Exploring Possible Improvements to Momentum Strategies with Deep Learning

Bachelor Thesis Econometrics and Operations Research

Adam Takács (456920)

Supervisor: S.H.L.C.G. Vermeulen

Second assessor: X. Xiao

Final version date: July 7, 2019

Abstract

Momentum is a well-known phenomenon that pervades in all major stock markets. Strategies that exploit this phenomenon have been documented to generate abnormal returns persistently over time. In this paper, we study these portfolio designs and explore possible improvements, employing machine-learning and deep-learning models in the return prediction. To forecast the returns of the best- and worst-performing stocks, we use support vector regression and the latest deep-learning techniques such as stacked (denoising) autoencoders and state-of-the-art generative adversarial networks. Our analysis focuses on the comparison of financial performance measures, such as average return, Sharpe ratio and maximum drawdown, between the simple momentum strategies and the advanced deep-learning models. The results confirm our expectations, namely that the complex models can improve momentum portfolios, mainly by avoiding large losses in times of crises. For generative models, the reduction of 50% in the maximum drawdown is achieved, compared to a basic momentum strategy. Furthermore, our results indicate a slight fading of the negative momentum, such that the stocks losing value in the past cannot be easily exploited by complex models to generate abnormal returns.

Contents

1 Introduction

In the world of investment and finance, as well as in the academic circles, momentum is a widely known and well-documented phenomenon. Fama and French (1996) have shown that publicly traded stocks that have recently generated positive returns are more likely to produce a positive return again in the near future. Similarly, there is a higher chance that the stocks that have been losing their value are going to continue following the same trend. The research done by Fama and French (1996) also lays out the basics of the momentum investment strategy. They have shown that following their strategy, an investor can generate returns that are in excess of the market return, without proportionally increasing their risk. Surprisingly, recent research into momentum portfolios by Kim (2019) shows that, even though the momentum phenomenon has become well-known among investors, the strategy is still able to persistently generate extraordinary returns. Even though it seems to contradict the efficient market hypothesis, Subrahmanyam (2018) believes that it can be explained either by behavioral over- and under-reaction theories or that the excess return is simply a compensation for taking greater risk. The latter explanation is valid if we assume that a stock's volatility increases with its market price. For this assumption to be generally accepted, however, more research has to be done, so in our paper we assume that stock's momentum is exploitable and is able to generate returns disproportional to the associated risk. In this paper we employ machine- and deep-learning models, with a focus on generative neural networks, that take advantage of this phenomenon.

There are many different momentum strategies, but they share a main idea which is straightforward and fairly easy to implement; buy the stocks that have performed extraordinarily well in the past year and sell the ones that have performed rather poorly. Jegadeesh and Titman (1993) buy stocks in the top decile of the US stock market, sorted by their past performance, while selling the bottom decile stocks. An alternative version, also proposed by Jegadeesh and Titman (2001), builds the portfolio by only buying the top decile of best performing stocks. In the rest of this paper, we will refer to these strategies as WML (winner-minus-loser) and WO (winner-only), respectively. There are advantages and disadvantages to both approaches. First of all, during times of expansion, WO strategy does not perform so well compared to WML, since it does not exploit a part of the momentum phenomenon, i.e. that past losers usually generate negative return in the near future. However, whereas WML performs better than WO in general, Moskowitz, Ooi, and Pedersen (2012) shows that its return during the periods where markets rebound after a crash are extremely bad. The two worst WML returns were in 1932 with a two-month return of -91.59% and in 2009 with a three-month return of -73.42%, showing

that WML suffers harsher reversals than WO strategy. To avoid these large negative losses of the momentum portfolio, Barroso and Santa-Clara (2015) devised a method that can be applied on top of the WO and WML trading strategies. It is achieved by scaling the portfolio weights (for both the top and bottom decile) based on the inverse of their past six months' realized volatility. This has proved to largely eliminate the extreme declines in value corresponding to the market rebounds (e.g. in August 1932 and in April/May 2009). Hence, in this paper we will use volatility-scaled returns together with the non-scaled returns to analyze our models.

A more sophisticated strategy was recently proposed by Kim (2019), which is a combination of the WO and WML strategies. The so-called selective winner-minus-loser strategy (SWML) places an additional constraint on the top and bottom decile portfolios, namely that the predicted return of the top decile must be positive, whereas the predicted return of the bottom decile must be negative, otherwise no trade is done. This way, the momentum portfolio return cannot be negative if the sign of the predicted decile return is correct. It can also be shown that both WML and WO strategies are special cases of the SWML strategy, hence they must never produce a higher return, in case our forecasts are correct. Consequently, to decide whether to buy the top decile and/or sell the bottom decile, a prediction of the return in the next period has to be made. The classical momentum strategy (as in Fama and French (1996) and Jegadeesh and Titman (1993)) uses the so-called look-back period. This method takes the cumulative return of the recent past and uses it as a forecast for the next periods return. However, in this paper we try to replace this simple prediction step with various complex models.

Purely econometric models have always been used in time-series forecasting, but nowadays investors and scientists put more focus on the application of various machine-learning and deep-learning models in predicting the future financial variables. Recent research by Kim (2019) introduces some state-of-the art deep learning models for momentum portfolio return prediction and also utilizes some simpler machine learning models, such as support vector regression. The latter has been commonly used and shown to be particularly good at predicting stock returns, volatility and prices, among many others by Law and Shawe-Taylor (2017) and I. Sapankevych and Sankar (2009). The field of deep learning is currently very dynamic and there are many recent studies showing the significant forecasting improvements, for example in option pricing, Berner, Grohs, and Jentzen (2018) and high-frequency trading, Dixon, Polson, and Sokolov (2018). Moreover, Bao, Yue, and Rao (2017) show that a certain type of deep neural networks, namely stacked autoencoders generate great results in financial time series forecasting. Therefore, it is of our interest to investigate the possible improvement in forecasting introduced by these models. Specifically, we will use the support vector regression and two types of stacked

autoencoders. To make our research novel we decided to also apply another deep-learning model, which will be described in more detail below.

In general, there are two main types of machine-learning models; discriminative and generative. The former one learns the conditional probability $P(y|x)$ of the target variable y from data x, whereas the latter learns the joint distribution $P(x, y)$ of the input and target data. Empirically, the discriminative models perform better for classification and regression purposes (the most famous application being written digits classification by LeCun et al. (1998)), however the generative models can be advantageous in smaller datasets (they are less prone to over-fitting) also being able to learn richer representation of the joint data, with a tradeoff that they are computationally heavier to train. However, with the recent technological developments in computing power, more studies have been done on generative models and their applications. One of the widely used generative models is the Generative Adversarial Network (GAN). The model has been invented by Goodfellow et al. (2014) and multiple improvements have been made, among others by Mirza and Osindero (2014) in terms of conditional GANs. Although GAN has its main purpose in fields other than time-series forecasting (e.g. in text generation, Dai et al. (2017) and in image-to-image mapping, Isola et al. (2017)), recent study done by Esteban, Hyland, and Rätsch (2017) has shown that they can be used to forecast medical time series variables. In this paper, we aim to investigate the possible forecasting improvements by GAN when applied in portfolio-building strategies. Hence, our main research question is:

Does the application of generative adversarial networks in time-series prediction enhance the performance of momentum portfolios?

To answer our main research question, we build various types of GAN, evaluate their performance and compare them to the performance of the classical look-back period, support vector regression and the two types of stacked autoencoders similar to those used in Kim (2019). Apart from the main research question, it is of our interest to find an economic interpretation of GAN, so that we can motivate its application in the context of portfolio construction.

The layout of this paper is as follows: In section 2 the process of obtaining our data is shown, section 3 explains all the models that we use in greater detail, with a subsection 3.5 dedicated specifically to generative adversarial networks and their economic interpretation. Section 4 shows how each of the models performs economically and statistically, and in section 5, we answer the research question based on performance of our models and suggest areas for further research.

2 Data

Since our paper builds on the results of Kim (2019), we make use of the same dataset. The data is taken from Kenneth French's Data Library (French (2019)), starting in January 1927 until March 2019. In this dataset, the daily and monthly returns of all deciles of the momentum portfolios are provided. These decile portfolios were constructed in a following way; first, all stocks traded on NYSE, AMEX and NASDAQ are ordered based on the performance over the past 12 months (excluding the last month). Then, the stock returns are grouped based on NYSE prior breakpoints (also taken from French (2019)) calculated as the deciles of the cumulative returns from the previous 12 months, excluding the last month. Based on this split the momentum portfolio returns are calculated. In this paper we only work with the top and the bottom decile, although we note that including other deciles (namely the second and ninth) might be worth exploring.

To obtain the volatility-scaled returns as suggested in Barroso and Santa-Clara (2015), we use the same procedure as Kim (2019), where the variance of the portfolio is calculated from the daily returns of the past 6 months. The portfolio weights, and hence returns, are then inversely related to these variances in a following manner:

$$r'_t = \frac{\sigma_{target}}{\hat{\sigma}_t} * r_t, \tag{1}$$

where r'_t denotes the volatility-scaled return, r_t is the original return, $\hat{\sigma}_t$ is the variance (estimated from the daily variance of the past six months) and σ_{target} a constant target volatility, set to the value corresponding to annualized volatility of 12%. Scaling the returns in this way means that in periods of large volatility in the markets, the portfolio exposure becomes smaller, whereas if the markets are stable the exposure is increased.

Moreover, to compare all portfolio performances we choose the excess market return to be the benchmark. The monthly returns for this portfolio are also acquired from Kenneth French's Data Library (French (2019)). After obtaining these returns the data is ready to be used in our machine- and deep-learning models. Tables with descriptive statistics of our data can be found in the appendix.

3 Methodology

In the selective WML strategy, after forecasting the return of the current month t, a long position is entered for the top decile portfolio if $r_t \geq 0$, otherwise no trade is done. Similarly, for the bottom decile portfolio, if $r_t \leq 0$ a short position is entered, otherwise nothing is traded. Hence, in the rest of this section we give a detailed account of all the methods used to forecast the returns for both the top and the bottom decile portfolio. We start by describing the support vector regression, then moving on to basic, and denoising stacked autoencoders, and finally the multi-layer perceptron and recurrent GAN. The economic interpretation of GAN in the context of return prediction concludes this section. Lastly, the metrics that are used to compare the performance of different models will be introduced in section 4.

3.1 Look-back Period

As has been discussed in the introduction, one of the basic strategies used when forecasting the momentum portfolio return is look-back period. The past twelve monthly returns are taken to calculate the cumulative return of the past year

$$R_t = \prod_{i=t-12}^{t-1} 1 + r_i, \tag{2}$$

where R_t is the cumulative return and r_i are regular monthly returns. This very simple prediction method has been shown to perform reasonably well (Barroso and Santa-Clara (2015)) and so we use it as a benchmark for comparison of the performances of other models.

3.2 Support Vector Regression

Support vector regression (SVR) is a type of regression method based on a machine-learning algorithm called support vector machine. The support vector machine classifies an input into discrete categories, whereas SVR extends the algorithm to return real-valued outputs. The goal of SVR is to minimize the normal vector w to the regression hyperplane, subject to constraint that all the data points considered have to lie within the error margin of our hyperplane. Mathematically this can be summarized into the following optimization:

$$\text{Minimize} \qquad \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N} \zeta_i \tag{3}$$

$$\text{such that} \qquad \mathbf{w}x_i + b - y_i \leq \epsilon + \zeta_i* \tag{4}$$

$$\mathbf{w}x_i + b - y_i \leq \epsilon + \zeta_i* \tag{5}$$

$$\zeta_i \geq 0 \tag{6}$$

where x_i are the data points with their targets y_i, a constant term b, the error margin ϵ, regularization constant C and the slack variables ζ and $\zeta*$. To transform this into non-linear SVR, we use the radial basis kernel function.

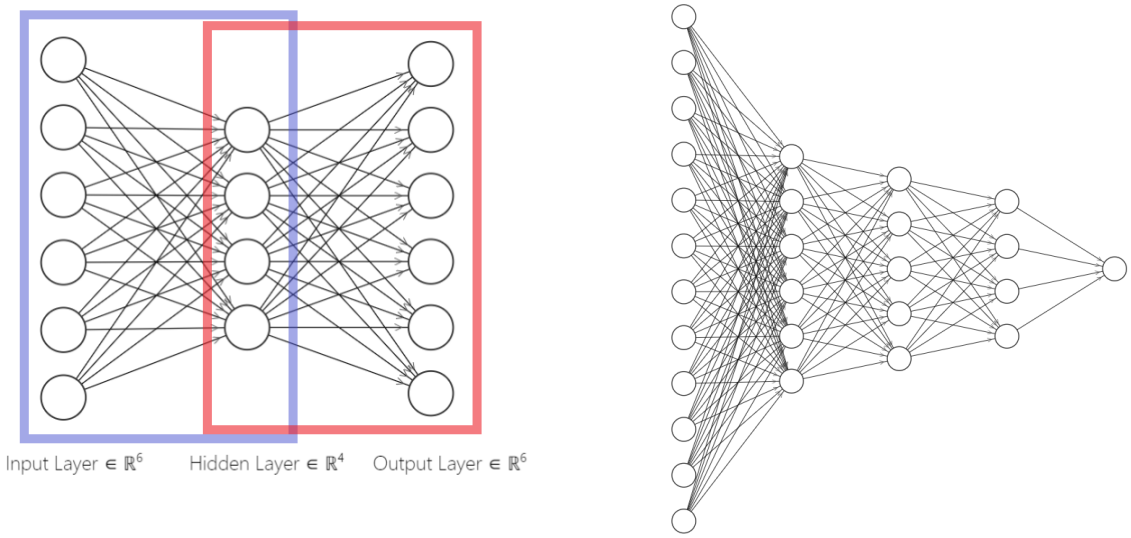
In our case, the input vector is formed from the monthly portfolio returns over the past year (12 observations) and the target is the next month's return. The training period consists of the past 300 vectors, hence using the returns of the last 312 months. Furthermore, to arrive at a good model specification, we optimize the hyperparameters C, ϵ and the kernel parameter γ by grid-searching through all the possible combinations, using the most recent 10% of training data to evaluate performances. The possible values for the hyperparameters are $\gamma \in \{10^i, 5{*}10^i; -5 \leq i \leq -1\}$, $\epsilon \in \{10^i, 5 * 10^i; -5 \leq i \leq -1\}$ and $C \in \{10^i, 5 * 10^i; -2 \leq i \leq 3\}$.

3.3 Stacked Autoencoder

Before we describe the stacked autoencoder network, it is important to briefly discuss what artificial neural networks (ANN) are. ANNs were made in approximation to a human brain's neurons, hence their setup and function is similar: these networks consists of multiple layers of the so-called neurons, where all neurons in one layer of the network are densely connected to all neurons in the next layer, meaning that the output of the neurons of layer x is used as an input in each neuron in layer $x + 1$. To determine how the information is processed in each neuron, an activation function is used:

$$z(W, X, b) = W^T X + b \tag{7}$$

This function combines the inputs X, weighs them based on the connection-specific weights W, and adds a 'bias' term b. The output is usually binary, taking the value 1 if $z(W, X, b)$ is above certain threshold, zero otherwise. The original data is fed into the first layer, and the processed information passes gradually through all the layers up to the last one. The final layer's output is then compared to the desired output and, based on the difference between the two (i.e. a loss function) the network's weights and biases are updated via back-propagation of the loss. The ultimate goal of the network is to learn the desired output for any possible input given to it.



(a) Autoencoder, with Encoder and Decoder

(b) Stacked Autoencoder model

Figure 1: Stacked autoencoder structure

Autoencoder is a type of artificial neural network used mainly as a dimension reduction technique. By means of unsupervised learning it finds a latent representation of the input, exploiting the user-defined, bottleneck structure of the neural network. In particular, both the input and the output layer have the same number of neurons, which is equal to the shape of the input. In our case this is 12 neurons, since we use the past 12 month's returns to predict next month's return. The intermediate layer of the network has fewer neurons than both the input and output layer, hence creating the bottleneck-type architecture. Figure 1(a) visualizes this network.

The autoencoder can be formally defined as transitions ϕ and ψ that minimize the mean squared error between the original input and the reconstructed input:

$$\begin{aligned}
&\phi : \mathcal{X} \to \mathcal{Y} \\
&\psi : \mathcal{Y} \to \mathcal{Z} \\
&\phi, \psi = \underset{\phi,\psi}{\arg\min} \|X - (\psi \circ \phi)X\|^2
\end{aligned}$$

where $(\psi \circ \phi)X$ is the reconstructed input X. The encoder transforms the input $\mathcal{X} \in \mathbb{R}^n$ to $\mathcal{Y} \in \mathbb{R}^m$:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{8}$$

with encoded input $\mathbf{y}$, activation function $f()$, the weight matrix $\mathbf{W}$, and a bias vector $\mathbf{b}$.

The decoder transforms the latent representation $\mathcal{Y} \in \mathbb{R}^m$ to $\mathcal{Z} \in \mathbb{R}^n$:

$$\mathbf{z} = f'\left(\mathbf{W}'\mathbf{y} + \mathbf{b}'\right) \tag{9}$$

where $\mathbf{W}'$ is the weight matrix of the decoder together with its bias term $\mathbf{b}'$, and $f'()$ denotes decoder's activation function.

For the autoencoder to learn non-linear, deep features of the data, we need to specify a nonlinear activation function $f(x)$, otherwise it would perform the same dimension reduction as regular Principal Component Analysis. For this reason we specified the activation function to be the sigmoid function: $f(x) = \frac{1}{1+e^{-x}}$, since this is one of the most popular choices (e.g. in Bao, Yue, and Rao (2017)). Another characteristic that is to be specified is the optimizer. The role of an optimizer in neural networks is crucial, because it iteratively updates the network weights based on the model loss, which could be interpreted as giving 'feedback' to the neurons. The well-known classical algorithm to do this is stochastic gradient descent (SGD), however, in many cases there are better, more advanced algorithms that can be used instead, possibly the network's training speed. One of them is the so-called 'Adam' , invented by Kingma and Ba (2015). It employs two extensions of SGD, namely adaptive gradient algorithm and root-mean-square propagation. In our study we decided to use the same optimizer settings as Kim (2019), using the Adam optimizer with a preset learning rate $\rho_a = 0.005$.

So far we have only described a simple autoencoder network. However, since previous studies suggest that the so-called stacked autoencoder shows excellent performance in stock return forecasting, we use this type of network too. The idea behind stacked autoencoders is rather straightforward: view the latent representation of one autoencoder as the input to the second autoencoder. The second autoencoder reduces the dimensions of the data's latent representation, hence extracting even 'deeper' features of the original data. In our case, three autoencoders are stacked onto each other in this manner, with the encoded input of the last autoencoder fully connected to one neuron in the last layer of the network. This layer enables us to use the stacked autoencoder as a return prediction model, using supervised learning. We train the last layer to return the prediction of the next month's return, based on the extracted deep features of the past 12 months. Figure 1(b) shows the structure of our stacked autoencoder network. At this point, we deviate from the study done by Kim (2019), where 12-6-6-6-1 stacked autoencoder structure was argued to perform best, because we believe that the network might not give consistent results. This is due to the second and the third autoencoder, since they both have a 6-6-6 structure. Hence, the 'bottleneck' architecture is missing, resulting in just a reshuffling of

the latent data. Therefore, we use a 12-6-5-4-1 structure, even though it has been shown that it may not provide the best results.

The way we train the model and predict the next month's return is as follows: our input is a vector of the past 12 monthly returns, and we train our first autoencoder model by feeding the previous 300 vectors to it (effectively using past 312 months' returns). To have the data reasonably scaled (on the interval $[0, 1]$ in our case), before feeding it to the autoencoder it is normalized (per vector of past returns x) as follows:

$$x \to \frac{x - x_{min}}{x_{max} - x_{min}} \tag{10}$$

After the model is trained, all the training data is passed through the encoder to get the latent representation. This constitutes the training set for the second autoencoder. The procedure is then repeated for the other autoencoders, with the output of the last one fed into the regression layer. This layer's target is the next month's return, so that the mean squared prediction error is minimized:

$$\text{MSPE} = \sum_{i=1}^{N} (y_i - \hat{y}_i) \tag{11}$$

After training all the models, the vector of this month's past returns is passed through all the encoders and the return for month $t + 1$ is predicted. The entire procedure is then repeated for every month, starting from January 1953 until March 2019.

3.4 Stacked Denoising Autoencoder

Stacked denoising autoencoders are extremely similar to the regular stacked autoencoders, so in this section we will only explain the differences between the two. The idea of denoising was first introduced by Vincent et al. (2010), who argues that if the initial input is noise-corrupted, the autoencoder doesn't train well to extract the latent factors, because the reconstructed data is not very meaningful. Since the inputs to our model are time series of portfolio returns, we suspect that they might contain noise and hence applying a denoising procedure is preferable.

The particular way in which the models are forced to 'denoise' the data is two-fold. Firstly, the original input is corrupted in some way, resulting in the encoder learning the following representation:

$$\mathcal{X}' \in \mathbb{R}^n \text{ to } \mathcal{Y} \in \mathbb{R}^m : \quad \mathbf{y} = f(\mathbf{W}\mathbf{x}' + \mathbf{b}) \tag{12}$$

which is very similar to the procedure described by equation 8, except that the original input x is substituted with the noise-corrupted one x'. Then, the decoder reconstructs the original data the same way as is done in regular decoders, just as in equation 9.

There are multiple ways of introducing noise to the input; two most popular ones are masking noise and Gaussian noise. Masking noise takes the input and randomly replaces certain fraction of it by zeros. This is often done in autoencoders trained for image classification purposes, and Vincent et al. (2010) has shown that, in general, these denoising autoencoders perform significantly better than the basic ones. The other denoising process, namely additive isotropic Gaussian noise works in a different manner; it adds values generated by normal distribution with zero mean and pre-specified standard deviation to each element of the input. Contrary to Kim (2019), we only replace the first autoencoder by a denoising one, because we believe that after denoising of the original data, the latent representation should not be noise-corrupted anymore.

In our study, we perform hyper-parameter search on both the optimal fraction of masked input in case of masking noise, and optimal standard deviation of the noise in case of additive isotropic Gaussian process. The grid for masking noise is chosen to be $\{0, 0.1, 0.25, 0.4\}$ whereas for the Gaussian noise it is $\{0.05, 0.1, 0.15, 0.2\}$. Moreover, the last 10% of the training data is always used to decide which value of the hyperparameter gives the best outcome, similarly as in support vector regression.

3.5 Generative Adversarial Network

Generative adversarial network (GAN) is a deep-learning algorithm in which two artificial neural networks compete with each other. Firstly, there is a generative model that, based on some latent input, generates data such that it resembles real data as closely as possible. Secondly, a discriminative model is trained to distinguish between actual data and artificial data created by the generator. This way, the generator is forced to learn the joint probability distribution $P(x, y)$ such that it confuses the discriminator into labeling the generated data as real. Hence, the more the generator is trained, the smaller the differences between real and artificial data, which in turn forces the discriminator to constantly improve. Figure 2 gives a basic layout of the network.

Our generator model is build from multiple layers of neurons, which are either regular (so-called perceptron) or "recurrent" units, which take into account the past values of the input

[1]Figure obtained from https://towardsdatascience.com/aifortrading-2edd6fac689d

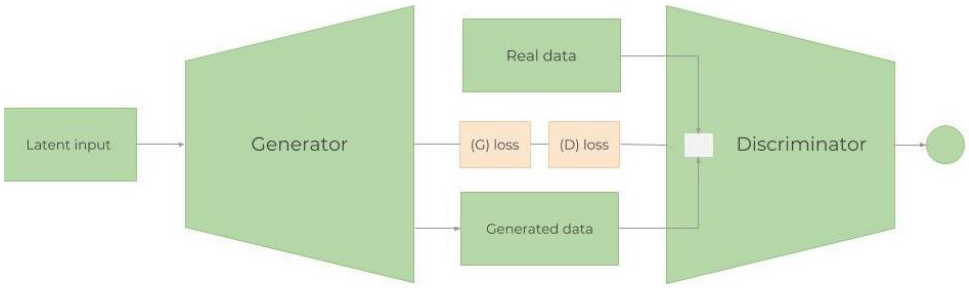


Figure 2: GAN layout[1]

in a special way. Both types of neurons are discussed in 3.5.1 and 3.5.2. In our case the data generated by the model is the return for the next h months, based on a latent input in the form of past 12 monthly returns. The output is then mixed with real data and is fed to the discriminator, which then tries to classify whether the data is artificial or real. The discriminator is usually built up as a convolutional or a recurrent neural network. The whole system is then optimized in two steps, firstly minimizing the loss of the discriminator with generator model fixed, and then minimizing the generator loss, keeping the weights of the discriminator fixed. The training is repeated for a certain number of epochs, or until the model converges, i.e. until the generator creates data that is so similar to real data that the discriminator cannot distinguish it. Moreover, the forecast is simply made by feeding the generator with the last 12 monthly returns such that it predicts the returns for the next h months. Formally, algorithm 1 shows the training and forecasting method.

Algorithm 1:

for *every hth month in dataset* **do**
- **compile** D and G models
- **set** learning rates ρ_G and ρ_D, batch size b
- **initialize** weights W_G and W_D
- **while** *not converged* **do**
  - randomly choose b observations from the training set
  - generate artificial data x' of length h
  - append x' to real data, creating $[12x1]$ vector
  - **Train** D model on the batch of real and generated data
  - **Fix** discriminator's weights
  - **Train** full network (G) on randomly chosen data
- **end**
- Predict the returns of the next h months.

Now that the general model layout and training procedure have been described, the technical setup of the network will be explained. Firstly, the activation functions for both discriminator

(D) and generator (G) are set to the so-called 'Leaky Rectified Linear Unit' function:

$$y = max(\alpha x, x) \tag{13}$$

with $\alpha = 0.2$ in our case. This activation function has been shown to eliminate the problems of vanishing gradient that is very common for GAN, while improving on the basic Rectified Linear Unit function which often suffers of 'dying' neurons (neurons that never get activated). Moreover, to prevent the exploding gradient problem, the gradients are clipped at the value of 0.5 and their norm is forced not to exceed 1.

In terms of the loss functions used, there are two issues to consider: the potential instability of the model, and its non-convergence. The problem of non-convergence and potential instability is due to the nature of the GAN, when viewed in a game-theoretic framework. This happens because some loss functions do not converge to their minima when using gradient descent in a minimax game (as is the setup here). A possible prevention is to use an augmented loss function for the generator model. In this paper, it is a combination of the forecast error and the so-called fool rate, that is, the fraction of time that the discriminator classifies artificial data as real. Moreover, since we specifically care about the correctness of the sign of generated return, the 'sign loss' is accounted for. The loss function of the generator, similarly to Zhou et al. (2018), looks as follows:

$$L_G(X, Y) = \lambda_1 * L_{BC}(D(\hat{Y}), 1) + \lambda_2 * L_{MSE}(\hat{Y}, Y) + \lambda_3 * L_{SL}(\hat{Y}, Y) \tag{14}$$

with $L_{BC}(D(\hat{Y}), 1)$ being the correct classification loss (computed as binary cross-entropy), L_{MSE} being the mean squared prediction loss, L_{SL} the sign loss (0 if the sign is correct, 1 otherwise), and loss weights $\lambda_1, \lambda_2, \lambda_3$. The values of lambdas are set to $\{\lambda_1, \lambda_2, \lambda_3\} = \{0.7, 0.21, 0.09\}$, hence putting emphasis on the original loss of not fooling the discriminator. The loss of the discriminator model is a simple binary cross-entropy loss. Moreover, to avoid overfitting of both D and G models, we introduce dropout rates of 0.4 and 0.3, respectively. This leaves out certain fraction of connections in the network during training. The following subsections describe two alternative GAN specifications, and the specific network setup.

3.5.1 Multi-layer Perceptron GAN

A perceptron is a basic type of neuron that takes a vector-valued input and, based on the weights and bias, returns a real-valued output. Since this is the most widely used kind of

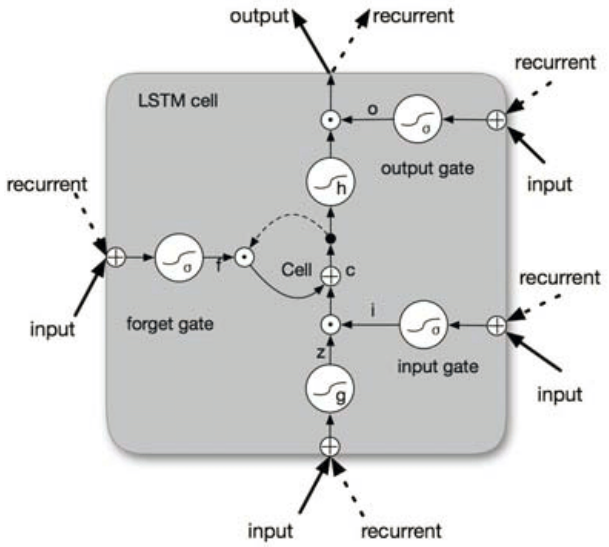


Figure 3: LSTM unit[2]

neuron, it is used in our basic generator model. Specifically, there are 4 layers of neurons, structured as 12-8-4-1 neurons per layer. The generator returns one value, hence the prediction window in this case is set to $h = 1$. The forecast is then appended to the past 11 months' returns and is passed to the discriminator, which has the same setup: 4 layers structured as 12-8-4-1 neurons per layer. The discriminator outputs a single binary value, classifying the input as either real (1) or fake (0). The best pair of learning rates for the generator and discriminator is found by grid-searching through the sets $\rho_G = \{0.01, 0.005, 0.001, 0.0005, 0.0001\}$ and $\rho_D = \{0.05, 0.01, 0.005, 0.001, 0.0005\}$, using the first 315 months to train the models. The search resulted in the best pair of learning rates $(\rho_G, \rho_D) = (0.001, 0.01)$.

3.5.2 Recurrent GAN

Recurrent neural networks rely on recurrent neurons to capture time-dependencies. These neurons are designed to consider the past values of the input when calculating the output. In our recurrent GAN, we make use of the 'Long Short-term Memory' (LSTM), first introduced by Hochreiter and Schmidhuber (1997). A single LSTM unit consists of a cell, which holds the memory, and 3 regulators, commonly known as gates. The input gate controls the flow of a new value into the LSTM cell, the output gate decides to what extent is the current cell value going to affect the output and the forget gate controls which elements are retained in the cell. A simplified visualization of the LSTM unit is shown in Figure 3. For a detailed mathematical representation of this model, refer to Hochreiter and Schmidhuber (1997).

[2]Figure obtained from https://www.researchgate.net/figure/Schematic-of-LSTM-unit_fig3_306093553

The generator in our case consists of 4 layers, with structure 8-4-1-h, where the first three layers are LSTM cells that return sequences and the last h cells outputs the forecast returns for the next h months. The discriminator network only consists of 2 layers of LSTM neurons followed by 2 basic perceptron layers, structured as 6-4-2-1. Due to a large computational burden, we did not perform any form of optimization of the structures.

Since the full model is large and it's computationally heavy to train, we decided on the best hyperparameter set in advance, using the same options for learning rates as above, and with forecast length $h \in \{1, 2, 3\}$. Based on the performance over 4 random periods, the learning rates for the top decile portfolio are set to $\rho_G = 0.001$ and $\rho_D = 0.005$, with forecast length $h = 3$. For the bottom decile portfolio, the grid search suggests using $(\rho_G, \rho_D, h) = (0.001, 0.005, 1)$. Moreover, to reduce the computationally-heavy training procedure, model is trained once every year and then four 1-step-ahead forecasts (of length h) are made. This corresponds to updating the portfolio weights h months, with the model being updated annually. However, it is worth noting that the outcomes for the different combinations vary when the hyperparameter search is repeated, suggesting a slight instability of the model.

3.5.3 Economic Interpretation

In the portfolio building environment, the investors often base their decisions on the past performance of a certain asset, whether it is a stock or a portfolio consisting of multiple stocks. In the setting of momentum portfolios, the investors typically look at the performance of top and bottom decile throughout the past year. Hence, the generator model can be interpreted as an artificial investor who, based on historical returns, predicts the future return of the portfolio and then decides whether it is worth investing in it or not. Doing this, the investor tries to minimize the difference between the future return based on their prediction and the true future return, also paying special attention to the correct sign of the return.

Once the prediction is made and acted upon and the prediction period passes, the investor looks back and compares the predicted return with the actual return of the period. He judges the correct probability of his/her own prediction, which serves as a feedback for the model that was used. This judgment corresponds to the discriminator model. All in all, GAN can be summarized in a game-theoretic framework as a game between the generator and the discriminator, with one of the possible equilibria being that the generator manages to learn the joint data distribution and the discriminator is forced to make a random guess when determining how the data originated.

Table 1: Regular returns (03/1953 - 03/2019)

	Mkt	WO	WML	Lookback	SVR	SAE	SDAE-I	SDAE-II	RGAN
Avg. return (%)	0.59	1.69	0.97	1.20	1.56	1.44	1.23	1.47	1.53
Max. return (%)	16.10	31.22	19.16	31.46	31.22	23.03	31.22	31.22	31.46
Min. return (%)	-23.24	-32.80	-60.40	-64.75	-26.21	-32.80	-60.40	-60.40	-32.32
Ann. return (%)	6.24	19.45	9.62	10.35	17.71	16.22	12.59	16.04	17.71
Ann. vol. (%)	14.82	21.78	20.31	28.19	21.10	20.57	22.24	22.02	19.72
Sharpe ratio	0.42	0.89	0.47	0.36	0.84	0.79	0.57	0.73	0.90
Info. ratio	NA	0.30	0.05	0.07	0.18	0.20	0.11	0.16	0.15
Kurtosis	1.91	2.90	21.01	9.53	3.21	2.85	14.99	13.73	4.95
Skewness	-0.54	-0.52	-2.79	-1.74	-0.53	-0.54	-1.87	-1.64	-0.96
MDD (%)	55.68	59.77	83.26	77.77	69.93	59.44	83.82	66.88	45.84

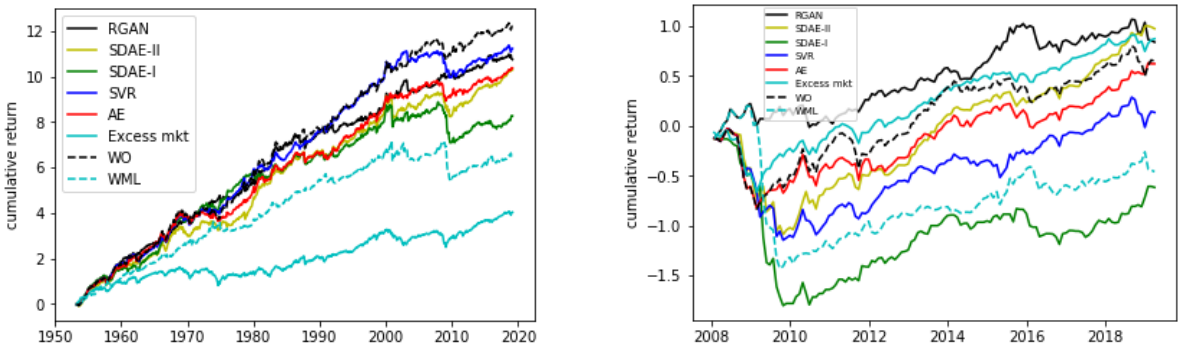
Explanations of the abbreviations can be found in the appendix 5.

4 Results

The models' performances are evaluated for both for the full sample (01/1953 - 03/2019) and a sub-sample (01/2008 - 03/2019) starting right before the financial crisis. All the models are evaluated statistically and after the portfolios are built their financial performance is measured. Moreover, the convergence of our GAN model with perceptron neurons was unfortunately not achieved, hence we do not report its results, as they differed significantly over multiple runs.

4.1 Regular Portfolios

Table 1 shows that, based on the average return, pure WO strategy performs best, not very closely matched by any other strategy. Contrary to Kim (2019), we find that SVR produces the highest returns among the SWML strategies, which is surprising, since it is one of the simpler models we used. However, the recurrent GAN model almost matches the average return of SVR, trailing by 0.03%. In general, the differences in average returns between the models are not large, with almost all of them being inside the interval [1.2% , 1.7%]. Considering the maximum and minimum return, all the portfolios except WML and SAE-SWML generate the highest return of around 31%, with the former two achieving around 20%, hence not much higher than the maximum of a simple market return (16.1%). Similarly, the minimum returns cluster around 2 values; -30% for WO, SVR, SAE and RGAN, and -60% for WML, Lookback, SDAE-I and SDAE-II. This shows that employing more advanced models in return prediction does not always reduce the risk of a large negative return. We note, however, that RGAN and SAE do reduce this risk. This can also be seen from the maximum drawdown, which shows the worst possible (negative) return over any period of time, which is very informative about the downside risk of a



(a) Full sample: 03/1953 - 03/2019

(b) Subsample: 01/2008 - 03/2019

Figure 4: Regular cumulative log-returns

given portfolio. We observe that RGAN reduces this kind of risk most, yielding the worst return of -46%, whereas the stacked denoising autoencoders do not improve in this sense over a pure WML portfolio. Moreover, RGAN is the only portfolio that outperforms the market portfolio in this measure.

Many investors consider another performance indicator, that is the Sharpe ratio. It describes the return of a portfolio taking into account its corresponding volatility, in our case calculated as the ratio between annualized return and annualized volatility. The highest Sharpe ratio is achieved by RGAN and pure WO strategy (0.9), with SVR closely matching them. The superior performance of RGAN and SVR comes mostly from their annualized returns, since the volatility is similar across all the strategies (at around 20%), except the Lookback period (with 28%). Interestingly, pure WML strategy has a low Sharpe ratio, at 0.47, that comes mainly from a low annualized return. Information ratio is an adjustment of the Sharpe ratio, and is calculated as the difference between the portfolio return and the annualized return of a benchmark portfolio, divided by the volatility of this difference. Using excess market return as the benchmark, we see that WO strategy yields the largest improvement, with more sophisticated strategies having similar information ratios (of around 0.18). On the other hand, WML and Lookback give very small improvements over the excess market portfolio. All in all, we see that WO performs best in most indicators, with SVR and RGAN being close behind, while RGAN performing very well in reducing the downside risk of the momentum strategy.

The left panel of figure 4 confirms that, for the full sample period, WO portfolio has the highest 66-year cumulative return, with SVR and RGAN trailing closely behind. Moreover, the excess market portfolio performs clearly the worst over the full period, with WML being a big improvement, nevertheless insufficient compared to the more complex models. The cumulative returns all increase exponentially over time, with one large drop around the year 2009, which

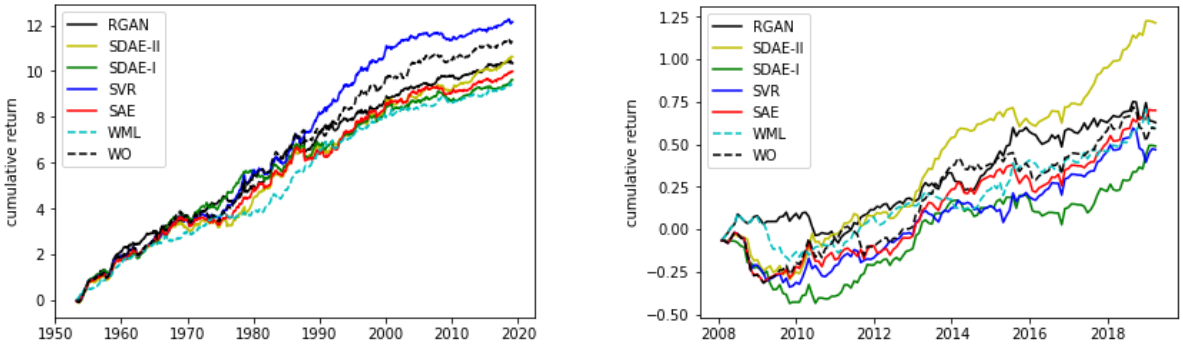
Table 2: Volatility-scaled returns (03/1953 - 03/2019)

	Mkt	WO	WML	Lookback	SVR	SAE	SDAE-I	SDAE-II	RGAN
Avg. return (%)	0.59	1.53	1.23	1.35	1.62	1.35	1.31	1.44	1.39
Max. return (%)	16.10	21.64	15.39	18.60	21.64	21.64	18.60	21.64	21.64
Min. return (%)	-23.24	-39.10	-25.33	-39.10	-28.65	-39.09	-39.10	-39.10	-39.09
Ann. return (%)	6.24	17.86	14.70	15.22	19.46	15.64	15.03	16.76	16.26
Ann. vol. (%)	14.82	18.83	13.70	19.56	17.48	17.51	17.45	18.22	17.29
Sharpe ratio	0.42	0.95	1.07	0.78	1.14	0.89	0.86	0.92	0.94
Info. ratio	NA	0.28	0.10	0.11	0.22	0.19	0.15	0.18	0.14
Kurtosis	1.91	5.65	5.77	5.56	2.87	6.23	7.58	7.38	8.97
Skewness	-0.54	-0.86	-1.04	-1.16	-0.33	-0.81	-1.07	-1.04	-1.37
MDD (%)	55.68	40.64	31.28	50.01	34.87	45.76	46.16	39.63	54.20

corresponds to the financial crisis and the rebounds of the markets afterwards. It is of our interest to inspect this subsample and see which portfolio minimizes the losses and hence may perform best in the future crises. The right panel of figure 4 shows that, starting in January 2008, WML and SDAE-I end up losing money until the end of the period (March 2019), whereas the other portfolios were profitable. Surprisingly, even though SVR performed well over the full sample, in the sub-period it breaks even only in the beginning of 2019. There is also a significant difference in cumulative returns of SDAE-II (being the most profitable) and SDAE-I underperforming even the pure WML portfolio. It is also worth noting that RGAN performed best in the most part of the subsample, being consistently above the excess market return. For detailed performance measures of the subsample, refer to Table 6 in the appendix.

4.2 Volatility-scaled Portfolios

According to Barroso and Santa-Clara (2015), volatility scaled portfolios should be able to avoid large drawdowns in value. Table 2 shows that for almost all portfolios, the maximum drawdown has decreased significantly, apart from RGAN. This supports our earlier claims that RGAN avoids large losses reasonably well. We also observe that for WO, SAE and SDAE-II, the average return has increased, however, the other portfolios experience a slight drop in the returns. We expect the reason to be that in times of large-positive and negative fluctuations, the top and bottom decile portfolios are assigned lower weights, reducing both possible large gains and losses. This is also visible in the maximum return of the volatility-scaled strategies (20%), which are all lower than the ones without the volatility scale (30%). Overall, the annualized volatility decreased to around 17%, which is below the target set to 20%. This, in effect, increased the Sharpe ratios to values around 0.9.



(a) Full sample: 03/1953 - 03/2019

(b) Subsample: 01/2008 - 03/2019

Figure 5: Volatility-scaled cumulative log-returns

In terms of specific portfolios, SVR now performs better than WO in terms of average return, with the other strategies all being similar and somewhat lower. The Sharpe ratio is the highest for SVR (1.14), confirming that this method is very effective when making return forecasts. Looking at the information ratio, WO performs best compared to the excess market portfolio, with SVR being marginally worse. Moreover, based on the values of skewness and kurtosis, we see that normality might be a reasonable assumption for the SVR returns, but not for the other ones, with negative skewness values and large kurtosis. In general, we conclude that volatility-scaling does not improve the results of advanced deep-learning models as compared to the simpler ones.

Comparing the cumulative returns of the volatility-scaled portfolios in the left panel of Figure 5 with the regular portfolios, we see that the gap between the best and the worst performance is much tighter, with SVR now dominating the other strategies, and SDAE-I performing only as good as the excess market portfolio. Moreover, the dip during the crisis has almost disappeared, so it is of our interest to investigate which model benefits most from the volatility-scaling. For detailed performance measures of the subsample, refer to Table 7 in the appendix.

The right panel of Figure 5 shows that the cumulative return of SDAE-II largely deviates from the rest, breaking even at the end of year 2011. Again, we observe big difference in the performance of the two stacked denoising autoencoders. Moreover, the value of RGAN portfolio never falls below the initial value, even though it is not performing as well as without the volatility scale. Since the performance of RGAN is of our special interest, we evaluate its statistical performance below, comparing it to the other predictive models.

Table 3: Statistical evaluation of the forecasts

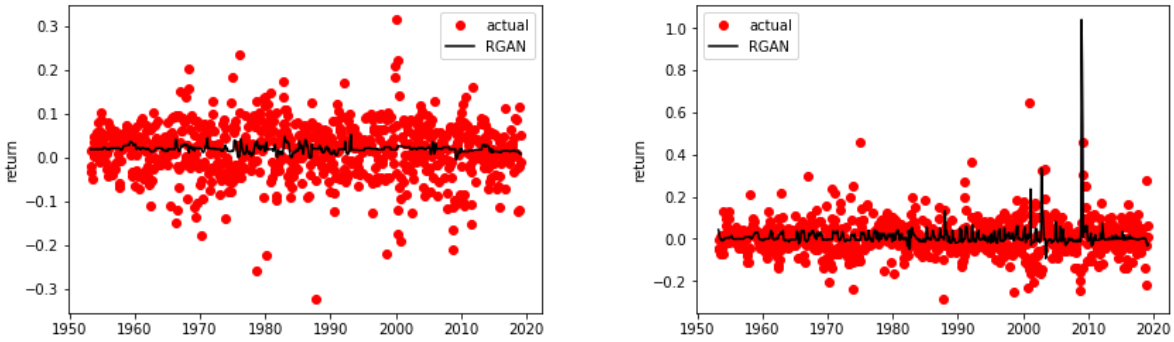
		Lookback	SVR	SAE	SDAE-I	SDAE-II	RGAN
Top decile	MSPE ($*10^{-3}$)	144.07	4.00	4.07	4.42	4.29	4.02
	hit rate (%)	60.73	63.76	65.03	61.49	61.24	64.90
Bottom decile	MSPE ($*10^{-3}$)	161.95	7.21	7.29	7.78	7.40	9.88
	hit rate (%)	53.79	53.41	51.89	51.64	53.41	53.03

4.3 Statistical Evaluation

We base or statistical evaluation of the forecasts on two measures; mean squared prediction error (MSPE), computed as in equation 11, and the so-called hit rate, which is the fraction of correct sign predictions. Inspecting Table 3, we notice that the lookback-period has large MSPE for both top and bottom decile portfolios, compared to the other models. Since its hit rates do not differ much from the rest, we conclude that the lookback largely overestimates the magnitudes of the returns. The other models have similar MSPE for both the top decile (around $4 * 10^{-3}$) and the bottom decile ($7.5 * 10^{-3}$ with RGAN being slightly larger). In general, the models perform much better for the top decile than for the bottom decile portfolio. In terms of hit rate, the largest one is found for SAE and RGAN for the top decile, and Lookback for the bottom decile, although the hit rates do not differ much from each other. For the bottom decile, all the models perform very poorly, with hit rates close to 50%, showing that they are not much better than guessing the sign of the next return. Moreover, when inspecting the graphs (6) of RGAN return forecasts in relation to the actual values, we clearly see that the model largely underestimates the magnitude of the returns in both directions and for both top and bottom decile portfolios. However, since we focus mostly on the correct sign prediction, it still produces one of the best portfolios (as shown earlier). For the graphs of SDAE-I and SDAE-II models, as well as statistical performance in the subsample, please refer to the table 8 and figures 7 and 8 in the appendix.

5 Conclusion

In this paper we investigated possible improvements in momentum strategies by employing simpler well-known models as more complex machine- and deep-learning models. Evaluating the performance and comparing it to the excess market portfolio and simple WO and WML strategies, we found that the deep learning models provide an extra profit, and mostly help with reducing the downside risk of the investments. Using stacked autoencoder networks similar



(a) Top decile portfolio

(b) Bottom decile portfolio

Figure 6: RGAN forecasts and actual returns

to Kim (2019), we observed somewhat different results, likely due to the different imposed structure. Moreover, we found that recurrent generative adversarial neural networks, although not improving the average return by a large margin, are of big importance when limiting the downside risk of a momentum portfolio. The maximum drawdown for portfolios based on these models have shown almost a 50% decrease compared to a winner-minus-loser portfolio, and a 15% reduction compared to winner-only portfolio.

We also observed a significant decrease in exploitability of the bottom decile stocks, which can be seen from the large differences of simple WO and WML strategies. Since our models are all used together with the so-called selective WML, we find them to be slightly disappointing compared to simple WO strategies. However, the stacked denoising autoencoders with Gaussian noise and recurrent GAN bring big improvements over the crisis periods, successfully avoiding large losses.

With this new knowledge, it would be in our interest to further investigate the possible improvements, this time combining the selective, prediction-based models with the winner-only approach. We believe this so-called selective winner-only would bring larger average returns with reduced downside risk to the momentum portfolios. Another further research includes training different generative models, such as variational autoencoders, since we have shown that generative models improve the portfolios over some discriminative models. We are also concerned whether the recurrent GANs and stacked autoencoders had the optimal hyperparameter set, as our search for the best set of values was quite restricted, sometimes even pre-set. This was done due to the time- and computational burden, so it is crucial to expand this search in the future studies. All in all, based on our research, as well as the economic and game-theoretic interpretation, we can now answer the earlier stated research question; yes, the application of generative adversarial networks enhances the performance of momentum portfolios.

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, 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.

Berner, Julius, Philipp Grohs, and Arnulf Jentzen (2018). "Analysis of the generalization error: Empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations". In: *CoRR* abs/1809.03062. arXiv: 1809.03062. URL: http://arxiv.org/abs/1809.03062.

Dai, Bo et al. (2017). "Towards Diverse and Natural Image Descriptions via a Conditional GAN". In: *The IEEE International Conference on Computer Vision (ICCV)*.

Dixon, Matthew F, Nicholas G Polson, and Vadim O Sokolov (2018). "Deep learning for spatio-temporal modeling: Dynamic traffic flows and high frequency trading". In: *Applied Stochastic Models in Business and Industry*.

Esteban, Cristóbal, Stephanie L Hyland, and Gunnar Rätsch (2017). "Real-valued (medical) time series generation with recurrent conditional gans". In: *arXiv preprint arXiv:1706.02633*.

Fama, Eugene F. and Kenneth R. French (1996). "Multifactor Explanations of Asset Pricing Anomalies". In: *The Journal of Finance* 51.1, pp. 55–84.

French, Kenneth R. (2019). *Kenneth R. French - Data Library*. URL: http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html.

Goodfellow, Ian et al. (2014). "Generative adversarial nets". In: *Advances in neural information processing systems*, pp. 2672–2680.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "LSTM can solve hard long time lag problems". In: pp. 473–479.

I. Sapankevych, Nicholas and Ravi Sankar (2009). "Time Series Prediction Using Support Vector Machines: A Survey". In: *Computational Intelligence Magazine, IEEE* 4, pp. 24 –38. DOI: 10.1109/MCI.2009.932254.

Isola, Phillip et al. (2017). "Image-to-image translation with conditional adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134.

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. ISSN: 00221082, 15406261. URL: http://www.jstor.org/stable/2328882.

— (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.

Kim, Saejoon (2019). "Enhancing the momentum strategy through deep regression". In: *Quantitative Finance* 0.0, pp. 1–13.

Kingma, Diederik and Jimmy Ba (2015). "Adam: a method for stochastic optimization (2014)". In: *arXiv preprint arXiv:1412.6980* 15.

Law, T. and J. Shawe-Taylor (2017). "Practical Bayesian support vector regression for financial time series prediction and market condition change detection". In: *Quantitative Finance* 17.9, pp. 1403–1416. DOI: 10.1080/14697688.2016.1267868.

LeCun, Yann et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Mirza, Mehdi and Simon Osindero (2014). "Conditional Generative Adversarial Nets". In:

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.

Subrahmanyam, Avanidhar (2018). "Equity market momentum: A synthesis of the literature and suggestions for future work". In: *Pacific-Basin Finance Journal* 51.C, pp. 291–296. DOI: 10.1016/j.pacfin.2018.08..

Vincent, Pascal et al. (2010). "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion". In: *J. Mach. Learn. Res.* 11, pp. 3371–3408. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1756006.1953039.

Zhou, Xingyu et al. (2018). "Stock Market Prediction on High-Frequency Data Using Generative Adversarial Nets". In: *Mathematical Problems in Engineering* 2018, pp. 1–11. DOI: 10.1155/2018/4907423.

Appendix

Table 4: Descriptive statistics of the raw data

	mean (%)	std. dev.	min	25%	50%	75%	max
Top decile	1.75	6.29	-32.32	-1.61	2.21	5.48	31.46
Bottom decile	0.74	8.44	-28.54	-3.70	0.42	4.48	64.75

Table 5: Financial performance measures of the raw data

	Ann. return (%)	Ann. vol. (%)	Sharpe	Skewness	Kurtosis	MDD (%)
Top decile	20.28	21.77	0.93	-0.48	2.82	59.38
Bottom decile	4.96	29.23	0.17	1.21	7.19	85.10

Model abbreviations

- Mkt: excess market return
- Lookback: SWML with look-back period
- SVR: SWML with support vector regression
- SAE: SWML with basic stacked autoencoder
- SDAE-I: SWML with denoising stacked autoencoder, masking noise
- SDAE-II: SWML with denoising stacked autoencoder, Gaussian noise
- RGAN: SWML with generative adversarial network, recurrent neurons

Table 6: Regular returns (01/2008 - 03/2019)

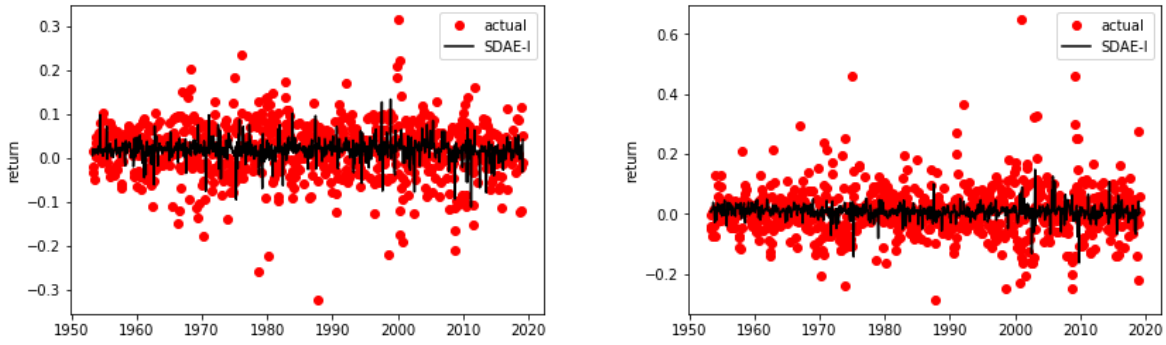
	Mkt	WO	WML	Lookback	SVR	SAE	SDAE-I	SDAE-II	RGAN
Avg. return (%)	0.75	0.61	-0.13	0.03	0.24	0.58	-0.25	0.87	0.76
Max. return (%)	11.35	15.77	14.38	24.82	14.38	15.77	15.77	15.77	14.60
Min. return (%)	-17.23	-21.32	-40.97	-45.85	-25.05	-21.32	-40.97	-25.68	-15.87
Ann. return (%)	8.09	5.05	-4.84	-5.06	0.37	4.88	-6.10	8.21	7.77
Ann. vol. (%)	15.53	21.67	24.41	31.38	22.15	20.48	24.07	21.65	17.85
Sharpe ratio	0.52	0.23	-0.20	-0.16	0.02	0.24	-0.25	0.38	0.43
Info. ratio	NA	-0.05	-0.10	-0.07	-0.09	-0.04	-0.14	0.02	0.001
Kurtosis	1.47	0.96	9.51	5.41	3.07	1.51	9.25	4.18	0.83
Skewness	-0.75	-0.74	-2.44	-1.36	-1.30	-0.74	-2.03	-1.43	-0.57
MDD (%)	46.34	56.03	81.03	75.32	68.25	55.67	82.00	65.56	24.84

Table 7: Volatility-scaled returns (01/2008 - 03/2019)

	WO	WML	Lookback	SVR	SAE	SDAE-I	SDAE-II	RGAN
Avg. return (%)	0.46	0.45	0.03	0.37	0.53	0.38	0.92	0.48
Max. return (%)	9.76	10.51	10.51	10.51	9.76	9.76	9.76	10.51
Min. return (%)	9.79	-10.01	-15.79	-9.79	-8.96	-8.74	-8.96	-10.01
Ann. return (%)	4.90	4.96	3.20	3.78	5.93	4.03	10.91	5.28
Ann. vol. (%)	12.03	10.35	13.90	11.61	11.29	11.01	10.82	11.14
Sharpe ratio	0.41	0.48	0.23	0.33	0.53	0.37	1.01	0.47
Info. ratio	-0.11	-0.05	-0.06	-0.10	-0.06	-0.08	0.04	-0.05
Kurtosis	0.57	1.53	1.40	1.34	0.55	0.86	1.05	1.59
Skewness	-0.51	-0.27	-0.46	-0.35	-0.37	0.07	-0.17	-0.23
MDD (%)	26.11	24.39	22.82	28.06	25.80	31.67	23.74	17.07

Table 8: Statistical evaluation of the forecasts (01/2008 - 03/2019)

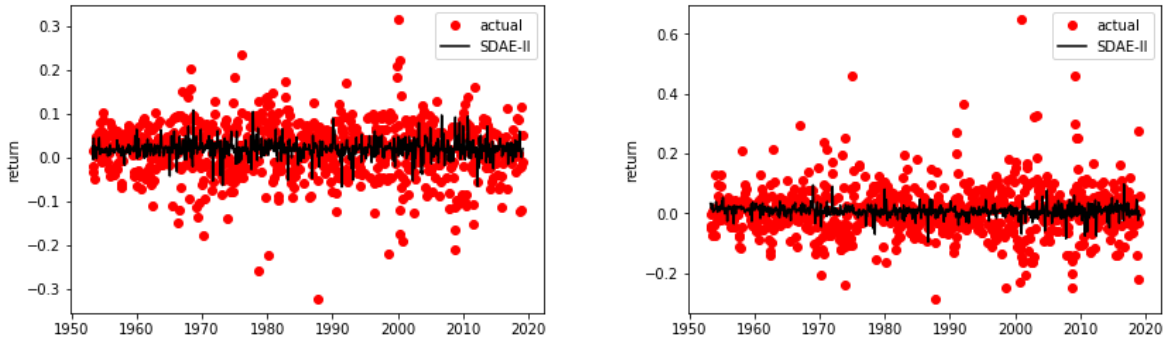
		Lookback	SVR	SAE	SDAE-I	SDAE-II	RGAN
Top decile	MSPE (*10^-3)	71.48	4.14	3.95	4.01	4.54	4.02
	hit rate (%)	51.11	60.00	60.74	54.07	59.26	60.00
Bottom decile	MSPE (*10^-3)	333.62	9.50	9.59	10.65	10.00	22.47
	hit rate (%)	51.11	53.33	54.07	50.37	55.55	51.85



(a) Top decile portfolio

(b) Bottom decile portfolio

Figure 7: SDAE-I forecasts and actual returns



(a) Top decile portfolio

(b) Bottom decile portfolio

Figure 8: SDAE-II forecasts and actual returns

Python code The following code files are provided with this report:

- cleanData.ipynb : cleans the data and transforms it into an easy-to-read data-type,
- Main.ipynb : code for basic strategies (WO, WML, SWML-lookback) and support vector regression,
- AE_alt.ipynb : code for stacked (denoising) autoencoders,
- r-r-gan.ipynb : code for the (recurrent) GAN,
- evaluate.ipynb : builds the portfolios, evaluates financial and statistical performances.