convex error plane. This brings forward the possibility that the results are simply numerical. Even though we aim to curtail this effect and provide stability of our results by training each network with different random seeds, this is a process that can always be refined further to acquire even more accurate results. For example, different optimization techniques besides back propagation, such as K-means or genetic algorithms could have been implemented. Qiu, Song and Akagi (2016), for instance, find that different optimization techniques tend to bring about different levels of accuracy in the domain of financial return forecasting.

5.2.3 Convolutional network

Furthermore, only a limited number of networks and architectures have been considered. There are plenty of other networks and network architectures that are worth investigating. One particularly interesting candidate is the convolutional neural network. Convolutional networks work through specialized kind of linear operations which measures dependencies between neighbouring inputs. Clearly, this network could be readily applied to financial time series, as neighbouring points have clear time dependencies.

5.2.4 Predictor set

Lastly, the strength of neural networks stems from the data that is being fed into the system. This paper only uses a small subset of the predictor variables mentioned in the literature. For instance, Harvey et al. (2016) count 316 predictive signals for describing stock return behaviour. It is therefore worth investigating whether the results presented in our research holds with different predictor sets.

Bibliography

Abadi, M., Agarwal, A., Barham, P. and Zheng, X. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv*: 1603.04467.

Abe, M and Nakayama, H (2018). Deep Learning for Forecasting Stock Returns in the Cross-Section. *arXiv*: 1801.01777

Ali, A., Hwang, L. S. and Trombley, M. A. (2003). Arbitrage risk and the book-to-market anomaly. *Journal of Financial Economics*, 69(2): 355–373.

Baltas, N., Jessop, D., Jones, C., and Zhang, H. (2013). Global Quantitative Research Monographs Trend-Following meets Risk Parity, tech. rep., UBS Limited.

Bergstra, J. and Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13: 281-305.

Bottou, L., LeCun, Y., Bengio, Y. and Haffner, P. (1998) Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.

Boyd, J. H., Hu, J. and Jagannathan, R. (2005). The Stock Market's Reaction to Unemployment News: Why Bad News Is Usually Good for Stocks. *The Journal of Finance*, 60(2): 649-672.

Buyuksalvarci, A. (2010). The Effects of Macroeconomics Variables on Stock Returns: Evidence from Turkey. *European Journal of Social Sciences*, 14(3): 404-416.

Chancharat, S., Valadkhani, A. and Havie, C. (2007). The Influence of International Stock Markets and Macroeconomics Variables on The Thai Stock Market. *Applied Econometrics and International Development,* 7(1): 221-238.

Chen, N. F., Roll, R. and Ross, S. (1986). Economic Forces and the Stock Market. *The Journal of Business, 59*(3): 383-403.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistcal Machine Translation. *arXiv*: 1406.1078.

Chordia, T., Roll, R. W. and Subrahmanyam, A. (2000). Market Liquidity and Trading Activity. *SSRN Electronic Journal*.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv*: 1412.3555.

Cohen, N., Shamir, O., and Shashua, A. (2015). On the expressive Power of deep learning: A tensor analysis. *arXiv*: 1509.05009.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4): 303–314.

Dauphin, N. Y., Pascanu, R., Gulcehre, C. and Cho, K. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *arXiv: 1405.4604.*

DeMiguel, V., Garlappi, L. and Uppal, R. (2007) Optimal *versus* naive diversification: How inefficient is the 1/N portfolio strategy? *The Review of Financial Studies, 22*(5): 1915–1953.

Diebold, F. X. and Mariano, R.S. (1995). Comparing Predictive Accuracy. Journal of Business and Economic Statistics, 13: 253-63.

Dietterich, T. G. (2000). Ensemble methods in machine learning. *Multiple Classifier Systems*, 1857: 1-15.

Duchi, J., Hazan, E and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(1): 2121–2159.

Eldan, R. and Shamir, O. (2016). The Power of Depth for Feedforward Neural Networks. *JMLR: Workshop and Conference Proceedings*, 49: 1–34.

Fama, E. F., MacBeth, J. D. (1973). Risk, Return, and Equilibrium: Empirical Tests. *Journal of Political Economy*, 81 (3): 607–636.

Fisher, K. L. and Statman, M. (2003) Consumer confidence and stock returns. *The Journal of Portfolio Management*, 30(1): 115-127.

Flannery, M. J. and Protopapadakis, A. (2002). Macroeconomic Factors Do Influence Aggregate Stock Returns. *The Review of Financial Studies*, 15(3): 751-782.

Franses, P. H. (2016). A note on the Mean Absolute Scaled Error. *International Journal of Forecasting,* 32 (1): 20–22.

Ge, R., Huang, F., Jin, C. and Yuan, Y. (2015). Escaping from saddle points: online stochastic gradient for tensor decomposition. *arXiv:* 1503.02101.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *International Conference on Artificial Intelligence and Statistics*.

Glorot, X., Bordes, A., Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. *Proceedings of Machine Learning Research,* 15(1): 315-323.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B., and Schmidhuber, J. (2017). LTSM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10): 2222-2232.

Gu, S., Kelly, B. T., and Xiu, D. (2018). Empirical Asset Pricing Via Machine Learning. *SSRN Electronic Journal*.

Gupta, M. R., Bengio, S. and Weston, J. (2014). Training highly multiclass classifiers. *Journal of Machine Learning Research,* 15(1): 1461–1492.

Hahnloser, R., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J. and Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405: 947–951.

Harvey, D., Leybourne, S. and Newbold, P. (1997). Testing the Equality of Prediction Mean Squared Errors. *International Journal of Forecasting*, 13: 281-291

Harvey, R. C., Liu, Y. and Zhu, H. (2016). … and the Cross-Section of Expected Returns. The Review of Financial Studies, 29(1):5-68.

He., K., Zhang, X., Ren, S. and Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv*: 1512.03385.

Heaton, J. (2015). *Artificial Intelligence for Humans: Deep learning and Neural Networks*. St. Louis: Heaton Research, Inc.

Hochreiter, S., and Schmidhuber, J. (1997). Long Short-term Memory. *Neural Computation,* 9(8): 1735-1780.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks,* 2(5): 359–366.

Humpe, A. and Macmillan, P. (2009). Can macroeconomic variables explain long term stock market movements? A comparison of the US and Japan. *Applied Financial Economics,* 19(2): 111-119.

Hyndman, R. J. and Koehler A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting, 22(4):* 679-688.

Jarrett, K., Kavukcuoglu, K., Ranzato, M. A., and Lecun, Y. (2009). What is the best multi-stage architecture for object recognition? *2009 IEEE 12th International Conference on Computer Vision.* 2146-2153.

Jegadeesh, N. (1990). Evidence of Predictable Behavior of Security Returns. *The Journal of Finance,* 45(3): 881–898.

Jegadeesh, N. and Titman, S. (1993). Returns to Buying Winners and Selling Losers: Implications for Stock Market Efficiency. *The Journal of Finance,* 48(1): 65–91.

Jobson, J. D. and Korkie, R. (1981). Performance hypothesis testing with the sharpe and treynor measures. *Journal of Finance*, 36: 889–908.

Keskar, N. S., Socher, R. (2017). Improving Generalization Performance by Switching from Adam to SGD. *arXiv*: 1712.07628v1

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv*: 1412.6980.
Masters, T. (1993*). Practical neural network recipes in C*. Boston: Academic Press.

McLean, D., Pontiff, J. (2016). Does Academic Research Destroy Return Predictability? *The Journal of Finance,* 61(1): 5.

Memmel, C. (2003). Performance Hypothesis Testing with the Sharpe Ratio. *Finance Letters*. 1(1).

Moran, M. (2003). Arguments for rejecting the sequential Bonferroni. *Oikos*, 100 (2): 403–405.

Nair, V., and Hinton, G.E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. *27th International Conference on Machine Learning*, 807-814.

Novy-Marx, R. (2012). Is momentum really momentum? *Journal of Financial Economics,* 103(3), 429–453.

Park, S., Wei, C. J. and Zhang, L. (2019). Does Expected Idiosyncratic Volatility Explain the Cross-Section of Expected Returns? *SSRN Electronic Journal*.

Qiu, M., Song, Y. and Akagi, F. (2016). Application of artificial neural network for the prediction of stock market returns: The case of the Japanese stock market. *Chaos Solitons and Fractals*, 85: 1-7.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088): 533–536.

Santurkar, S., Tsipras, D., Ilyas, A. and Madry, A. (2019). How Does Batch Normalization Help Optimization? *arXiv: 1805.11604.*

Sergey, L and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv*: 1502.03167

Sum, V. (2014). Effects of Business and Consumer Confidence on Stock Market Returns: Cross-Sectional Evidence. *Economics, Management, and Financial Markets*, 9(1): 21-25.

Tieleman, T. and Hinton, G. (2012) Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude. *COURSERA: Neural Networks for Machine Learning*.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. *arXiv*: 1705.08292.

Zarei, A., Ariff, M. and Bhatti, I. (2019). The impact of exchange rates on stock market returns: new evidence from seven free-floating currencies. *The European Journal of Finance*, 25(14): 1277–1288.

Zeiler, M. D. (2012). AdaDelta: An adaptive learning rate method. *arXiv*: 1212.5701.

Appendix

A.1: Predictor set

ID	Name	Measurement	Reference	Freq	Source
1	Beta	Beta of one-factor model of excess market return on excess stock return over 52 weeks.[15]	Fama and MacBeth (1973)	D	--
2	Beta squared	Squared beta of one-factor model of excess market return on excess stock return over 52 weeks.	Fama and MacBeth (1973)	D	--
3	Business confidence (M)	Business confidence index (BCI).	Sum (2014)	M	OECD
4	Consumer confidence (M)	Consumer confidence index (CCI).	Fisher and Statman (2003)	M	OECD
5	Default spread	Spread between Moody's average BAA-rated seasoned corporate bond yield and 10-year treasury bond yield.	Chen, Ross and Roll (1986)	D	Moody's; FRED
6	Dollar trading volume	The log of daily trading volume expressed in adjusted closed market price.	Chordia et al. (2001)	D	--
7-11	Exchange rate	Daily percentage change of exchange rates between USD and, RMB, CAD, MXN, JPY and EUR.	Zarei et al. (2019)	D	FRED
12	FED fund rate	Effective overnight federal reserve fund rate.	Flannery and Protopapadakis (2002)	D	FRED
13-18	GDP growth (Q)	Growth rate of seasonally adjusted real gross domestic product of USA, CHN, CAN, MEX, JPN and GER.	Flannery and Protopapadakis (2002)	Q	OECD
19	Gold price	Natural logarithm of the daily gold price.	Buyuksalvarci (2010)	D	FRED
20	Gold price change	Daily percentage change of gold price.	Buyuksalvarci (2010)	D	--
21	Housing starts (M)	Approximation of housing units being (re)built deflated by prior quarter total of US housing units.	Flannery and Protopapadakis (2002)	M	FRED; Census
22	Industrial production (M)	Industrial production index (IPI).	Humpe and Macmillan (2009)	M	FRED

[15] The *Wilshire 5000 Total Market Full Cap Index* and *10-Year treasury bond* yield are usesd as a proxy for the market portfolio and risk-free rate, respectively.

A.1: Predictor set (continued)

ID	Name	Measurement	Reference	Freq	Source
23	Inflation, consumer (M)	Monthly percentage change of consumer price index (CPI)	Chen, Ross and Roll (1986)	M	OECD
24	Inflation, producer (M)	Monthly percentage change of producer price index (PPI)	Flannery and Protopapadakis (2002)	M	OECD
25	Junk bond premium	Spread between Moody's average BAA-rated and AAA-rated seasoned corporate bond yield.	Flannery and Protopapadakis (2002)	D	Moody's
26	Momentum 3-month	Cumulative return over 3 months ending t-1.	Jegadeesh and Titman (1993); Gu et al. (2018)	D	--
27	Momentum 6-month	Cumulative return over 6 months ending t-1.	Jegadeesh and Titman (1993); Gu et al. (2018)	D	--
28	Momentum 12-month	Cumulative returen over 12 months ending t-1.	Jegadeesh (1990); Gu et al. (2018)	D	--
29	Momentum change	Daily percentage change in *6-month momentum* measure.	Gu et al. (2018)	D	--
30	Money supply (W)	The log of the M2 money supply level.	Chancharat et al. (2007)	W	FRED
31	Oil price	The log of the daily brent oil price.	Fedorova and Pankratov (2010)	D	--
32	Oil price change	Daily percentage change of brent oil price.	Fedorova and Pankratov (2010)	D	FRED
33	Reversal short-term	Cumulative return over 1 month ending t-1.	Jegadeesh (1990)	D	--
34	Reversal long-term	Cumulative return over 12 months ending t-6.	Novy-Marx (2012)	D	--
35	Trade balance (M)	Trade balance of goods and services deflated by prior quarter GDP (measured in current dollars).	Flannery and Protopapadakis (2002)	M	FRED; OECD
36	Treasury bill yield	Yield to maturity for the three-month treasury bill.	Flannery and Protopapadakis (2002)	D	FRED
37	Unemployment (M)	The number of unemployed as a percentage of the labor force.	Boyd et al. (2005)	M	FRED

A.1: Predictor set (continued)

38	Unemployment duration $_{(M)}$	Average weeks of unemployment, seasonally adjusted.	Boyd et al. (2005)	M	FRED
39	Volatility liquidity	Monthly standard deviation of daily dollar trading volume.	Gu et al. (2018)	D	--
40	Volatility earnings	Monthly standard deviation of daily returns.	Park et al. (2019)	D	--
41	Volatility idiocincratic	Monthly standard deviation of residuals from the one-factor model.	Ali, Hwang and Trombley (2003)	D	--
42	Yield spread 30Y-10Y	30 year minus the 10 year treasury bond yield.	Chen, Rossand Roll (1986)	D	FRED
43	Yield spread 10Y-2Y	10 year minus the 2 year treasury bond yield.	Chen, Ross and Roll (1986)	D	FRED
44	Yield spread 2Y-3M	2 year minus the 3 month treasury bond yield.	Chen, Ross and Roll (1986)	D	FRED
Note: the variables with subscripts $_{(W)}$, $_{(M)}$ and $_{(Q)}$ are delayed by a week, month and three months, respectively, to avoid look-ahead biases. To avoid a zero-bias, the scaling of the weekly, monthly and quarterly predictors is done by taking the mean and standard deviation of the non-zero values (*see* Section 2.10).					

A.2: Hyperparameter framework

Hyperparameter	Search distribution
Learning rate	$10^{\log(\eta)}$ with $\eta \in (0.0001, 0.01)$, evenly spaced
L1 penalty	$10^{\log(\ell 1)}$ with $\ell 1 \in (0.0001, 0.05)$, evenly spaced
L2 penalty	$10^{\log(\ell 2)}$ with $\ell 2 \in (0.0001, 0.05)$, evenly spaced
Batch size	[64, 96, 128, 256]
Neurons, first layer	[1, 2, 3, …., 44]
Optimizer	[Adam, Adagrad, RMSProp, SGD]
Patience	2
Ensemble size	10
Gridsearch iterations	250

Note: the log distribution of both the learning rate and penalty terms assures that the distribution is skewed to the right, stimulating the selection of more sensible parameter values.

A.3: Analyzed financial assets

ETF	Description	Expense Ratio	Asset Class
BND	Vanguard Total Bond Market ETF	0.05	US Fixed Income
VTI	Vanguard Total Stock Market ETF	0.04	US Equities
VNQ	Vanguard Real Estate ETF	0.12	US Real Estate
VOE	Vanguard Mid-Cap value ETF	0.07	US Equities
VOT	Vanguard Mid-Cap growth ETF	0.07	US Equities

A.4: Back propagation algorithm

The back propgation algorithm is used to efficiently evaluate the gradient, i.e. compute the first order derivatives of the loss function with regards to the network weights. In order to implement back propagation, the activation function should be differentiable and monotonically increasing. For simplicity, we apply the algorithm on the feed forward network with one hidden layer, as visualized in Fig. 2.

For purpose of derivation, we define x_{ji} as the ith input to node j, w_{ji} as the weight between the ith input and the node j, z_j as the weighted sum of the input for unit j, i.e. $z_j = \sum_i w_{ji}x_{ji}$, and a_j as the activated value of unit j, i.e $a_j = \varphi(z_j)$ where $\varphi(\cdot)$ is the ReLu activation function.

First, we derive the error over the training set using the mean-squared error loss over all output units in the output layer. That is,

$$L = \frac{1}{2}\sum_{k\in O}(y_k - a_k)^2 \tag{A.1}$$

Now, we can derive the partial derivative of the loss function w.r.t. to each weight $\delta L/w_{ji}$ required for the gradient in equation (2.19). Note that the weights can only influence the loss function through z_j. Hence, by applying the chain rule we get:

$$\frac{\delta L}{\delta w_{ji}} = \frac{\delta L}{\delta z_j}\frac{\delta z_j}{\delta w_{ji}} \Leftrightarrow \frac{\delta L}{\delta z_j}\frac{\delta}{\delta w_{ji}}\left(\sum\nolimits_i w_{ji}x_{ji}\right) \Leftrightarrow \frac{\delta L}{\delta z_j}x_{ji} \tag{A.2}$$

Now, we proceed with finding $\delta L/\delta z_j$. Using the chain rule, we obtain:

$$\frac{\delta L}{\delta z_j} = \frac{\delta L}{\delta a_j}\frac{\delta a_j}{\delta z_j} \tag{A.3}$$

Notice that $\delta a_j/\delta z_j$ is simply the derivative the activation function, in our case the ReLu. As a result we obtain:

$$\frac{\delta L}{\delta z_j} = \begin{cases} 0 & if\ z_j < 0 \\ \dfrac{\delta L}{\delta a_j} & if\ z_j > 0 \end{cases} \tag{A.4}$$

Finally, we have to find $\delta L/\delta a_j$:

$$\frac{\delta L}{\delta a_j} = \frac{\delta}{\delta a_j}\left(\frac{1}{2}\sum_{k\in O}(y_k - a_k)^2\right) \Leftrightarrow \frac{\delta}{\delta a_j}\left(\frac{1}{2}(y_j - a_j)^2\right) \Leftrightarrow -(y_j - a_j) \tag{A.5}$$

Notice that for all cases were $k \neq j$, the derivative will be zero. This allows us us to drop the summation over the output units. After substitution, we can derive the partial derivative of the loss function w.r.t. w_{ji} as follows:

$$\frac{\delta L}{\delta w_{ji}} = \begin{cases} 0 & if\ z_j < 0 \\ -(y_j - a_j)x_{ji} & if\ z_j > 0 \end{cases} \tag{A.6}$$

This therefore provides us with the partial derivatives of the weight connections between the hidden layer and the output layer. Similar steps can be conducted to derive the partial derivatives between the input layer and the hidden layer. The back propagation technique can be extended to allow for one or more layers.

A modified version of this algorithm, called backpropagation through time (BPTT), is used for RNNs (Gruslys et al., 2016).

It is worth noting that, by adding a penalty term to our loss function as in equation (2.31), we have to add another part to equation (A.6). In specific, the derivative of the elastic net is calculated as follows:

$$\frac{\delta}{\delta w_{ji}}\left(\lambda_1|w_{ji}| + \lambda_2 w_{ji}^2\right) = \begin{cases} -\lambda_1 + 2\lambda_2 w_{ji} & if\ w_{ji} < 0 \\ \lambda_1 + 2\lambda_2 w_{ji} & if\ w_{ji} > 0 \end{cases} \tag{A.7}$$

The sum of (A.6) and (A.7) then provides us with the partial derivative of the penalized loss function. These partial derivatives then determine the weight adjustments in the gradient descent algorithm.

A.5: Derivation of portfolio optimization objective

As demonstrated by Baltas et al. (2013), the original risk-parity objective:

$$w_i \cdot MCR_i = \text{constant}\,, \qquad \forall i \tag{A.8}$$

can be rephrased into a non-linear constrained optimization problem:

$$\max \sum_{i=1}^{N} \log(w_{i,t}) \;\; s.t. \sigma_{p,t} \equiv \sqrt{\boldsymbol{w_t^T} \cdot \boldsymbol{\Sigma_t} \cdot \boldsymbol{w_t}} < \sigma_{target} \tag{A.9}$$

with the constraint that $\sum_{i=1}^{N} w_{i,t} = 1$ applied through normalization after the optimization procedure. The corresponding Lagrangian is then calculated as follows:

$$\mathcal{L}(\boldsymbol{w_t}) = \sum_{i=1}^{N} log(w_{i,t}) - \lambda(\sigma_{p,t} - \sigma_{target}) \tag{A.10}$$

From this, the partial derivatives of the Lagrangian with respect to each portfolio weight are calculated as follows:

$$\frac{\delta \mathcal{L}(\boldsymbol{w_t})}{\delta w_{i,t}} = \frac{1}{w_{i,t}} - \lambda \left(\frac{\sigma_{p,t}}{w_{i,t}} \right), \qquad \forall i \tag{A.11}$$

Note that $\sigma_{t,p}/w_{t,i} = MCR_{t,i}$. Hence, setting the above equation equal to zero and substitution therefore yields:

$$\frac{1}{w_{i,t}} - \lambda\big(MCR_{i,t}\big) = 0 \;\Leftrightarrow w_{i,t} \cdot MCR_{i,t} = \frac{1}{\lambda}, \qquad \forall i \tag{A.12}$$

This implies that the constrained optimization requires the total contribution of risk of each asset to equal a constant: the reciprocal of the Lagrangian multiplier λ. In other words, the risk parity objective is satisfied.

By applying the same trick, we can assure that the contribution of each asset to the total portfolio volatility is proportional to a non-negative asset-specific score, s_i. That is, after multiplying $\log(w_{i,t})$ with s_i in the objective function, we obtain:

$$\mathcal{L}(\boldsymbol{w}_t) = \sum_{i=1}^{N} s_{i,t} \cdot log(w_{i,t}) - \lambda(\sigma_{p,t} - \sigma_{TGT}) \tag{A.13}$$

Then, after calculating the partial derivatives of the Langrarian and equating these to zero, we obtain:

$$w_{i,t} \cdot MCR_{i,t} = \frac{s_{i,t}}{\lambda}, \qquad \forall i \tag{A.14}$$

That is, the total contribution to risk of each asset is equal to a certain positive asset-specific score scaled by the inverse of the Lagrangian multiplier, λ. Consequently, the assets contribute an amount to overall volatility proportional to a non-negative asset-specific score:

$$w_{i,t} \cdot MCR_{i,t} \propto s_{i,t}, \qquad \forall i \tag{A.15}$$

Finally, we can allow for short selling and negative asset specific scores by assuring that $w_{i,t} > 0$ if $s_{i,t} > 0$ and $w_{i,t} < 0$ if $s_{i,t} < 0$, while wrapping absolute values around the weights and asset specific score in the objective function:

$$\begin{aligned} \underset{\boldsymbol{w}_t}{\text{maximize:}} \quad & \textstyle\sum_{i=1}^{N} \left|s_{i,t}\right| \cdot \log\left(\left|w_{i,t}\right|\right) \\ \text{subject to:} \quad & \sigma_{p,t} \equiv \sqrt{\boldsymbol{w}_t^{\boldsymbol{T}} \cdot \boldsymbol{\Sigma}_t \cdot \boldsymbol{w}_t} < \sigma_{target} \\ & w_{i,t} > 0 \; if \; s_{i,t} > 0 \\ & w_{i,t} < 0 \; if \; s_{i,t} < 0 \end{aligned} \tag{A.16}$$

A.6: Augmented Dicky Fuller tests over loss differentials

These table reports the calculated probabilities of the Augmented Dicky Fuller test over the loss differentials of each method over financial asset 'BND'. The null hypothesis of the ADF is that there is a unit root, with the alternative that there is no unit root. The plurality of tests reject the null and provide evidence in favour of covariance stationarity. The results can be regarded as a close approximation to the DM assumption, making the test statistics reliable.

Calculated probabilities of Augmented Dicky Fuller tests for 'BND' loss differentials

Model	FNN2	FNN3	LSTM1	LSTM2	LSTM3	GRU1	GRU2	GRU3
FNN1	-1.97	0.109	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
FNN2		0.049	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
FNN3			0.125	0.126	0.13	0.13	0.13	0.13
LSTM1				<0.01	<0.01	0.493	<0.01	<0.01
LSTM2					<0.01	<0.01	<0.01	<0.01
LSTM3						<0.01	<0.01	<0.01
GRU1							<0.01	<0.01
GRU2								0.59

Calculated probabilities of Augmented Dicky Fuller tests for 'VNQ' loss differentials

Model	FNN2	FNN3	LSTM1	LSTM2	LSTM3	GRU1	GRU2	GRU3
FNN1	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
FNN2		<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
FNN3			<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
LSTM1				<0.01	<0.01	<0.01	<0.01	<0.01
LSTM2					<0.01	<0.01	<0.01	<0.01
LSTM3						<0.01	<0.01	<0.01
GRU1							<0.01	<0.01
GRU2								<0.01

Calculated probabilities of Augmented Dicky Fuller tests for 'VOE' loss differentials

Model	FNN2	FNN3	LSTM1	LSTM2	LSTM3	GRU1	GRU2	GRU3
FNN1	<0.01	<0.01	<0.01	0.01	<0.01	<0.01	<0.01	<0.01
FNN2		0.01	0.07	<0.01	0.09	0.06	0.06	0.07
FNN3			<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
LSTM1				<0.01	<0.01	<0.01	<0.01	<0.01
LSTM2					<0.01	<0.01	<0.01	<0.01
LSTM3						<0.01	<0.01	<0.01
GRU1							<0.01	<0.01
GRU2								<0.01

Calculated probabilities of Augmented Dicky Fuller tests for ‘VOT’ loss differentials

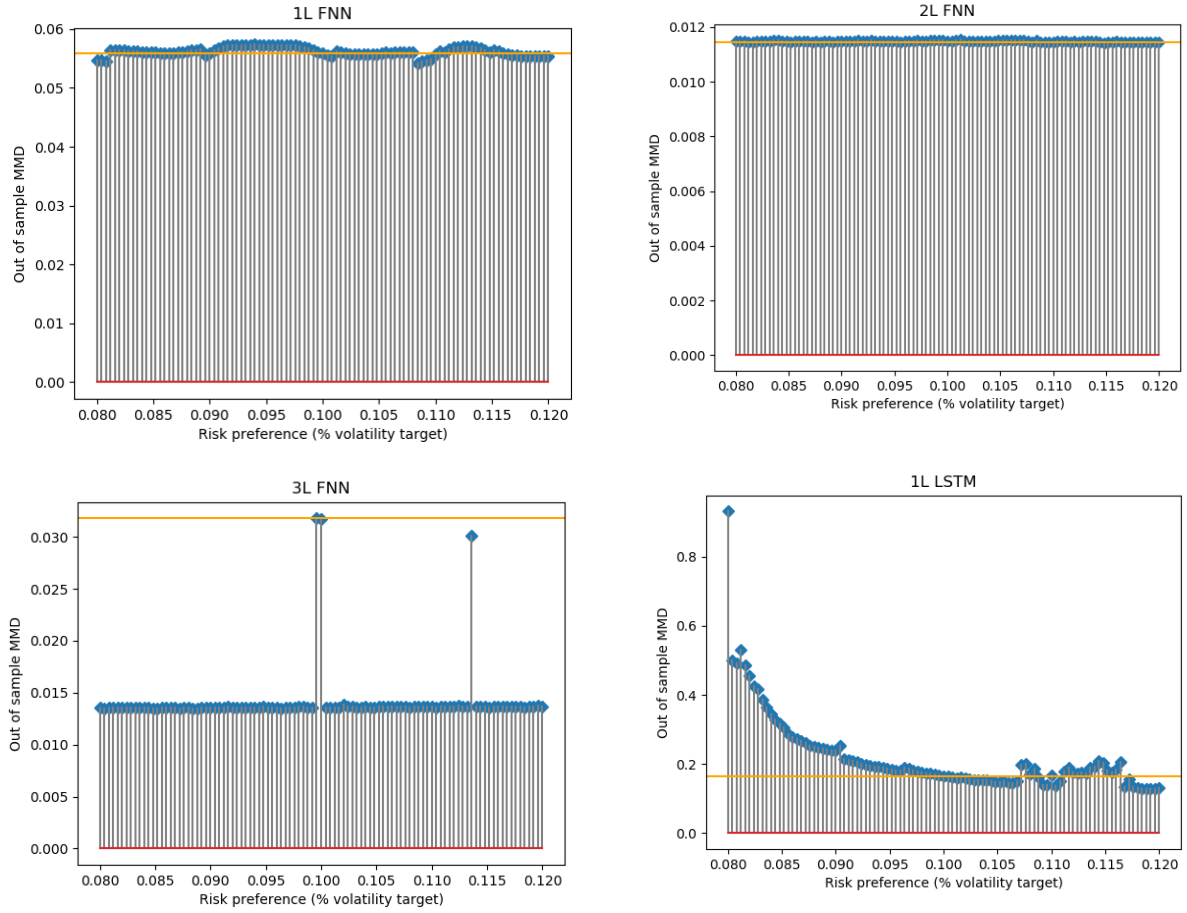
Model	FNN2	FNN3	LSTM1	LSTM2	LSTM3	GRU1	GRU2	GRU3
FNN1	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
FNN2		<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
FNN3			0.03	0.03	0.04	0.03	0.03	0.03
LSTM1				0.09	0.09	<0.01	0.08	0.07
LSTM2					0.18	<0.01	<0.01	<0.01
LSTM3						<0.01	<0.01	<0.01
GRU1							<0.01	<0.01
GRU2								<0.01

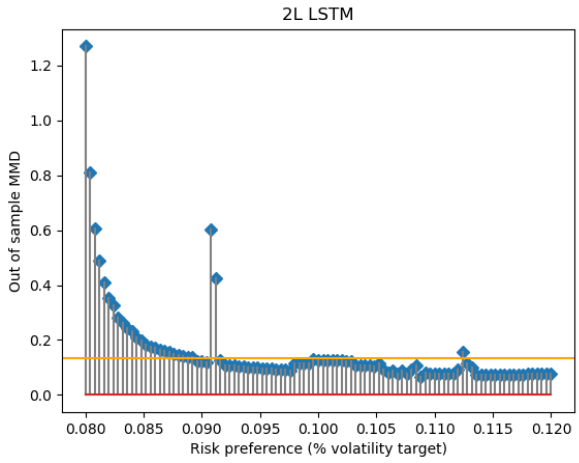
Calculated probabilities of Augmented Dicky Fuller tests for ‘VTI’ loss differentials

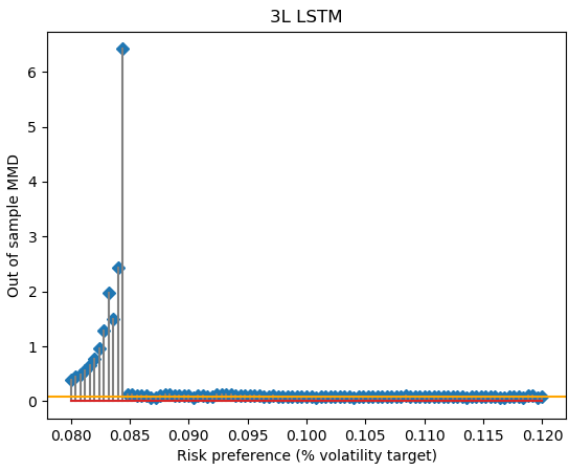
Model	FNN2	FNN3	LSTM1	LSTM2	LSTM3	GRU1	GRU2	GRU3
FNN1	<0.01	<0.01	<0.01	0.04	0.08	<0.01	<0.01	<0.01
FNN2		<0.01	0.14	0.02	0.02	0.06	0.06	0.07
FNN3			<0.01	<0.01	<0.00	<0.01	<0.01	<0.01
LSTM1				<0.01	<0.01	<0.01	<0.01	<0.01
LSTM2					<0.01	<0.01	<0.01	<0.01
LSTM3						<0.01	<0.01	<0.01
GRU1							<0.01	<0.01
GRU2								<0.01

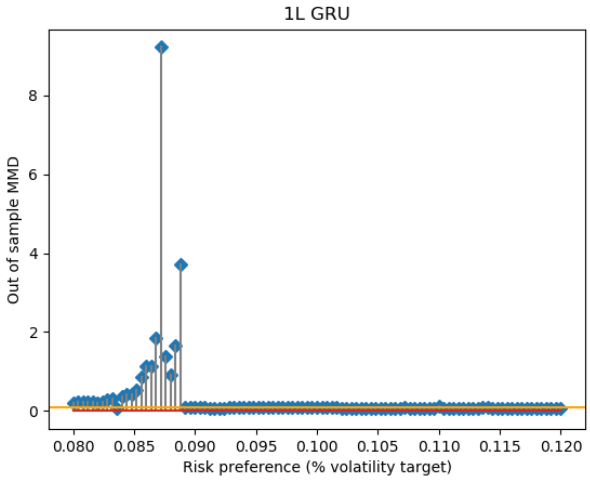
A.7: Robustness of portfolios (incl. ‘BND’) to risk preference

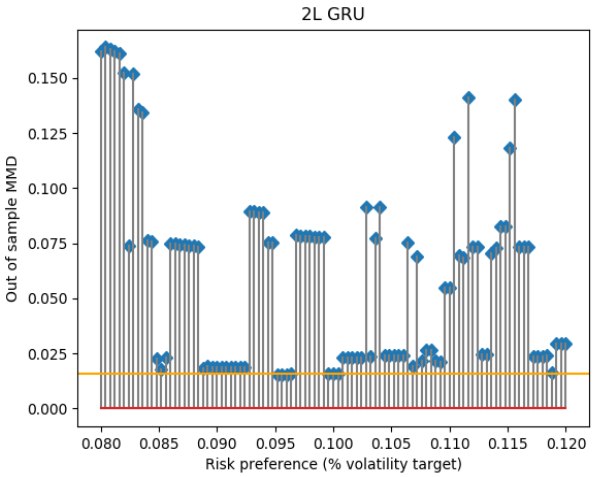
This figure demonstrates the out-of-sample maximum draw down of all the considered network-based portfolios based on hunderd different risk preferences. The horizontal orange line indicates the MMD corresponding to $\sigma_{TGT} = 10\%$. In line with Section 4.1 and 4.2, we expect the noise of the results to stem from the inability of the networks to accurately forecast ‘BND’ returns using the implemented predictor set.

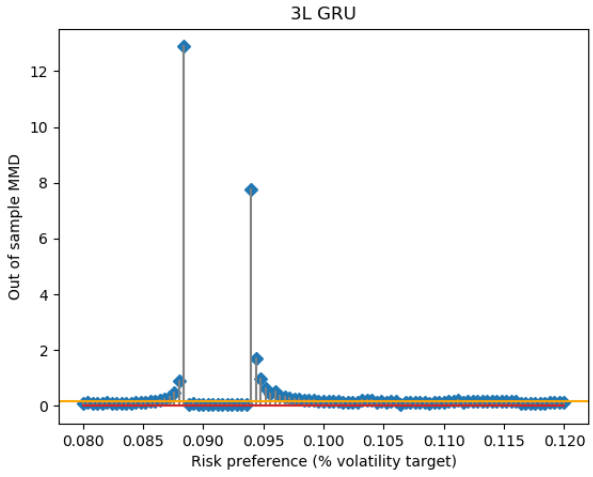



2L LSTM
Out of sample MMD
Risk preference (% volatility target)


3L LSTM
Out of sample MMD
Risk preference (% volatility target)


1L GRU
Out of sample MMD
Risk preference (% volatility target)


2L GRU
Out of sample MMD
Risk preference (% volatility target)


3L GRU
Out of sample MMD
Risk preference (% volatility target)