Erasmus University Rotterdam
Erasmus School of Economics

Bachelor Thesis

Econometrics and Operations Research

Enhancing the Momentum Strategy Using Recurrent Neural Networks

Author: Luc Cronenberg (455663)
Supervisor: S.H.L.C.G. Vermeulen
Second assessor: X. Xiao

July 7, 2019

Abstract

The momentum strategy of buying assets which performed well in the past and shorting those which performed poorly has been shown to generate persistent abnormal returns. Existing literature suggests combining this concept with predictive models in order to determine trading positions for momentum portfolios. In this paper, we propose the use recurrent neural networks in combination with such momentum strategies. Using data from 1953 to 2019, our empirical results show that utilizing recurrent neural networks results in similar profitability compared to classical methods. However, as main advantage, we find that up to a 15% improvement can be made with respect to Sharpe ratio. Additionally, during times of relative economic turmoil, recurrent neural networks are found to perform significantly better than the benchmark strategies.

Contents

1 Introduction

Momentum based investment strategies are considered one of the main anomalies in financial economics, generating excess returns unexplained by traditional asset pricing models. Introduced by Jegadeesh and Titman (1993), momentum refers to the persistence in asset returns; stocks performing well in the past tend to perform well in the near future, while the contrary holds for assets with poor past performance.

Applying momentum in finance, Jegadeesh and Titman (1993), proposed an investment strategy which buys stocks or assets that have had the best performance over the past 2 to 12 months (winners) and shorts the assets which performed worst in this period (losers). This strategy is referred to as the winner-minus-loser (WML) strategy and is found to generate a monthly profit of at least 1% (Jegadeesh and Titman (1998, 2001)). Besides this classical strategy, Jegadeesh and Titman (2001) showed that a strategy in which only the (historically) best performing assets are bought generate excess returns, and, as opposed to buying both winners and shorting losers, this strategy is unaffected by the case where losers generate positive returns. This strategy is referred to as the winner only (WO) strategy.

Although Jegadeesh and Titman (1998, 2001) used a set time horizon and holding period, looking back 12 months and generating predictions for the next, many papers have found evidence of abnormal excess returns across different time periods, asset types and forecasting horizons (Hurst, Ooi, and Pedersen (2017), Lemperiere et al. (2014), Baz et al. (2015)). In addition to these variables, using different trading rules also affects profits. In this paper we are particularly interested in the strategy proposed by Kim (2019). They combine the WML and WO strategies, creating a 'selective' winner-minus-loser (SWML) strategy; buying the top decile of stocks only if their *expected* return is positive and shorting the worst performing decile of stocks only if its *expected* return is negative.

While WML and WO trading strategies perform well under most economic conditions, Barroso and Santa-Clara (2015) observed extreme losses during market crashes, with Daniel and Moskowitz (2016) reporting a return of the winner portfolio in March 2009 of 8% while that of the loser portfolio outperforms it greatly with a return of 169%. In order to overcome this issue, Barroso and Santa-Clara (2015) propose using volatility-scaled returns, improving overall performance greatly.

Using the SWML strategy, it is necessary to compute predictions for the return in the holding period. A simple yet effective method for this purpose is using the look-back period-based method. Shown its effectiveness in Moskowitz, Ooi, and Pedersen (2012) and Barroso and Santa-Clara (2015), this approach uses the product of the past 12 months as prediction for the next and may be formulated as $\prod_{i=1}^{12}(1 + r_{t-i}) - 1$, where r_{t-i} is the return in month $t - i$. Besides this more ordinary method, recently the use of machine learning, and especially deep neural networks, have found their way into financial time-series forecasting. Having already been shown to perform well in fields such as medicine (Miotto et al. (2017)), demand forecasting (Ke et al. (2017)) and finance (Heaton, Polson, and Witte (2016)), Kim (2019) showed that machine learning can also be applied to the forecasting of trends for the purpose of applying them to momentum strategies. They established a significant improvement in returns when forecasting them using an architecture known as a stacked denoising autoencoder, compared to the look-back approach. However, while Kim (2019) uses deep learning techniques with promising results, more modern architectures such as recurrent neural networks (RNNs) are not considered. Many papers have found that RNNs can accurately predict financial time series, for instance, in forecasting currency volatility (Dunis and Huang (2002)) and stock return prediction Rather, Agarwal, and Sastry (2015). Especially the most modern RNN architectures such as Long-Short-Term-Memory and Gated Recurrent Units have been shown to perform well compared to regular feedforward artificial neural networks.

In this study, our main objective is to investigate whether recurrent neural networks can be used for the forecasting of momentum portfolio returns, and if so, how their performance compares to that of already investigated methods. More specifically, we will use the long-short-term-memory and gated recurrent unit architectures, while the models used by Kim (2019) will be used for comparative purposes. These models include the look-back period-based approach, support vector regression, and a stacked denoising autoencoder. Then, after expected returns are generated using these models, trading positions will be determined by the SWML strategy. In our empirical study, the data will be obtained from the Kenneth French data library (French (2019)), which contains decile momentum portfolios containing stocks from NYSE, AMEX, and NASDAQ from 1927 to present. Our research contributes to existing literature by combining the selective winner-minus-loser strategy with recurrent neural networks, which, to the best of our knowledge, has not been studied before.

The remainder of the paper is organized as follows. In section 2 we discuss our data, section 3 describes the forecasting models as well as their training procedure. Then, in section 4 we present our results, while in section 5 we summarize and discuss our main findings.

2 Data

For comparative purposes as well as the ease of access, the same data set is used as by Kim (2019), namely the data set containing momentum-based portfolios from 1927 to present as obtained from the Kenneth French data library (French (2019). These portfolios are constructed according to prior return data of the previous 2-12 months and consist of stocks from the NYSE, AMEX, and NASDAQ markets which have sufficient prior return data. Furthermore, there are some requirements set the for returns. Each stock must have a price at the end of month $t-13$, where t is the month for which the portfolio is constructed, a good return for month $t-2$, and any missing data must be denoted as -99.0.

Besides the data directly obtained from Kenneth French data library, which is used for generating the prediction, we will also apply volatility scaling to this original data set in order to achieve more stable performance. Volatility scaling is done by taking the product of the ratio of target volatility and actual volatility

$$r_t^* = \frac{\sigma_{\text{ target}}}{\hat{\sigma}_t} \cdot r_t \tag{1}$$

where r_t is the return of the plain momentum strategy and $\hat{\sigma}_t$ the volatility of the predicted return which is computed from the daily returns of the previous 6 months. The scaled volatility is computed for both a target volatility of 12% and 15% as was recommended by Barroso and Santa-Clara (2015) and Lim, Zohren, and Roberts (2019) respectively.

3 Methodology

This section discusses the different strategies and models which are applied in this paper. First, a quick summary of all strategies is presented, then all different forecasting models will be discussed, and finally, the training procedure of the models is clarified.

3.1 Strategies

As mentioned in the introduction, many different momentum strategies have been proposed and investigated. In this paper we will particularly investigate the strategy proposed by Kim (2019). Theoretically, their Selective Winner-Minus-Loser (SWML) strategy should always outperform strategies in which forecasting models are used in conjuncture with WML or WO. The idea behind the SWML strategy is that that top 10% of stocks (top decile) is bought only when its expected return is positive, and nothing is done with it otherwise. On the other hand, the worse performing 10% of stocks (bottom decile) is shorted only when its expected return is negative, and nothing is done otherwise. This could lead to the situation where nothing is invested in stocks, in which case one could invest in say risk free bonds such as treasury bonds. For comparison reasons, we will also compute the performance measures for the classic WML and WO strategies.

3.2 Look-back

In order to predict the (sign of the) return in the holding period one can use the following formula the day before the start of month t

$$\prod_{i=1}^{12}(1+r_{t-i})-1 \tag{2}$$

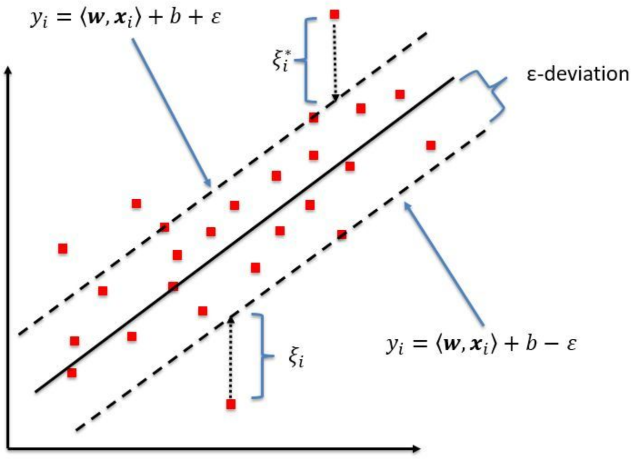
where r_{t-i} represents the return in month $t-i$. Several papers have shown that this is a simple yet effective method of predicting momentum portfolios, also in combination with a range of momentum strategies (Moskowitz, Ooi, and Pedersen (2012)), Kim (2019)).

3.3 Support Vector Regression

Support vector regression (SVR) is a shallow machine learning technique already widely applied in many fields such as engineering (Clarke, Simpson, and Griebsch (2003)), travel-time prediction (Chun-Hsin Wu, Jan-Ming Ho, and Lee (2004)) as well as in financial time series prediction (Tay and Cao (2001)). First introduced by Drucker et al. (1996), SVR is based on the similarly named support vector machines (SVM) (which is used for classification), taking most features from it, only slightly changing the loss function to account for the distance from a point to the fitted hyperplane.

While most traditional regression techniques attempt to find a function such that it produces the least deviation between predicted and true values, SVR attempts to minimize some generalized error bound in order to achieve a generalized performance (Basak, Pal, and Patranabis (2007)). The main idea behind this is that in computing the loss, only errors which are outside a certain margin ϵ from the fitted curve/hyperplane are considered in the loss functions, as is also illustrated in figure 1 below. In general, SVR can be formulated as

$$\text{Minimize } \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$
$$\text{subject to } \begin{cases} y_i - \langle \omega, x_i \rangle - b \leq \varepsilon + \xi_i \\ \langle \omega, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases} \tag{3}$$



$y_i = \langle \mathbf{w}, \mathbf{x}_i \rangle + b + \varepsilon$
ξ_i^*
ε-deviation
ξ_i
$y_i = \langle \mathbf{w}, \mathbf{x}_i \rangle + b - \varepsilon$

Figure 1: Support Vector Regression

where w represents the the vector of all coefficients in the regression function $y_i = \langle w, x_i \rangle + b + \varepsilon$ (similar the the β vector in OLS), C is the penalty parameter for the error, and ξ_i and ξ_i^* are the slack variables accounting for the deviation from ϵ.

Originally, SVR and SVM are linear classifiers, however, in order to deal with non-linear trends such as in stock returns, one can apply the kernel trick. The kernel trick is an approach to operating in a high-dimensional space, mapping the non-linear function to a linear space without explicitly computing all coordinates. This is done using a kernel function, which replaces the $\langle w, x_i \rangle$ with $\langle \varphi(x), \varphi(y) \rangle$ in equation (3) above. In our paper we will be using a Gaussian Kernel, to stay consistent with Kim (2019) as well as it has found to perform well in stock prediction. The Gaussian Kernel is specified as

$$\langle \varphi(x), \varphi(y) \rangle = \exp\left(-\gamma\|x - y\|^2\right) \tag{4}$$

where $\gamma = \frac{1}{2\sigma^2}$, which can be tuned to fit the data best.

3.4 Autoencoders

An autoencoder is a specific architecture within artificial neural networks that learns to represent the input data in a limited amount of factors (latent space). In other words, it learns to extract the hidden features in a certain dataset. Since we work with portfolio returns of stocks that share the feature that they performed similarly in the past as well as that they come from the same stock markets, it may be expected that they share some common driving factors.

Before discussing the specifics of an autoencoder, it is helpful to understand the basics behind artificial neural networks (ANNs). ANNs are based on how the neurons in our brain interact and make computations. Due to its complexity it performs exceptionally well in finding hidden relationships within a data set, may it be linear or non-linear (Zhang (2003)). Figure 2 shows a general structure of an ANN. The 'nodes' all the way to the left represents the input data, while the nodes in the last layer, the right most column, represents the outputs. All layers in between the input and output are the 'hidden' layers,

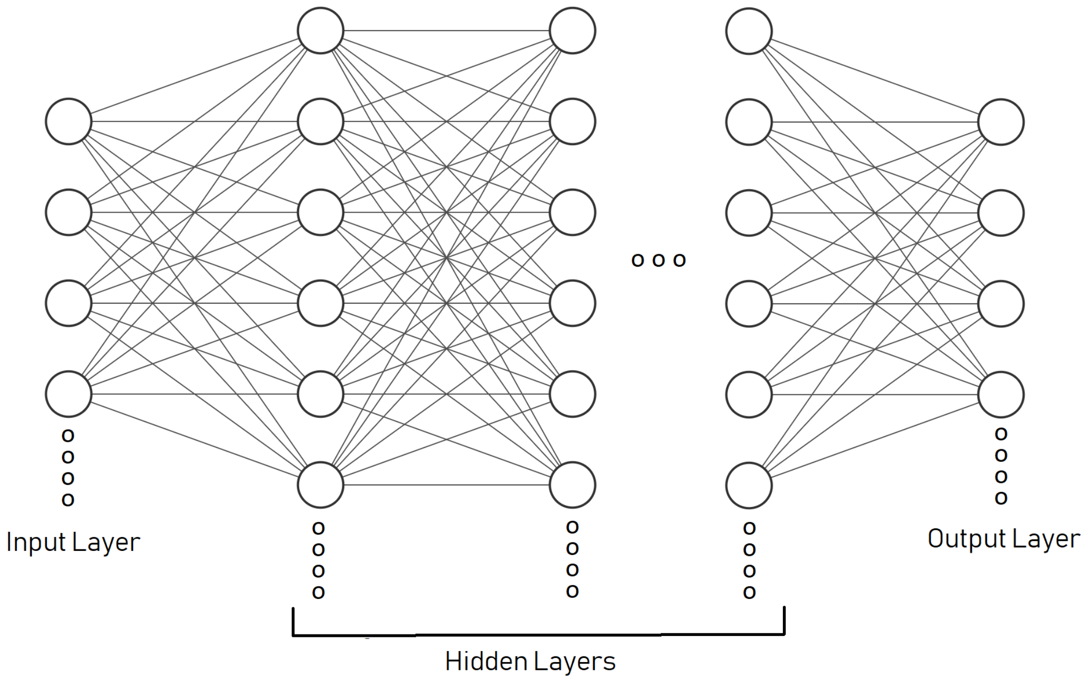


Figure 2: General Artificial Neural Network Structure

As can be seen, all the nodes in all the layers are densely connected; each node in one layer is connected to each node in the previous layer. When the network is used for training or prediction, information 'flows' from one side of the network to the other. The way this is done is determined by the activation of each node and, subsequently, the activation function. This activation function is a function of the weights of each of the connected nodes as well as an assigned bias. A general activation function can be formulated as

$$z(W, X, b) = f(W^T X + b) \tag{5}$$

for some fixed f, with W representing the vector containing all weights, X the input vector and b the vector of corresponding biases. In this paper two specific activation functions are used, the sigmoid function and the tanh function. The sigmoid function is an s-shaped curve which outputs a value between 0 and 1. It is a popular activation function within machine learning since traditionally architectures such as artificial neural networks are used for classification and an output between 0 and 1 may then be considered as probabilities of a certain class. On the other hand, the tanh function ranges from -1 to 1. This has the advantage that the negative inputs will be mapped strongly negative and inputs close to zero will also be mapped close to zero by the tanh function. Additionally, both functions also benefit from being highly differential, which is of great benefit to updating the weights and bases during the training of the network.

$$\sigma(W, X, b) = \frac{1}{1 + e^{W^T X + b}} \qquad (6) \qquad \tanh(W, X, b) = \frac{2}{1 + e^{-2(W^T X + b)}} - 1 \qquad (7)$$

In order to map the inputs to the outputs as accurately as possible, the network needs to be trained by tuning the weights and biases of all nodes. The training is done by providing the network with a set of inputs and corresponding outputs, after which the network repeatedly updates its parameters by means of back propagation (a tuning procedure using the gradient error function to update the parameters, similar to gradient descent).

An autoencoder is a bottleneck type of artificial neural network that uses the features of an ANN to represent data in a user specified number of dimensions. More specifically, during the training of the network, the input and output are defined as the same, while the hidden layer(s) contain less nodes than the input/output. This forces the network to train the parameters in such a way that it contains as much information as possible in the limited number of nodes, as it attempts to minimize the reconstruction loss. As can be seen in figure 3, an autoencoder consists of an encoder and a decoder. The encoder compresses the input into a latent representation, while the decoder attempts to reconstruct the original output from the latent space. In most uses of the autoencoder, the network is trained using the whole network structure, after which the encoder is used to compress the input data which can then be used for a multitude of other purposes for which a low dimensionality is desirable.

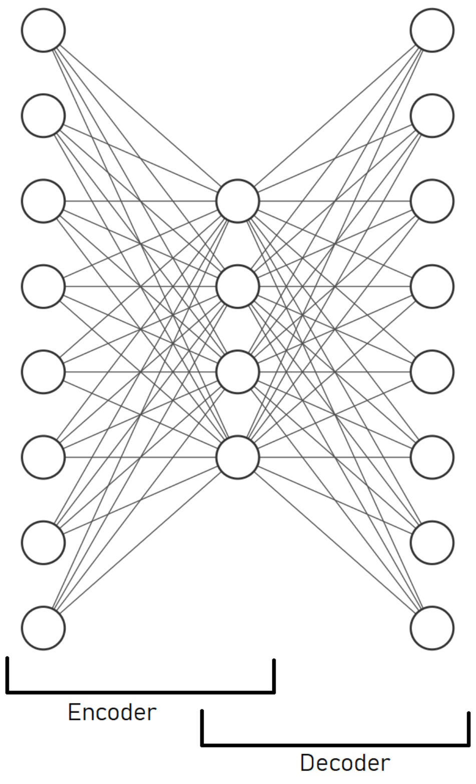


Figure 3: Autoencoder structure

More formally, the process of an autoencoder can be formulated as

$$\begin{aligned} \mathbf{h} &= f_e(\mathbf{x}) = s_e\left(\mathbf{W_e x} + \mathbf{b_e}\right) \\ \mathbf{x_r} &= f_d(\mathbf{x}) = s_d\left(\mathbf{W_d h} + \mathbf{b_d}\right) \end{aligned} \tag{8}$$

where $f_e(x)$ is the encoder function, and $f_d(x)$ that of the decoder. Then, h can be seen as the latent factors and x_r the reconstructed input. This is then compared to the original input using a certain loss function $\mathcal{L}\left(\mathbf{x}, \mathbf{x_r}\right)$ which in this paper is specified as the mean squared error.

In this study we use an architecture known as a *stacked* autoencoder. The main idea of a stacked autoencoder is that you train multiple single layered autoencoders consecutively, each time using the encoded output from the previous encoder as input for the next. Particularly, we use a stacked autoencoder consisting 3 basic autoencoders, each with encoded output of 6 nodes, as can be seen in figure 4 below. The 12 input nodes represent the returns in the previous 12 months, while the single output node is that of the next month which is trained with the final layer as a simple regression using the output from the last autoencoder. This structure was chosen as it was found to perform best by Kim (2019).

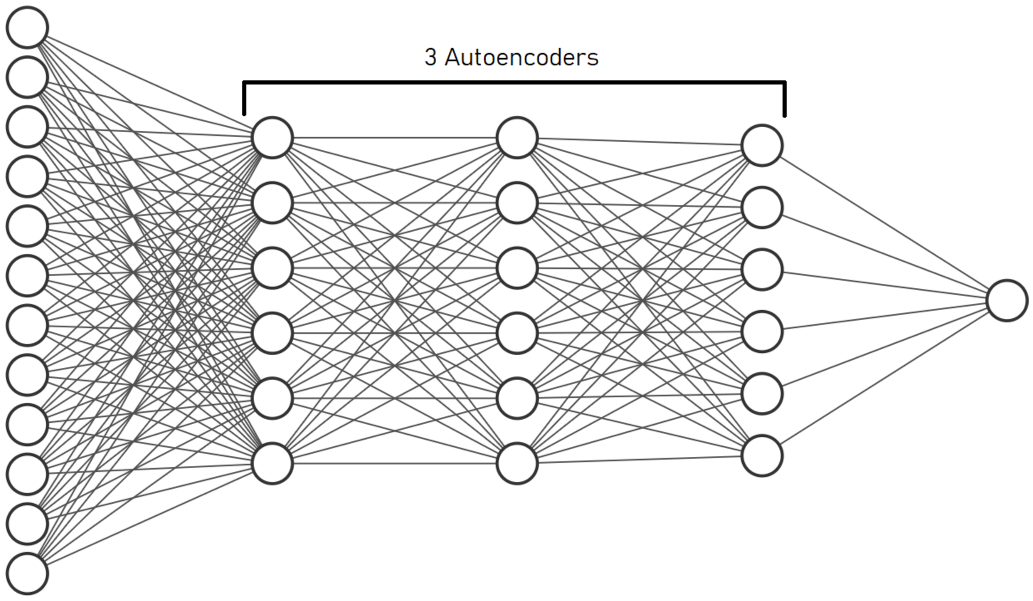


Figure 4: Stacked Autoencoder structure

An extension to the ordinary autoencoder is the so-called *denoising* autoencoder. In this type of autoencoder, the architecture of the network and computations are the same as before, however, the input data is corrupted. More specifically, some kind of 'noise' is added to the input data. The main motivation behind adding noise to the data is that one can expect the original data to be already 'polluted' in some way. Then, by training the network using the noisy input and non-noisy output, the weights and biases are tuned in such a way that the noise is filtered out and only a 'clean' latent representation is generated. There are several types of noise one can add. In this study we use masking noise, again, as it was found to perform best in previous literature. Masking noise can be thought of as randomly setting a certain percentage of the input data to zero. By doing this the network is forced to learn more robust features as well as that it is less likely to over-fit.

3.5 Recurrent Neural Networks

Besides regular architectures of neural networks such as a (stacked) autoencoder, more recently the use of recurrent neural networks (RNNs) has gained in popularity. A RNN is a type of deep neural network that allows information to persist within a layer over a longer period of time (as opposed to regular neural networks in which information only 'flows' forward from layer to layer). RNNs gain this temporal dimension by adding a hidden state in each node in which information is stored. Numerous specific architectures of RNNs exist, of which in this paper we apply two; Long-Short-Term-Memory (LSTM) and Gated Recurrent Unit (GRU). The main idea behind these two architectures is similar; instead of using all past information in computing the current output like regular RNN architectures, they introduce a memory cell in which only the most relevant information of the past inputs is filtered out and propagated to the next point in time $(t+1)$.

3.5.1 Long-Short-Term-Memory

While in general regular RNNs model time series well, they struggle with learning long term dependencies due to having a vanishing gradient. This vanishing gradient occurs because when the weights of a network are tuned using the estimation error, this error is also passed through all previous memory cells of the node to update the recurrent weights. However, the problem arises when the network is initialized, since then the initial weights are assigned as random numbers close to zero. Hence, when moving backward through the model updating the weights using the error (gradient of the loss) through some matrix multiplication, the error quickly 'vanishes' as it gets close to zero. This causes observations further back in time to not be as influential as might be optimal.

First introduced by Hochreiter and Schmidhuber (1997), LSTM overcomes this issue by using memory cells. These cells consist of different gates which regulates what information is conserved and propagated to the next period in time, as well as that it computes the output of the cell. In figure 5 the structure of such cell is shown including its main components. Firstly, x_t and h_t are the input and output of the cell, respectively, while C_t is the current cell state. Then, there is the so-called forget gate f_t which determines how much of the previous cell state is let trough to the current one. It can be formulated as

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{9}$$

where W and b are the weight and bias vectors within a sigma function as specified earlier. Then, i_t and $\tilde{C}_t$ are the values for the input gate and the candidate value for the current cell state as computed from the current input. These can be computed as

$$\begin{aligned} i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \\ \tilde{C}_t &= \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \end{aligned} \tag{10}$$

which are then combined and added to the current cell state C_t by computing

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{11}$$

where C_{t-1} is the previous cell state. Finally, to compute the values for cells output gate o_t and the cell output itself h_t one can apply the following formulas:

$$\begin{aligned} o_t &= \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \\ h_t &= o_t * \tanh\left(C_t\right) \end{aligned} \tag{12}$$

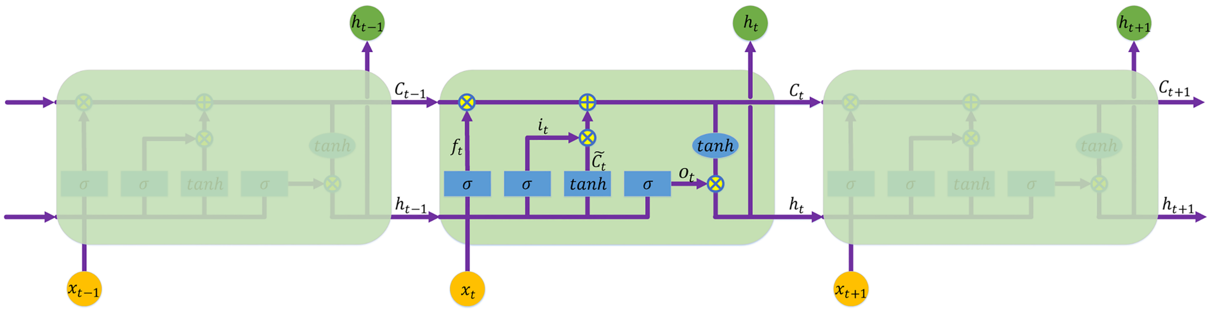


Figure 5: LSTM cell [1]

Overall, by including LSTM in this paper we ensure to have a model that can learn trends over a long period of time. Since we deal with time series data with over 60 years of observations, this may proof to be worthwhile.

3.5.2 Gated Recurrent Unit

A variation to LSTM introduced by Cho et al. (2014) is the Gated Recurrent Unit. GRU varies from LSTM in that it combines the input and forget gates into a single 'update gate'. Moreover, it does not differentiate between output and cell state, but instead merges them to generate a single value that is used as output as well as input for the cell state in $t + 1$.

Comparing GRU to LSTM, it is most often found that they yield comparable performance (Jozefowicz, Zaremba, and Sutskever (2015). However, GRU needs less parameters to be optimized compared to LSTM, and hence one may expect it to need less data to generalize (Chung et al. (2014). This may be helpful in our case since the amount of data used in this paper is not considered as a lot within machine learning world (1106 observations). Furthermore, Shen et al. (2018) found that GRU performed better than SVR in financial sequence predictions, and therefore it is worth investigating how it will perform in predicting momentum portfolios.

[1]Figure obtained from Bao, Yue, and Rao (2017)

3.6 Training Procedure

In order for fair comparison, all the aforementioned models will be trained using the same the same data and training/test split. Specifically, they will be trained to predict one month ahead using the past 12 months as input. The training will be done with the set of past 300 input output pairs, where the input is 12 previous months and the output is the next. In order to get most out of the machine learning techniques used in this paper, a search is done to find the optimal hyperparameters. Firstly, for SVR, the three main parameters to be tuned are C, ϵ, and γ, where C is the penalty parameter for the error, ϵ is the magnitude where within the error is not associated with the training loss, and γ is the parameter in the Gaussian kernel. The values used for the hyper parameter search can be found in Appendix A. As for the autoencoders, the optimal hyperparameters found by Kim (2019) are used. Finally, as for the recurrent neural networks, a limited amount of hyperparameters is tested. This is due to the expensive computation times for these architectures. Extensive preliminary testing was done however to find the most fitting hyperparameters to use in the a grid search. In the end, the following forecasting methods and strategies will be evaluated:

- *Mkt*: The market returns.
- *WML*: Winner-minus-loser strategy.
- *WO*: Winner-only strategy.
- *Look-back*: Look-back based predictions in combination with the selective-winner-minus-loser strategy.
- *SVR*: Predictions made using support vector regression in combination with the selective-winner-minus-loser strategy.
- *SdAE*: Predictions made using the stacked denoising autoencoder in combination with the selective-winner-minus-loser strategy.
- *LSTM*: Predictions made using long-term-short memory network in combination with the selective-winner-minus-loser strategy.
- *GRU*: Predictions made using gated recurrent unit in combination with the selective-winner-minus-loser strategy.

4 Results

Having generated the predictions and computed the returns according to the trading rules of the different strategies, the strategies are judged according to the following performance metrics:

(i) *Profitability:* Average, minimum and maximum return, and annual return.

(ii) *Risk:* Annual volatility and maximum drawdown (MDD).

(iii) *Performance ratios:* Sharpe ratio and the information ratio.

(iv) *Return distribution:* Kurtosis and skewness.

Having generated all the out-of-sample predictions and returns from 1953 to 2019, the performance metrics are computed net of transaction fees. This is done by adopting the method used by Barroso and Santa-Clara (2015) using a 75% turnover and a one way transaction costs of 0.2% and 0.1% from 1953 to 1997 and 1997-2019 respectively. Tables 1-3 show the metrics divided into different periods; the entire period from 1943 to 2019, 1953 to 1985, and 1985 to 2019, respectively. This division is made such that it reflects different economic conditions. Where between 1953 and 1985 the economic market was without much turmoil or big fluctuations, between 1985 and 2019 the economic situation was more volatile, including the 1987 stock market crash, the dot-com bubble, and the 2008 financial crisis.

Table 1 indicates that over the entire period WO and LSTM show best performance according to most metrics. Firstly, looking at profitability, average monthly return is equal, while LSTM performs only slightly better than WO according to annualized returns. As for the other strategies, we find that GRU and SVR perform worse by a small margin compared to WO and LSTM, whereas Look-back and the SdAE hardly outperform the market. Finally, WML clearly shows worse performance, even failing to achieve better returns than the market. Looking at tables 2-3, similar conclusions can be drawn. As expected, during the period of 1953-1985 the returns of the strategies were greater than that of the later period, mainly due to a relative smooth, and hence easier to predict, trend without many fluctuations. However, considering table 3, LSTM shows much better returns relative to the other models, only decreasing 12% in annualized return with respect to the previous period, compared to a decrease of 26% and 30% for WO and GRU respectively.

Inspecting the other metrics such as the risk adjusted return (Sharpe ratio), it can be noticed that LSTM outperforms all other strategies quite significantly, with the WO as well as GRU lagging behind due to having a relatively higher annualized volatility. Furthermore, the SdAE and SVR still outperform the market returns, while this is not the case for WML and the Look-back strategy. As for the distribution of the returns where low kurtosis and a small negative skewness is desirable, we see that WO and LSTM again show the best values, although the skewness of the LSTM is preferred over that of WO. It is also worth noticing the high kurtosis of the WML strategy, mainly caused by a large amount of large negative returns between 1985 and 2019, indicating much heavier tails.

The fact that LSTM performs best in more volatile periods, relative to the other strategies, is also supported by the fact that it has the lowest maximum drawdown, suggesting low downside risk and therefore whenever losses are made they are relatively lower than those of the other models. Compared to the baseline strategies, this holds for all strategies based on predictions except the SVR approach.

Table 1: Performance metrics plain momentum strategies 1953-2019

	Mkt	WML	WO	Look-back	SVR	SdAE	LSTM	GRU
Avg. Return	0.010	0.005	0.016*	0.010	0.014	0.011	0.016*	0.015
Max. Return	0.166	0.096	0.312	0.308	0.312	0.312	0.312	0.312
Min. Return	-0.226	-0.298	-0.328	-0.338	-0.262	-0.200	-0.262	-0.328
Ann. Return	0.120	0.063	0.214	0.130	0.176	0.144	0.215*	0.192
Ann. Vol	0.148	0.101*	0.217	0.202	0.197	0.176	0.188	0.200
Sharpe Ratio	0.813	0.622	0.984	0.643	0.894	0.820	1.139*	0.960
IR	NA	-0.081	0.187*	0.012	0.085	0.036	0.130	0.094
Kurtosis	1.923	20.432	3.017	3.938	3.270	3.392	3.021	4.745
Skewness	-0.513	-2.695	-0.565	-0.273	-0.335	0.097	-0.038	-0.560
MDD	0.504	0.525	0.602	0.499	0.625	0.485	0.406*	0.435

Notes: The best value per performance metric is indicated by a *

Table 2: Performance metrics plain momentum strategies 1953-1985

	Mkt	WML	WO	Look-back	SVR	SdAE	LSTM	GRU
Avg. Return	0.009	0.007	0.019*	0.013	0.016	0.012	0.017	0.017
Max. Return	0.166	0.089	0.227	0.223	0.230	0.227	0.230	0.230
Min. Return	-0.122	-0.137	-0.266	-0.270	-0.262	-0.180	-0.262	-0.262
Ann. Return	0.116	0.082	0.247*	0.173	0.214	0.155	0.228	0.227
Ann. Vol	0.143	0.080*	0.210	0.195	0.198	0.175	0.194	0.201
Sharpe Ratio	0.806	1.022	1.175	0.890	1.079	0.884	1.177*	1.129
IR	NA	-0.053	0.281*	0.070	0.174	0.060	0.178	0.174
Kurtosis	0.940	6.174	2.037	2.987	2.861	2.067	2.601	2.872
Skewness	-0.036	-1.314	-0.497	-0.433	-0.456	0.015	-0.369	-0.563
MDD	0.464	0.176*	0.469	0.397	0.390	0.485	0.406	0.435

Notes: The best value per performance metric is indicated by a *

Table 3: Performance metrics plain momentum strategies 1985-2019

	Mkt	WML	WO	Look-back	SVR	SdAE	LSTM	GRU
Avg. Return	0.010	0.004	0.014	0.007	0.011	0.011	0.015*	0.012
Max. Return	0.129	0.096	0.310	0.308	0.312	0.312	0.312	0.312
Min. Return	-0.226	-0.298	-0.333	-0.338	-0.220	-0.200	-0.165	-0.328
Ann. Return	0.124	0.045	0.183	0.091	0.141	0.134	0.202*	0.160
Ann. Vol	0.151	0.117*	0.223	0.208	0.194	0.177	0.183	0.198
Sharpe Ratio	0.820	0.387	0.821	0.437	0.724	0.761	1.104*	0.805
IR	NA	-0.104	0.112*	-0.036	0.023	0.014	0.097	0.042
Kurtosis	2.660	19.885	3.694	4.641	3.755	4.596	3.523	6.626
Skewness	-0.892	-2.896	-0.607	-0.134	-0.223	0.172	0.321	-0.562
MDD	0.504	0.525	0.602	0.499	0.625	0.286*	0.321	0.381

Notes: The best value per performance metric is indicated by a *

Figures 6-8 show the cumulative returns for aforementioned time periods. Again, it confirms claims made before, showing best cumulative returns for LSTM just before WO and GRU. Moreover, it is clear that the curves from the strategies based on predictions are much smoother overall, compared to the WML, WO, and Mkt. Additionally, as can be seen in figures 7 and 8, WO had highest cumulative returns in the first time period, however, mainly due great performance during the dot-com bubble and stable returns during the 2008 financial crisis, LSTM generated much higher returns during the early 2000's.

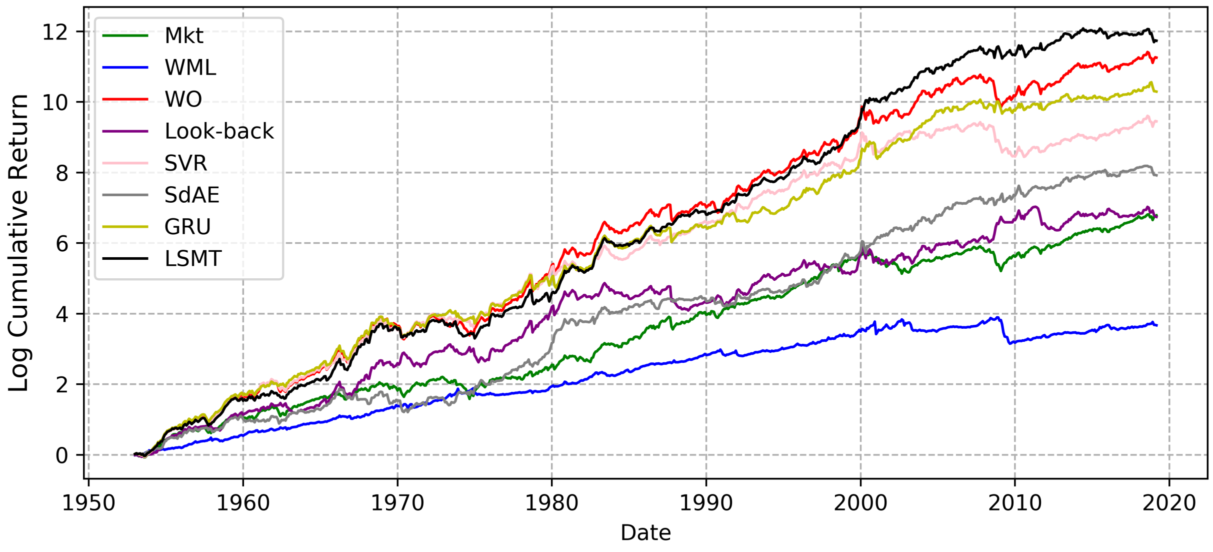


Figure 6: Cumulative returns 1953 - 2019

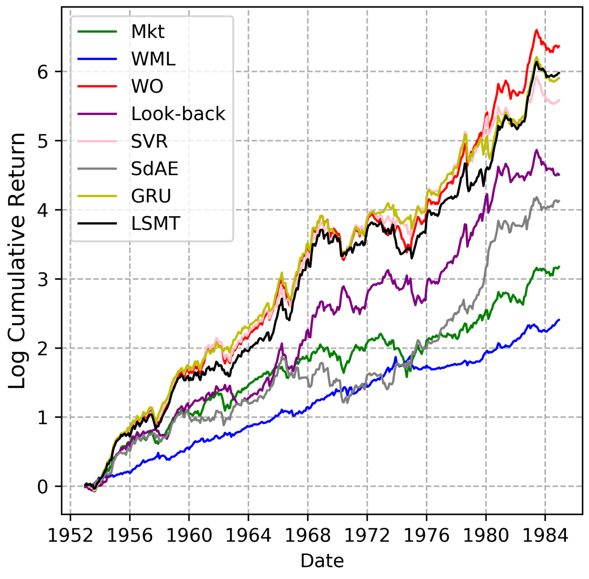


Figure 7: Cumulative returns 1953 - 1985

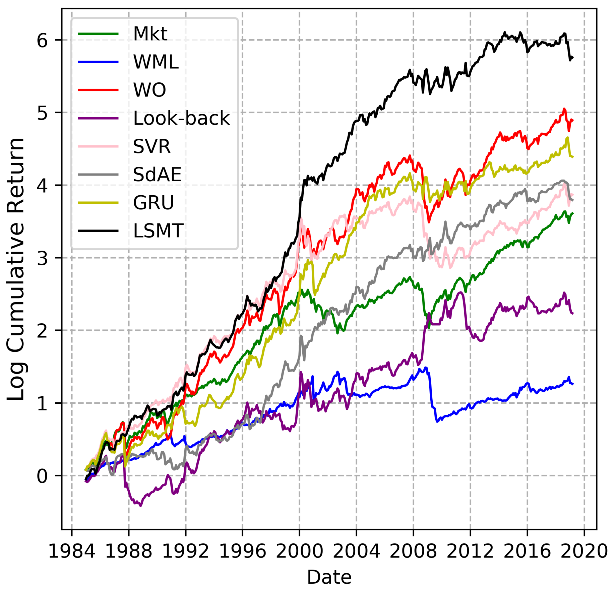


Figure 8: Cumulative returns 1985 - 2019

Tables 4 and 5 show the strategy returns for the volatility scaled returns for a volatility target of 12% (from Moskowitz, Ooi, and Pedersen (2012)) and 15% (from Lim, Zohren, and Roberts

(2019)) respectively. Looking at table 4, the main difference with the non-scaled returns is that LSTM performs much worse relative to the other strategies. For example, its annualized return is 35% lower, while WO only decreased with 2% and SVR even improved by 35%. Moreover, looking at the Sharpe Ratio as well as the MDD, WML greatly improved over the plain returns, outperforming all other models. Considering the great increase in annualized return of SVR and a relative decrease thereof with the LSTM, it can again be concluded that SVR performs much better in months which have relative low volatility since those it is re-scaled to be bigger. On the other hand, LSTM performs worse having a 12% target, implying it makes its biggest profits in times of high volatility.

As can be seen in table 5, setting the volatility target to 15% majorly changes the results again. In this case, WO and GRU perform best with both having an average monthly return of 1.9%. Moreover, GRU is now the best performer when looking at the Sharpe ratio. However, when inspecting the distribution metrics, it can be seen that all strategies, with exception of SdAE, display less desirable values.

Overall, the main conclusion to be drawn from the volatility scaled returns is that WO performs best according to profitability, while GRU, SVR and WML show more desirable properties when considering the risk adjusted returns. Moreover, unlike with the plain returns, LSTM is never the best performing strategy for any performance metric. From this it can be concluded that LSTM benefits most in times of high volatility, GRU and SVR both perform best whenever the volatility is low (since then the returns are scaled up), and WO is the most constant performer. As for the different periods of the volatility scaled returns, Appendix B contains the tables for both sub-periods and volatility targets, mostly containing expected results with, for example, all returns being lower in the later sub-period due to the more volatile returns.

Table 4: Performance measures volatility scaled returns (12% target) 1953-2019

	Mkt	WML	WO	Lookback	SVR	SdAE	LSTM	GRU
Avg. Return	0.009	0.007	0.016*	0.010	0.015	0.011	0.014	0.014
Max Return	0.166	0.083	0.227	0.193	0.227	0.194	0.193	0.194
Min Return	-0.226	-0.129	-0.407	-0.413	-0.299	-0.227	-0.413	-0.407
Ann. Return	0.120	0.082	0.209*	0.133	0.191	0.136	0.179	0.186
Ann. Vol	0.148	0.071*	0.196	0.176	0.171	0.154	0.184	0.185
Sharpe Ratio	0.813	1.151*	1.068	0.755	1.113	0.880	0.968	1.005
IR	NA	-0.059	0.185*	0.017	0.118	0.025	0.108	0.121
Kurtosis	1.923	5.438	5.679	8.209	3.332	3.085	7.496	7.109
Skewness	-0.513	-0.938	-0.848	-0.941	-0.185	-0.091	-1.081	-1.005
MDD	0.504	0.165*	0.423	0.547	0.354	0.472	0.465	0.455

Notes: The best value per performance metric is indicated by a *

Table 5: Performance measures volatility scaled returns (15% target) 1953-2019

	Mkt	WML	WO	Look-back	SVR	SdAE	LSTM	GRU
Mean	0.009	0.008	0.019*	0.013	0.018	0.013	0.017	0.019*
Max	0.166	0.100	0.273	0.233	0.273	0.233	0.233	0.273
Min	-0.226	-0.155	-0.489	-0.489	-0.358	-0.273	-0.489	-0.358
Ann. Return	0.120	0.099	0.255*	0.166	0.233	0.165	0.226	0.251
Ann. Vol	0.148	0.085*	0.235	0.212	0.206	0.185	0.222	0.198
Sharpe Ratio	0.813	1.160	1.087	0.782	1.132	0.890	1.022	1.268*
IR	NA	-0.031	0.223*	0.052	0.161	0.062	0.163	0.174
Kurtosis	1.923	5.438	5.679	7.725	3.332	3.085	7.109	3.737
Skewness	-0.513	-0.938	-0.848	-0.870	-0.185	-0.091	-1.005	-0.192
MDD	0.504	0.199*	0.498	0.626	0.419	0.543	0.410	0.541

Notes: The best value per performance metric is indicated by a *

Besides directly judging the performance of the models according to their final return after trading rules have been applied, it might also be interesting to compare the forecasting models and their forecasting performance and if it directly relates to the returns of the strategy. In table 6 two forecasting evaluation metrics are shown. Firstly, the root mean squared error is computed, which is a measure of accuracy formulated as RMSE $= \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}}$, where y_t is the actual return at time t and $\hat{y}_t$ the predicted value for time t. Right away it is clear that Look-back is the main anomaly when it comes to the accuracy of its predictions. This is expected as it is designed not to accurately predict returns, but rather its main purpose is to give an expectation of the sign of the return. As for the other models, we see that their accuracy is rather similar, with LSTM performing best, but only differing from GRU and SVR by 0.0001 and 0.0004 respectively. Secondly, the hit rate was computed for the forecasting models. The hit-rate is the fraction of correctly predicted signs of the returns and hence essential for these momentum strategies as they fully base the long/short decision on the sign of the prediction. Unlike for RMSE, Look-back performs much more reasonable when considering hit-rate, correctly predicting about 57% of the signs. However, as is consistent with RMSE, LSTM shows best hit-rate by far, predicting almost 61% of the signs of the returns correctly. GRU and SVR also perform relatively well with a hit-rate of almost 59%, while the SdAE shows a rather poor hit-rate of 56%. Comparing the forecasting performance metrics with those of the strategy returns, we see that mostly the better the model scores for hit-rate, the better its final returns are, with only inconsistency being that SdAE showed worse forecasting performance compared to Look-back, but performed better with respect to annualized return and Sharpe ratio.

Table 6: Forecasting and return performance metrics

	RMSE	Hit-rate	Annualized Return	Sharpe Ratio
Look-back	0.3903	0.572	0.130	0.643
SVR	0.0739	0.585	0.176	0.894
SdAE	0.0753	0.561	0.144	0.820
LSTM	0.0735	0.607	0.215	1.139
GRU	0.0736	0.588	0.192	0.960

5 Conclusion

In this paper we investigated the performance of several predictive models in combination with the selective winner-minus-loser strategy. We compared their performance to that of classical momentum strategies. Firstly, considering the plain, non-scaled, returns, it was found that the best performing strategies were WO and LSTM, both yielding an average monthly return of 1.6%. However, LSTM would be preferred over WO in most situations as it demonstrates a 15% improvement in risk adjusted return, a lower maximum drawdown, as well as exhibiting a more desirable distribution of its returns. As for the other models, they are all shown to have inferior performance compared to WO and LSTM. For the different sub-periods WO performs exceptionally well in times of relatively calm economic conditions, while LSTM is superior in more recent times during several market crashes and bubbles.

This view changes when considering the volatility scaled returns, for which LSTM performs worse compared to most other strategies. Still, WO is the best performing strategy according to profitability (together with GRU for a volatility target of 15%), but WML outperforms it according to Sharpe ratio and maximum drawdown. It is also worth noticing that, in contrast to the conclusions drawn by Kim (2019), we found SdAE to not outperform WO and SVR. This may be due to the randomness involved in the training of neural networks and/or the slightly different specification of the model.

As for our main objective of identifying whether recurrent neural networks can be used for the prediction of momentum portfolios, we have shown that this is certainly a viable and effective approach. However, it must be taken into account that a limited search range for the hyper-parameters was done for the deep learning methods. Hence, there is still improvement to be made which would most certainly cause them to outperform the classical momentum strategies. Moreover, under specific circumstances using the SWML strategy nothing is invested and one could invest in risk free bonds during this month. This is not considered during this paper but would increase overall profits.

As for future research, we suggest altering the several deep learning methods such that they directly generate the trading position by changing the loss function. In this way, one could directly maximize the returns or Sharpe ratio, or minimize the maximum drawdown depending on one's main objective. Additionally, instead of fully relying on the sign of the predictions in deciding the trading positions, one could propose a strategy in which the magnitude of the prediction is taken into account.

References

Bao, Wei, Jun Yue, and Yulei Rao (2017). “A deep learning framework for financial time series using stacked autoencoders and long-short term memory”. In: *PLOS ONE* 12.7, pp. 1–24. DOI: 10.1371/journal.pone.0180944.

Barroso, Pedro and Pedro Santa-Clara (2015). “Momentum has its moments”. In: *Journal of Financial Economics* 116.1, pp. 111 –120. ISSN: 0304-405X. DOI: https://doi.org/10.1016/j.jfineco.2014.11.010.

Basak, Debasish, Srimanta Pal, and Dipak Chandra Patranabis (2007). “Support vector regression”. In: *Neural Information Processing-Letters and Reviews* 11.10, pp. 203–224.

Baz, Jamil et al. (2015). “Dissecting Investment Strategies in the Cross Section and Time Series”. In: *SSRN Electronic Journal*. DOI: 10.2139/ssrn.2695101.

Cho, Kyunghyun et al. (2014). “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *CoRR* abs/1406.1078. arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

Chun-Hsin Wu, Jan-Ming Ho, and D. T. Lee (2004). “Travel-time prediction with support vector regression”. In: *IEEE Transactions on Intelligent Transportation Systems* 5.4, pp. 276–281. ISSN: 1524-9050. DOI: 10.1109/TITS.2004.837813.

Chung, Junyoung et al. (2014). “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In: *CoRR* abs/1412.3555. arXiv: 1412.3555.

Clarke, Stella M., Timothy William Simpson, and Jan H. Griebsch (2003). “Analysis of support vector regression for approximation of complex engineering analyses”. English (US). In: *Proceedings of the 2003 ASME Design Engineering Technical Conferences and Computers in Information in Engineering Conference. Volume 2: 29th Design Automation Conference*. Vol. 2 A, pp. 535–543.

Daniel, Kent and Tobias J. Moskowitz (2016). “Momentum crashes”. In: *Journal of Financial Economics* 122.2, pp. 221 –247. ISSN: 0304-405X. DOI: https://doi.org/10.1016/j.jfineco.2015.12.002.

Drucker, Harris et al. (1996). “Support Vector Regression Machines”. In: *Proceedings of the 9th International Conference on Neural Information Processing Systems*. NIPS’96. Denver, Colorado: MIT Press, pp. 155–161. URL: http://dl.acm.org/citation.cfm?id=2998981.2999003.

Dunis, Christian L. and Xuehuan Huang (2002). “Forecasting and trading currency volatility: an application of recurrent neural regression and model combination”. In: *Journal of Forecasting* 21.5, pp. 317–354. DOI: 10.1002/for.833.

French (2019). *Sorts involving Prior Returns.* Available online at: http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html.

Heaton, J. B., Nicholas G. Polson, and J. H. Witte (2016). “Deep Learning in Finance”. In: *CoRR* abs/1602.06561. arXiv: 1602.06561.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long Short-term Memory”. In: *Neural computation* 9, pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

Hurst, Brian, Yao Hua Ooi, and Lasse Heje Pedersen (2017). “A Century of Evidence on Trend-Following Investing”. In: *The Journal of Portfolio Management* 44.1, pp. 15–29. ISSN: 0095-4918. DOI: 10.3905/jpm.2017.44.1.015.

Jegadeesh, Narasimhan and Sheridan Titman (1993). “Returns to Buying Winners and Selling Losers: Implications for Stock Market Efficiency”. In: *The Journal of Finance* 48.1, pp. 65–91. DOI: 10.1111/j.1540-6261.1993.tb04702.x.

— (1998, 2001). “Profitability of momentum strategies: An evaluation of alternative explanations”. In: *The Journal of finance* 56.2, pp. 699–720.

— (2001). “Profitability of Momentum Strategies: An Evaluation of Alternative Explanations”. In: *The Journal of Finance* 56.2, pp. 699–720. DOI: 10.1111/0022-1082.00342.

Jozefowicz, Rafal, Wojciech Zaremba, and Ilya Sutskever (2015). “An Empirical Exploration of Recurrent Network Architectures”. In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France: JMLR.org, pp. 2342–2350. URL: http://dl.acm.org/citation.cfm?id=3045118.3045367.

Ke, Jintao et al. (2017). “Short-term forecasting of passenger demand under on-demand ride services: A spatio-temporal deep learning approach”. In: *Transportation Research Part C: Emerging Technologies* 85, pp. 591 –608. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2017.10.016.

Kim, Saejoon (2019). “Enhancing the momentum strategy through deep regression”. In: *Quantitative Finance* 19.7, pp. 1121–1133. DOI: 10.1080/14697688.2018.1563707.

Lemperiere, Y. et al. (2014). “Two centuries of trend following”. In: 1404.3274. URL: https://ideas.repec.org/p/arx/papers/1404.3274.html.

Lim, Bryan, Stefan Zohren, and Stephen Roberts (2019). “Enhancing Time Series Momentum Strategies Using Deep Neural Networks”. In: *arXiv preprint arXiv:1904.04912*.

Miotto, Riccardo et al. (2017). “Deep learning for healthcare: review, opportunities and challenges”. In: *Briefings in Bioinformatics* 19.6, pp. 1236–1246. ISSN: 1477-4054. DOI: 10.1093/bib/bbx044.

Moskowitz, Tobias J., Yao Hua Ooi, and Lasse Heje Pedersen (2012). “Time series momentum”. In: *Journal of Financial Economics* 104.2. Special Issue on Investor Sentiment, pp. 228 –250. ISSN: 0304-405X. DOI: https://doi.org/10.1016/j.jfineco.2011.11.003.

Rather, Akhter Mohiuddin, Arun Agarwal, and V.N. Sastry (2015). “Recurrent neural network and a hybrid model for prediction of stock returns”. In: *Expert Systems with Applications* 42.6, pp. 3234 –3241. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2014.12.003.

Shen, Guizhu et al. (2018). “Deep Learning with Gated Recurrent Unit Networks for Financial Sequence Predictions”. In: *Procedia Computer Science* 131. Recent Advancement in Information and Communication Technology: pp. 895 –903. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.04.298.

Tay, Francis E.H and Lijuan Cao (2001). “Application of support vector machines in financial time series forecasting”. In: *Omega* 29.4, pp. 309 –317. ISSN: 0305-0483. DOI: https://doi.org/10.1016/S0305-0483(01)00026-3.

Zhang, G.Peter (2003). “Time series forecasting using a hybrid ARIMA and neural network model”. In: *Neurocomputing* 50, pp. 159 –175. ISSN: 0925-2312. DOI: https://doi.org/10.1016/S0925-2312(01)00702-0.

Appendices

A Hyperparameter search range

Model	Hyperparameter	Search Range
	C	$\{10^i, 5 \cdot 10^i : -2 \leq i \leq 3\}$
SVR	ϵ	$\{10^i, 5 \cdot 10^i : -5 \leq i \leq -1\}$
	γ	$\{10^i, 5 \cdot 10^i : -5 \leq i \leq -1\}$
	Learn Rate	(0.1, 0.25)
RNN	Dropout	([12, 6, 1], [12, 6, 6, 1], [12, 6, 4, 1])
	Structure	(0.05, 0.005, 0.0005)

B Results volatility scaled returns sub-periods

Table 7: Performance measures volatility scaled returns (12% target) 1953-1985

	Mkt	WML	WO	Look-back	SVR	SdAE	LSTM	GRU
Mean	0.009	0.007	0.018	0.013	0.016	0.012	0.017	0.016
Max	0.166	0.060	0.227	0.175	0.227	0.152	0.178	0.175
Min	-0.122	-0.100	-0.299	-0.303	-0.299	-0.227	-0.299	-0.303
Ann. Return	0.116	0.087	0.244	0.174	0.213	0.149	0.222	0.212
Ann. Vol	0.143	0.069	0.201	0.185	0.191	0.168	0.192	0.192
Sharpe Ratio	0.806	1.263	1.213	0.940	1.114	0.890	1.153	1.106
IR	NA	-0.046	0.273	0.073	0.175	0.052	0.202	0.186
Kurtosis	0.940	2.712	2.699	3.667	3.636	2.818	2.999	3.178
Skewness	-0.036	-0.762	-0.504	-0.538	-0.502	-0.316	-0.676	-0.734
MDD	0.464	0.130	0.402	0.363	0.354	0.472	0.401	0.393

Table 8: Performance measures volatility scaled returns (12% target) 1985-2019

	Mkt	WML	WO	Look-back	SVR	SdAE	LSTM	GRU
Mean	0.010	0.006	0.014	0.008	0.013	0.010	0.012	0.012
Max	0.129	0.083	0.194	0.193	0.194	0.194	0.194	0.193
Min	-0.226	-0.129	-0.407	-0.413	-0.126	-0.165	-0.407	-0.413
Ann. Return	0.124	0.077	0.177	0.097	0.171	0.123	0.153	0.148
Ann. Vol	0.151	0.073	0.190	0.167	0.151	0.140	0.177	0.177
Sharpe Ratio	0.820	1.056	0.931	0.577	1.132	0.878	0.865	0.836
IR	NA	-0.070	0.109	-0.037	0.073	-0.002	0.052	0.043
Kurtosis	2.660	7.292	9.020	14.187	1.731	3.120	12.275	12.898
Skewness	-0.892	-1.065	-1.243	-1.486	0.360	0.243	-1.427	-1.520
MDD	0.504	0.165	0.423	0.547	0.315	0.263	0.465	0.455

Table 9: Performance measures volatility scaled returns (15% target) 1953-1985

	Mkt	WML	WO	Look-back	SVR	SdAE	LSTM	GRU
Mean	0.009	0.008	0.023	0.016	0.019	0.014	0.020	0.020
Max	0.166	0.073	0.277	0.213	0.273	0.183	0.213	0.273
Min	-0.122	-0.122	-0.353	-0.358	-0.358	-0.273	-0.358	-0.358
Ann. Return	0.116	0.104	0.311	0.217	0.260	0.182	0.271	0.270
Ann. Vol	0.143	0.083	0.242	0.223	0.229	0.201	0.231	0.219
Sharpe Ratio	0.806	1.261	1.289	0.974	1.135	0.901	1.174	1.229
IR	NA	-0.017	0.320	0.110	0.213	0.088	0.240	0.217
Kurtosis	0.940	2.826	2.566	3.440	3.636	2.818	2.999	4.248
Skewness	-0.036	-0.783	-0.449	-0.487	-0.502	-0.316	-0.676	-0.531
MDD	0.464	0.157	0.457	0.433	0.419	0.543	0.410	0.455

Table 10: Performance measures volatility scaled returns (15% target) 1985-2019

	Mkt	WML	WO	Look-back	SVR	SdAE	LSTM	GRU
Mean	0.010	0.007	0.017	0.009	0.016	0.012	0.014	0.018
Max	0.129	0.098	0.235	0.233	0.233	0.233	0.233	0.233
Min	-0.226	-0.157	-0.482	-0.489	-0.151	-0.198	-0.489	-0.134
Ann. Return	0.124	0.093	0.224	0.120	0.208	0.149	0.186	0.233
Ann. Vol	0.151	0.088	0.228	0.201	0.181	0.168	0.212	0.175
Sharpe Ratio	0.820	1.054	0.980	0.597	1.148	0.887	0.876	1.332
IR	NA	-0.044	0.165	-0.005	0.116	0.037	0.095	0.138
Kurtosis	2.660	7.493	8.566	13.421	1.731	3.120	12.275	1.634
Skewness	-0.892	-1.110	-1.167	-1.397	0.360	0.243	-1.427	0.404
MDD	0.504	0.199	0.498	0.626	0.368	0.312	0.246	0.541

C Cumulative return volatility scaled returns

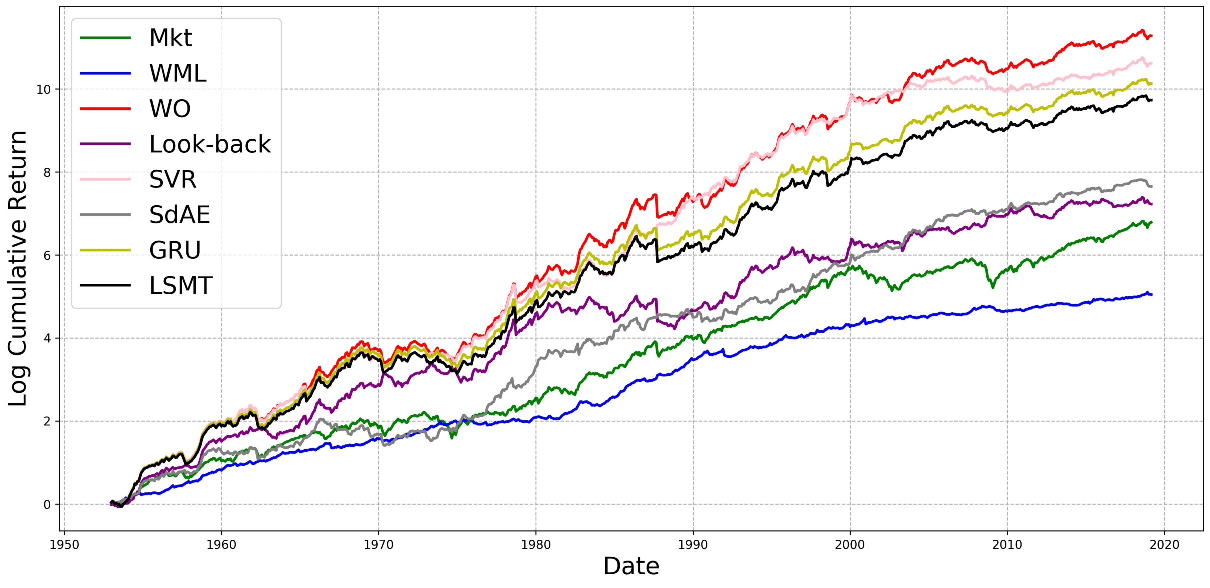


Figure 9: A figure

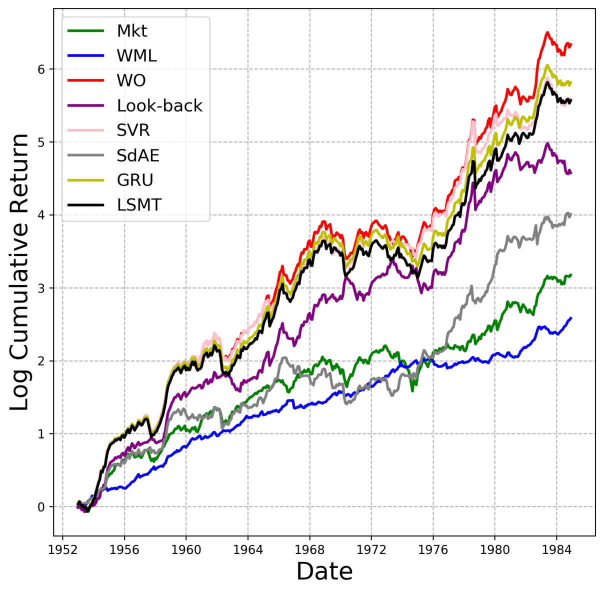


Figure 10: A figure

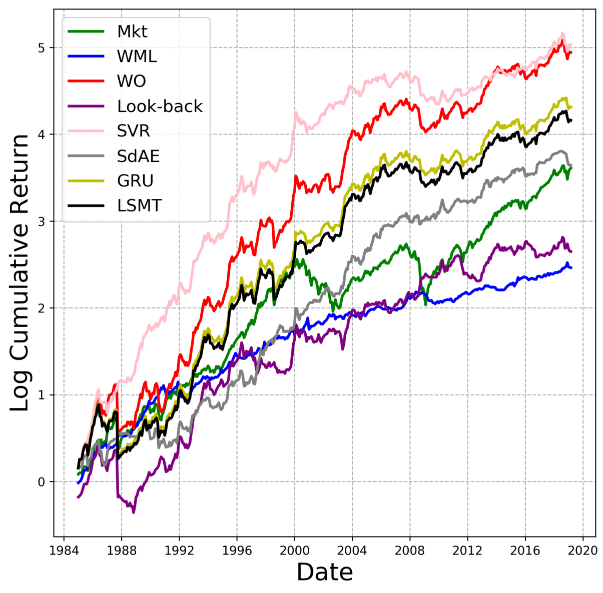


Figure 11: Another figure

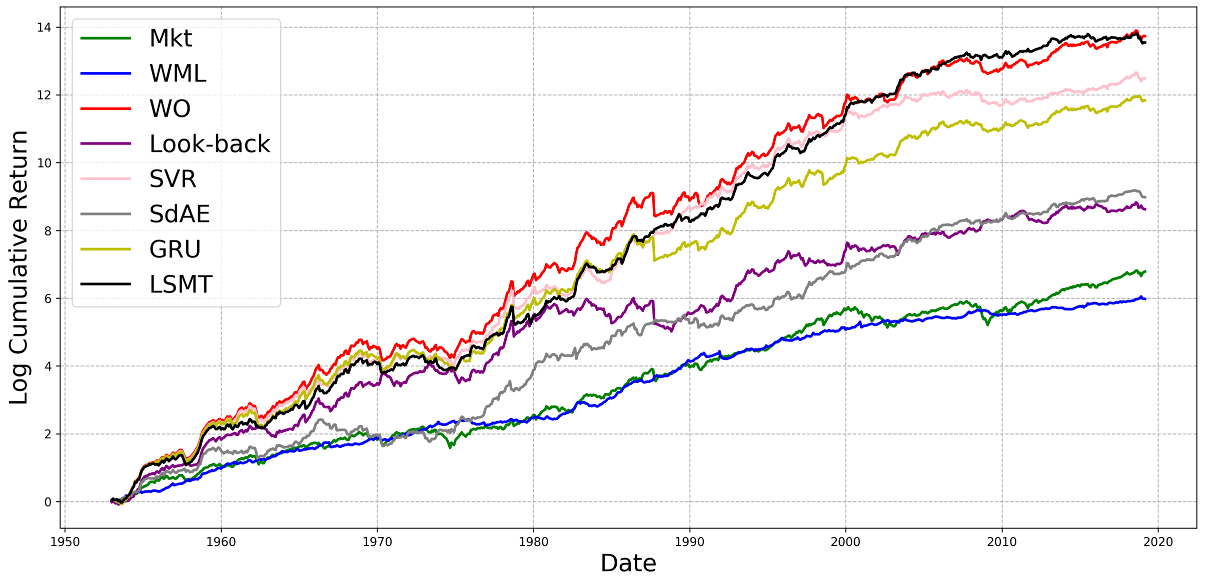


Figure 12: A figure

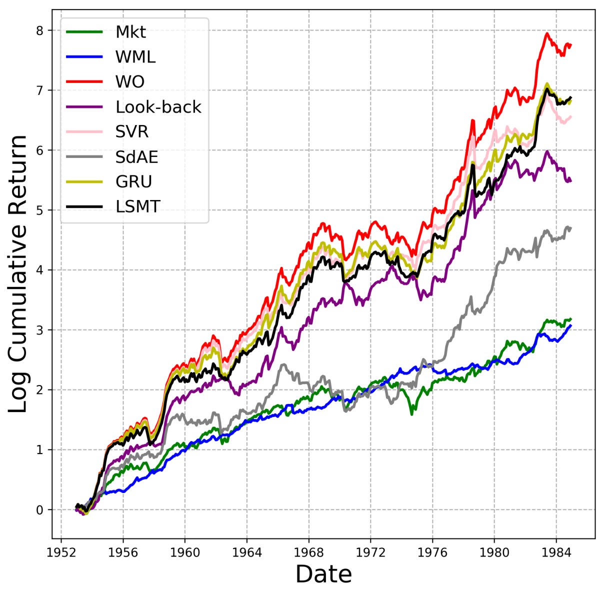


Figure 13: A figure

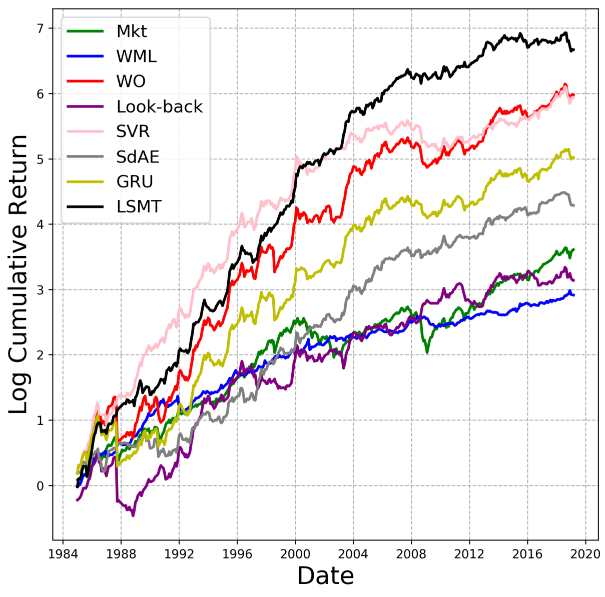


Figure 14: Another figure

D Code

The following files have been used to generate and evaluate the results:

Look-back.ipynb: Generates the results for the look-back based approach.

SVR.ipynb: Generates the results for the support vector regression approach.

SdAE.ipynb: Generates the results for the stacked denoising autoencoder approach.

GRU.ipynb: Generates the results for the gated recurrent unit approach.

LSTM.ipynb: Generates the results for the long-short-term-memory approach.

Returns.ipynb: Computes strategies return from the predictions

Pfm.ipynb: Computes performance measures and makes plots