Dautel, Alexander J.; Härdle, Wolfgang Karl; Lessmann, Stefan; Seow, Hsin-Vonn

Working Paper

Forex Exchange Rate Forecasting Using Deep Recurrent Neural Networks

IRTG 1792 Discussion Paper, No. 2019-008

Provided in Cooperation with:
Humboldt University Berlin, International Research Training Group 1792 "High Dimensional Nonstationary Time Series"

Suggested Citation: Dautel, Alexander J.; Härdle, Wolfgang Karl; Lessmann, Stefan; Seow, Hsin-Vonn (2019) : Forex Exchange Rate Forecasting Using Deep Recurrent Neural Networks, IRTG 1792 Discussion Paper, No. 2019-008, Humboldt-Universität zu Berlin, International Research Training Group 1792 "High Dimensional Nonstationary Time Series", Berlin

This Version is available at:
https://hdl.handle.net/10419/230784

IRTG 1792 Discussion Paper 2019-008

Forex Exchange Rate Forecasting Using Deep Recurrent Neural Networks

Alexander J. Dautel [*]
Wolfgang K. Härdle [*]
Stefan Lessmann [*]
Hsin-Vonn Seow [*2]

[*] Humboldt-Universität zu Berlin, Germany
[*2] Nottingham University, Malaysia

This research was supported by the Deutsche Forschungsgesellschaft through the International Research Training Group 1792 "High Dimensional Nonstationary Time Series".

http://irtg1792.hu-berlin.de
ISSN 2568-5619

Digital Finance manuscript No.
(will be inserted by the editor)

Forex Exchange Rate Forecasting Using Deep Recurrent Neural Networks

Alexander Jakob Dautel · Wolfgang Karl Härdle · Stefan Lessmann · Hsin-Vonn Seow

Received: date / Accepted: date

Abstract Deep learning has substantially advanced the state-of-the-art in computer vision, natural language processing and other fields. The paper examines the potential of contemporary recurrent deep learning architectures for financial time series forecasting. Considering the foreign exchange market as testbed, we systematically compare long short-term memory networks and gated recurrent units to traditional recurrent architectures as well as feedforward networks in terms of their directional forecasting accuracy and the profitability of trading model predictions. Empirical results indicate the suitability of deep networks for exchange rate forecasting in general but also evidence the difficulty of implementing and tuning corresponding architectures. Especially with regard to trading profit, a simpler neural network may perform as well as if not better than a more complex deep neural network.

Keywords Deep learning · Financial time series forecasting · Recurrent neural networks · Foreign exchange rates

Alexander Jakob Dautel
School of Business and Economics, Humboldt-Universiät zu Berlin, Unter-den-Linden 6, 10099 Berlin, Germany
E-mail: a.j.dautel@gmail.com

Wolfgang Karl Härdle
School of Business and Economics, Humboldt-Universiät zu Berlin, Unter-den-Linden 6, 10099 Berlin, Germany
Singapore Management University, 50 Stamford Road, Singapore 178899
E-mail: haerdle@hu-berlin.de

Stefan Lessmann
School of Business and Economics, Humboldt-Universiät zu Berlin, Unter-den-Linden 6, 10099 Berlin, Germany
E-mail: stefan.lessmann@hu-berlin.de

Hsin-Vonn Seow
Nottingham University Business School, 43500 Semenyih Selangor Darul Ehsan, Malaysia
E-mail: Hsin-Vonn.Seow@nottingham.edu.my

1 Introduction

Deep learning has revitalized research into artificial neural networks, a machine learning method invented in the 1950s and 1960s, in the past decade. Substantial methodological advancements associated with the optimization and regularization of large neural networks, the availability of large data sets together with the computational power to train large networks, and last but not least the advent of powerful, easy to use software libraries, deep neural networks (DNNs) have achieved breakthrough performance in computer vision, natural language processing and other domains (LeCun et al., 2015). A feature that sets deep learning apart from conventional machine learning is the ability automatically extract discriminative features from raw data (Nielsen, 2015). Reducing the need for manual feature engineering, this feature decreases the costs of applying a learning algorithm in industry and broadens the scope of deep learning applications.

Two specific neural network structures have been particularly successful: convolutional neural networks and recurrent neural networks (RNNs). While CNNs are the model of choice for computer vision tasks, RNNs are helpful whenever sequential data need to be analyzed including natural language, written text, audio, and generally, any type of time series. The paper is concerned with recurrent neural networks and their potential for financial time series forecasting.

Given the rarity of deep learning applications in financial forecasting, the paper provides a thorough explanation of the operating principles of RNNs and how it differs from conventional feedforward neural networks (FNN). To that end, we focus on two powerful types of RNNs, Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs) that overcome the vanishing gradient problem, which rendered previous attempts to train deep RNNs infeasible. To further expand the body of knowledge in the scope of RNN-based financial forecasting, the core contribution of the paper consists of an empirical analysis of the directional accuracy and trading profitability of LSTM and GRU compared to benchmark forecasting models. We chose the foreign exchange market. In addition to the academic and practical relevance of accurate price forecasts in the foreign exchange rate market, exchange rates have been shown to be particularly difficult to predict (Wu and Chen, 1998; Czech and Waszkowski, 2012). These factors make exchange rate prediction a suitable testbed for the focal study.

The paper is organized as follows: the next section elaborates on neural network-based forecasting and introduces LSTM and GRU. Thereafter, we review related work and show how the paper contributes to closing gaps in research. Subsequently, we describe the experimental design of the forecasting comparison and report empirical results. We then conclude the paper with a summary and discussion of findings and an outlook to future research.

2 Neural Network Architectures

2.1 Feedforward Neural Networks

Neural networks consist of multiple connected layers of computational units called neurons. The network receives input signals and computes an output through a concatenation of matrix operations and nonlinear transformations. In this paper, the input represents time series data and the output a (price) forecast. Every neural network consists of one input and one output layer, and one or multiple hidden layers, whereby each layer consists of several neurons. The connections between the neurons of different layers carry a weight. Network training refers to the of tuning these weights in such a way that network output (i.e., forecast) matches the target variable as accurately as possible. The training of a neural network through adjusting connection weights is equivalent to the task of estimating a statistical model through empirical risk minimization (Härdle and Leopold, 2015).

The layer of a FNN comprise fully connected neurons without shortcuts or feedback loops. When processing sequential data, the activation of a hidden layer unit in an FNN at time t can be described as:

$$h_t = g_h(W_h^T x_t + b_h),$$

where the hidden activation function g_h is a function of the hidden weight matrix W_h, an input vector x_t, and a bias b_h. It produces the hidden activation h_t. A prediction $\hat{y}_t$ is the result of applying a suitable output activation function to the weighted sum of the activations received in the output layer. As such, the prediction is again only a function of inputs, weights, and biases.

$$\hat{y}_t = g_y(W_y^T h_t + b_y)$$

The weights and biases determine the mapping from x_t to $\hat{y}_t$ and will be learned during training. A FNN will treat input sequences as time-invariant data and thus be agnostic of inherent features of time series.At any given point in time, t, a FNN takes an input x_t and maps it to a hidden activation h_t. This is repeated at the time of the next observation, $t+1$, while the two mappings are not interconnected.

2.2 Recurrent Neural Networks

RNNs are designed for sequential data processing. To this end, they include feedback loops and fed the output signal of a neuron back into the neuron. This way, information from previous time steps is preserved as hidden state h_t and can be discounted in network predictions. When viewed over time, RNNs resemble a chain-like sequence of copies of neural networks, each passing on information to its successor. More formally, a RNN takes as additional argument the previous time step's hidden state h_{t-1}:

$$h_t = g_h(W_h[h_{t-1}, x_t] + b_h)$$

The objective of training a neural network is to minimize the value of a

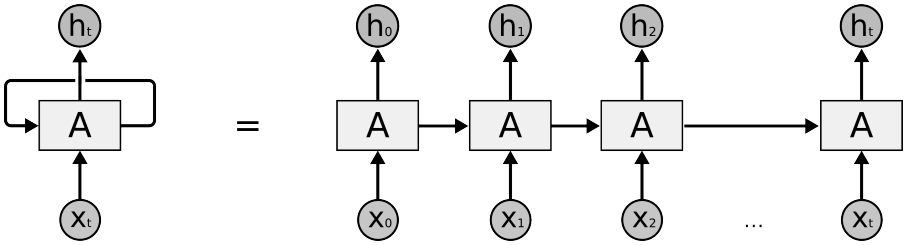

Fig. 1 A recurrent neural network architecture Olah (2015): The RNN feeds learned information back into the network via the output h_t.

loss function, which represents the cumulative difference between the model's outputs and the true labels. Since the output of a neural network is a function of the weights and biases of all its connections, the loss function value can be changed by modifying the network's weights and biases. Therefore, computing the gradient of the loss function with respect to the network's weights obtains the information needed to train the network. The backpropagation algorithm uses the insight that the gradient with respect to each weight can be found by starting at the gradient with respect to the output and then propagating the derivatives backwards through the network using the chain rule (Rumelhart et al., 1986). In RNNs, the gradient descent-based training of the network is called backpropagation through time, as the error derivatives are not only backpropagated through the network itself but also back through time via the recurrent connections (Werbos, 1990).

RNNs often use activation functions such as the hyperbolic tangent (tanh) or the logistic sigmoid (σ). The derivative of both lies in the interval $[0, 1]$ and thus any gradient with respect to a weight that feeds into such an activation function is bound to be squeezed smaller (Hochreiter, 1998b). Considering that we are successively computing the derivative of an activation function by use of the chain rule, the gradient gets smaller and smaller the further away—and in RNNs, the further back in time—-from the current output layer a weight is located. That also implies that the magnitude of weight adjustments during training decreases for those weights. Effectively, weights in early layers learn much slower than those in late hidden layers (closer to the output) (Nielsen, 2015). For large networks or many time steps, this can keep a neural network from learning and prevent it from storing information—to a point where "the vanishing error problem casts doubt on whether standard RNNs can indeed exhibit significant practical advantages over time window-based feedforward networks." (Gers et al., 1999)

2.3 Long Short-Term Memory

One solution to the vanishing gradient problem was proposed by Hochreiter and Schmidhuber (1997) in the form of LSTM. Twenty years after its invention, LSTM and its variants have turned out to become a state-of-the-art neural network architecture for sequential data. The following discussion of the LSTM cell follows Graves (2013) as it seems to be one of the most popular LSTM architectures in recent research and is also available in the widely used Python library *Keras* (Chollet et al., 2018b).

2.3.1 The Cell State

The central feature that allows LSTM to overcome the vanishing gradient problem is an additional pathway called the cell state. The cell state is a stream of information that is passed on through time. Its gradient does not vanish and enforces a constant error flow through time (Hochreiter, 1998a). The cell state allows the LSTM to remember dependencies through time and facilitates bridging long time lags (Hochreiter and Schmidhuber, 1997). Figure 2 depicts

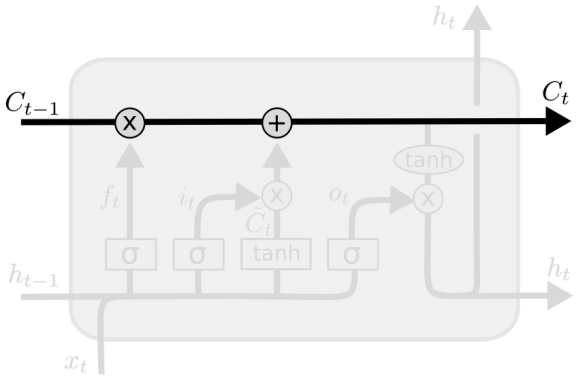

Fig. 2 A single LSTM unit (a memory block) according to (Olah, 2015) with all but the cell state pathway grayed out

a single LSTM cell with all but the cell state pathway grayed out. Note that the cell state contains no activation functions but only linear operations. In that way, it is "immune" to the vanishing gradient. The following discussion details how the cell state is maintained, updated or read.

2.3.2 Gate Units

The LSTM cell contains a number of gate structures that allow accessing the cell. A typical LSTM cell receives two inputs: the current input x_t and the recurrent input, which is the previous time step's hidden state h_{t-1}. Gating

units control how these inputs change the cell state in order to produce an updated cell state, C_t, or read from it to make use of the information that the cell state embodies.

The logistic sigmoid function, defined as $\sigma(x) = \frac{1}{1+e^{-x}}$, plays an important role in the gating mechanisms of the LSTM cell. It takes the weighted current and recurrent inputs and maps them to the interval $[0, 1]$. This enables the network to control the information flow through the gates, which also explains the term "gate". The values of 0 and 1 can be interpreted as allowing no information and all information to pass through a specific gate, respectively. In addition to the "gatekeeper" sigmoid, two of the three LSTM gates make use of the hyperbolic tangent function, defined as $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. The tanh is the usual activation function for input and output gates in the LSTM (Greff et al., 2017) and pushes its inputs into the interval $[-1, 1]$.

The logistic sigmoid and the hyperbolic tangent have relatively simple derivatives, $\frac{d}{d(x)}\sigma(x) = \sigma(x)(1 - \sigma(x))$ and $\frac{d}{d(x)}\tanh(x) = 1 - \tanh^2(x)$, which makes them a suitable choice for network training (e.g., backpropagation).

2.3.3 The Forget Gate

The forget gate f_t determines the parts of C_{t-1}, which the cell state passed on from the previous time step, that are worth remembering. As shown in Figure 3, this is achieved by means of a sigmoid gatekeeper function:

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

f_t is then multiplied with C_{t-1} to selectively allow information to remain in memory. Values of $f_t = 1$ and $f_t = 0$ imply that all information from C_{t-1} is kept and erased, respectively.

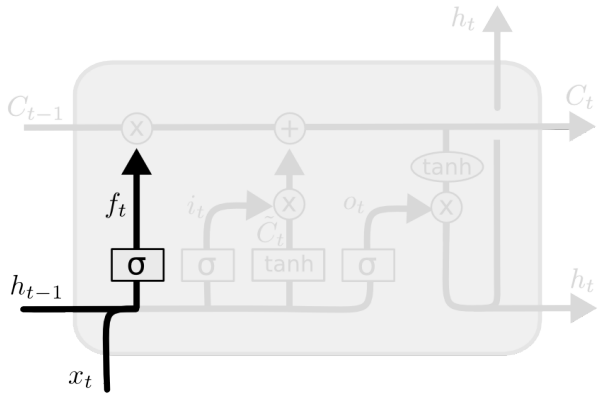

Fig. 3 The forget gate f_t is multiplied with the previous cell state C_{t-1} to selectively forget information Olah (2015).

2.3.4 The Input Gate

The input gate i_t, highlighted in Figure 4, employs a sigmoid to control information flow:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

The objective of this gate is to protect the information of the cell state, which has accumulated over previous time steps, from irrelevant updates. Therefore, the input gate selectively updates the cell state with new information (Hochreiter and Schmidhuber, 1997). To this end, a new set of candidate values $\widetilde{C}_t$ is generated by an activation function; typically a hyperbolic tangent:

$$\widetilde{C}_t = tanh(W_C[h_{t-1}, x_t] + b_C)$$

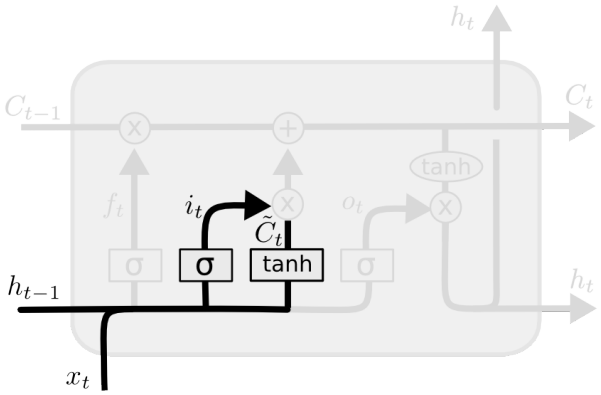

Fig. 4 The input gate i_t directs where to update the cell state with new candidate values $\widetilde{C}_t$ (Olah, 2015).

2.3.5 The Updated Cell State

Based on the mechanisms of the input and forget gate, the new cell state C_t is obtained in two ways: a part of the old cell state C_{t-1} has been remembered (via f_t) and has been updated with the new candidate values from $\widetilde{C}_t$ where needed (via i_t). This updated cell state will be relayed to the next time step, $t + 1$:

$$C_t = f_t \circ C_{t-1} + i_t \circ \widetilde{C}_t$$

Note that $i_t = f_t$ does not necessarily hold, and neither does $i_t = 1 - f_t$. It is not exactly the parts that got remembered that get updated, and not exactly the ones that were forgotten either. While both the forget gate and the input gate make use of a sigmoid as their activation function and take the same

arguments (h_{t-1} and x_t), in general their weights and biases will differ (Olah, 2015).

2.3.6 The Output Gate

The output gate steers the actual prediction of the LSTM, which is determined by both, the current input x_t and the cell state C_t. A hyperbolic tangent is applied to the values of the current cell state to produce a version of the cell state that is scaled to the interval $[-1, 1]$:

$$C_t^* = \tanh(C_t)$$

The output gate o_t consists of a sigmoid with arguments h_{t-1} and x_t, and determines which information to pass on to the output layer and subsequent time steps in the new hidden state h_t.

As shown in Figure 5,

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

o_t and C_t^* are then multiplied to construct the hidden output, h_t, of the current time step:

$$h_t = o_t \circ C_t^*$$

This output represents the recurrent input at time $t + 1$ and the basis for the prediction at time t. As in a FNN, predictions are computed from the hidden state by applying an output activation in the final layer.

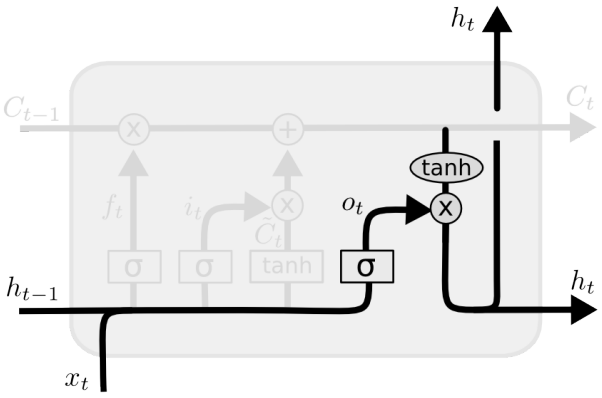

Fig. 5 The output gate o_t controls the network's predictions (Olah, 2015)

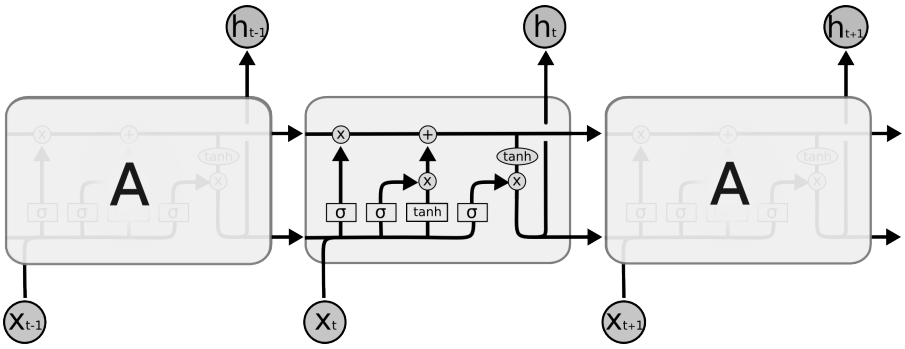

Fig. 6 A sequence of LSTM units through time (Olah, 2015)

2.3.7 The LSTM Cell

A typical LSTM cell with forget gate, input gate, and output gate is depicted in Figure 6. The different gates and activations work together to save, keep, and produce information for the task at hand. When considering the gates and cell state together as $h_t = o_t \circ \tanh(f_t \circ C_{t-1} + i_t \circ \widetilde{C}_t)$, it can be seen that h_t is essentially a sophisticated activation function:

$$\begin{aligned} h_t =& \sigma(W_o[h_{t-1}, x_t] + b_o) \circ \tanh(\sigma(W_f[h_{t-1}, x_t] + b_f) \cdot C_{t-1} + \\ &+ \sigma(W_i[h_{t-1}, x_t] + b_i) \circ \tanh(W_C[h_{t-1}, x_t] + b_C)) = \\ &= g_h(W_h, h_{t-1}, x_t) \end{aligned}$$

This architecture is an augmented version of the original LSTM architecture and the setup most common in the current literature (Greff et al., 2017). Figure 7, which depicts a sequence of LSTM cells through time, conveys how information can be transported through time via the cell state. The three gates f_t to the left of i_t under and o_t above the hidden layer unit ("—" for closed and "O" for open) control which parts of the cell state are forgotten, updated, and output at each time step.

There exist a few variants of the LSTM cell with fewer or additional components. For example, one modification concerns the use of *peephole connections*, which allow the cell state to control the gates and have been shown to increase LSTM resilience toward spikes in time series (Gers and Schmidhuber, 2000). Greff et al. (2017) perform a comprehensive evaluation of the LSTM and the marginal efficacy of individual components including several recent adaptations such as peephole connection. They start from an LSTM cell with all gates and all possible peephole connections and selectively remove one component, always testing the resulting architecture on data from several domains. The empirical results suggest that the forget gate and the output activation function seem particularly important while none out of the investigated modifications of the above LSTM cell significantly improves performance Greff

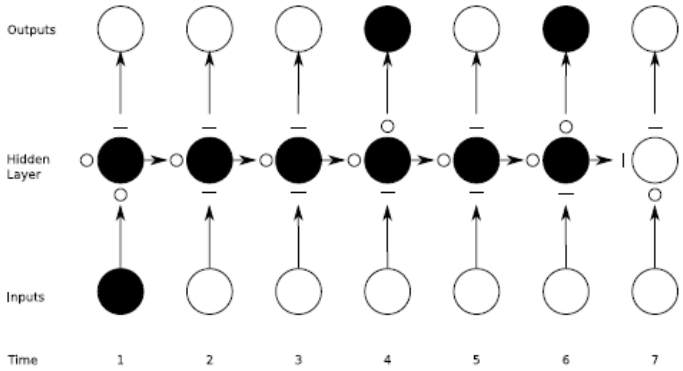

Fig. 7 Preservation of gradient information by LSTM (Graves, 2012)

et al. (2017). In view of these findings, we focus on the LSTM as described above.

2.4 Gated Recurrent Units

A second approach to overcome the vanishing gradient problem in RNNs are GRUs (Cho et al., 2014). They also use gates but simplify the handling of the cell state. The hidden state in a GRU is controlled by two sigmoid gates: an update gate couples the tasks of LSTM's forget and input gates. It decides how much of the recurrent information is kept:

$$z_t = \sigma\big(W_z[h_{t-1}, x_t] + b_z\big)$$

A reset gate controls to which extent the recurrent hidden state is allowed to feed into the current activation:

$$r_t = \sigma\big(W_r[h_{t-1}, x_t] + b_r\big)$$

A closed reset gate ($r_t = 0$) allows the memory cell to disregard the recurrent state and act as if it were reading the first observation in a sequence (Chung et al., 2014). The new activation can be computed as

$$\tilde{h}_t = \tanh\left(W_{\tilde{h}}[r_t \circ h_{t-1}, x_t] + b_{\tilde{h}}\right)$$

and the new hidden state is

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t.$$

Again, the GRU cell can be seen as a sophisticated activation function:

$$\begin{aligned} h_t =& (1 - \sigma\big(W_z[h_{t-1}, x_t] + b_z\big)) \circ h_{t-1} + \\ &+ \sigma\big(W_z[h_{t-1}, x_t] + b_z\big) \circ \tanh\left(W_{\tilde{h}}[\sigma\big(W_r[h_{t-1}, x_t] + b_r\big) \circ h_{t-1}, x_t] + b_{\tilde{h}}\right) = \\ &= g_h(W_h, h_{t-1}, x_t) \end{aligned}$$

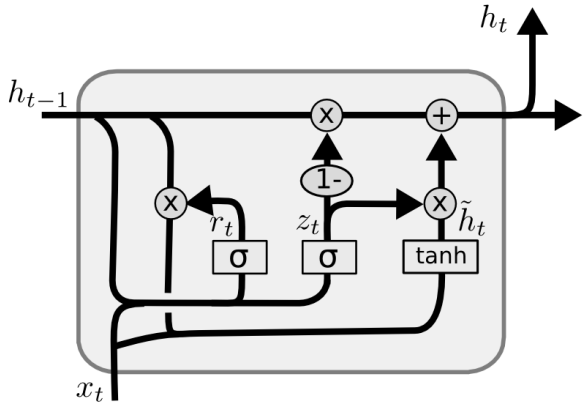

Fig. 8 A GRU cell in detail with the recurrent hidden state h_{t-1}, update gate z_t, reset gate r_t, a hidden state candidate vector $\tilde{h}_t$, and the new hidden state h_t (Olah, 2015).

There is no output activation function like in the LSTM, but the hidden cell state is bounded because of the coupling of input and forget gate in the GRU's update gate (Greff et al., 2017). Figure 8 illustrates that GRUs have less parameters than LSTMs, which should make them computationally more efficient. In terms of forecasting performance, previous results on GRUs versus LSTMs are inconclusive (see, e.g., Chung et al. (2014) versus Jozefowicz et al. (2015)). Therefore, we consider both types of RNNs in our empirical evaluation.

3 Related Work

Forecasting developments in financial markets is a well-studied research area. Starting with seminal work of Fama (1970), a large body of literature has examined the informational efficiency of financial markets. Empirical findings do not offer a clear result. Considering the foreign exchange market, for example, Wu and Chen (1998) investigate seven currency pairs and find the efficient market hypothesis (EMH) to hold, while Hakkio and Rush (1989) and Czech and Waszkowski (2012) reject the EMH for at least some of the exchange rates tested by Wu and Chen (1998). It might be because of such contradictions that a statistical modeling of market prices, volatility, and other characteristics continues to be a popular topic in the forecasting and machine learning literature. Neural networks are a particularly popular instrument for financial forecasting. For example, several studies have used FNNs to predict price movements (see, e.g, Hsu et al. (2016)). From a methodological point of view, RNNs are better suited to process sequential data (e.g., temporal financial data) than other network architectures. Therefore, we focus the review of previous literature to studies that employed RNNs for financial forecasting and summarize corresponding studies in Table 1. To depict the state-of-the-art in the field, we

consider the type of RNN as well as benchmark methods, the type of features used for forecasting, the target variable and whether a study employed a trading strategy. In addition to reporting statistical measures of forecast accuracy such as the mean-squared error, a trading strategy facilitates examining the monetary implications of trading model forecasts. The last column of Table 1 sketches the main focus of a paper such as testing the EMH or the merit of a specific modeling approach such as ensemble forecasting.

Table 1 suggests that there is no unified experimental framework. Notable differences across the financial time series considered in previous work exemplify this variation. About half of the studies adopt a univariate approach and use either historic prices, returns or transformations of these as feature. Other studies derive additional features from the time series, for example in the form of a technical indicator, or consider external sources such as prices from other financial instruments. Evaluation practices display a similar variance with roughly 50 percent of papers performing a trading evaluation and the rest focusing exclusively on forecast accuracy.

In terms of neural network architectures, studies examining RNNs in the 1990s can be seen as forerunners, with comparatively little research on the applications of RNNs available at that time. One of the earliest studies includes Kamijo and Tanigawa (1990) who use an RNN in the scope of technical stock analysis. Interestingly, Table 1 also identifies some earlier studies that examine the foreign exchange market. For example, Tenti (1996) constructs three different RNNs to predict the returns of exchange rate futures with encouraging results, while Kuan and Liu (1995) assess RNNs compared to FNNs and an ARMA model to obtain mixed results as to the superiority of the former. Giles et al. (2001) further expands these studies through examining directional currency movement forecasts in a RNN framework.

These studies predate the publication of the seminal LSTM paper by Hochreiter and Schmidhuber (1997) and use relatively short input sequences (of length smaller than 10) as features. More recent studies consider longer input sequences using memory-networks like LSTM and GRU.Xiong et al. (2015) predict the volatility of the S&P 500 and find that LSTM outperforms econometric benchmarks in the form of L1- and L2-regression as well as GARCH. Fischer and Krauss (2018) compare the performance of a single-layer LSTM against several benchmark algorithms, namely random forests, a FNN, and a logistic regression, and find that LSTM "beat[s] the standard deep networks and the logistic regression by a very clear margin" and outperforms a random forest in most periods. Shen et al. (2018) test GRUs against a FNN and a support vector machine on three financial indices, with the GRUs producing the best results.

An interesting finding of Table 1 concerns the foreign exchange market. While many earlier studies consider this market, we find no study that examines the ability of recent RNN architectures in the form of LSTM and GRU to forecast exchange rates. To the best of our knowledge, the 2008 studies of Kiani and Kastens (2008) and Hussain et al. (2008) represent the latest attempts to model foreign exchange markets using a RNN framework. This observation

inspires the focal paper. We contribute original empirical evidence through comparing different types of RNNs —a simple RNN, a LSTM, and a GRU— in terms of their ability to forecast exchange rate returns. To set observed results into context, we contrast the performance of RNNs with that of a FNN and a nave benchmark model.

Table 1 Selected studies employing recurring neural networks for financial forecasting.

Authors	Year	Data	Recurrent Neural Networks	Benchmarks	Features	Target	Trading Strategy	Discussion
Kamijo and Tanigawa (1990)	1990	Stocks: TSE (chart signals)	RNN	-	weekly prices	detection of patterns	No	-
Kuan and Liu (1995)	1995	Forex: GBP, CAD, DM, JPY, CHF vs. USD	RNN	FNN, ARMA	daily prices (1-6 days)	log returns	No	univariate >multivariate
Tenti (1996)	1996	Forex: DM/USD Futures	RNN	-	log returns, SD, technical indicators (8 out of last 34 days)	log returns	Yes	EMH, practical application
Saad et al. (1998)	1998	Stocks: various	RNN	TDNN, PNN	daily prices	detection of profit opportunities	No	-
Giles et al. (2001)	2001	Forex: DM, JPY, CHF, GBP, CAD vs. USD	RNN	FNN	symbolic encodings of differenced daily prices (3 days)	directional change	No	EMH, extraction of heuristics
Kiani and Kastens (2008)	2008	Forex: GBP, CAD, JPY vs. USD	RNN	FNN, ARMA, GSS	prices	prices	Yes	-
Hussain et al. (2008)	2008	Forex: EUR/DM, JPY, GBP vs. USD	FNN/RNN combination	FNNs	(i) prices, (ii) returns (6 days)	(i) prices, (ii) returns	Yes	EMH, inputs: prices >returns
Huck (2009)	2009	Stocks: S&P 100	RNN	-	pairs of weekly returns (3)	returns spread direction	Yes	-
Chen et al. (2015)	2015	Stocks: SSE & SZSE	LSTM	Random Prediction	up to 10 features from daily prices and volume of stock & index (30 days)	3-day earning rate (classification)	Yes	-
Rather et al. (2015)	2015	Stocks: NSE	RNN	FNN, ARMA, Exp. Smooth.	weekly returns (3 weeks)	weekly returns	No	Ensembling
Xiong et al. (2015)	2015	Index: S&P 500 (volatility)	LSTM	L1-Reg., L2-Reg., GARCH	returns, volatility & Google search trend (10 days)	volatility	No	-
Di Persio and Honchar (2017)	2017	Stock: GOOGL	RNN, LSTM, GRU	-	daily price and volume (30 days)	directional change	No	-
Fischer and Krauss (2018)	2018	Stocks: S&P 500 constituents	LSTM	FNN, Random Forest, Log. Reg.	daily returns (240 days)	above-median return (binary)	Yes	EMH, heuristics, time-variance of performance
Shen et al. (2018)	2018	Indices: HSI, DAX, S&P 500	GRU	FNN, SVM	daily returns (240 days)	directional change	Yes	-
Zhao et al. (2018)	2018	Indices: SSE, CSI, SZSE	LSTM	SVM, Random Forest, AdaBoost	time-weighted returns (4 days)	trend labeling	No	-

4 Experimental Design

4.1 Data

The empirical evaluation grounds on data of four major foreign exchange rates: the Euro (EUR), the British Pound (GBP), the Japanese Yen (JPY), and the Swiss Franc (CHF), each of which we measure against the U.S. Dollar (USD). The selection of data follows previous work in the field (Kuan and Liu, 1995; Tenti, 1996; Giles et al., 2001; Kiani and Kastens, 2008; Hussain et al., 2008). The data set consists of 12,710 rows representing daily bilateral exchange rates from January 4, 1971 until August 25, 2017. However, the time series are not of the same length. For example, the EUR was first reported in 1999 so that the EUR/USD exchange rate time series only contains 4,688 non-null observations compared to the 11,715 observations for the longest time series in the data set.

Table 2 provides a overview of the characteristics of the time series' one-day percentage returns. The exchange rates and the corresponding daily returns are also plotted in Figure 9, together with a combination of a histogram (grey bars), a kernel density estimation (black line) and a rug plot (black bars along the x-axis). From the return plots in the middle column, we observe that the transformation from returns to prices removes trends, but the return series still exhibit non-stationarity. In particular, the histograms, kernel density estimators, and rug plots indicate leptokurtic distributions, and the large kurtosis values in Table 2 support this observation.

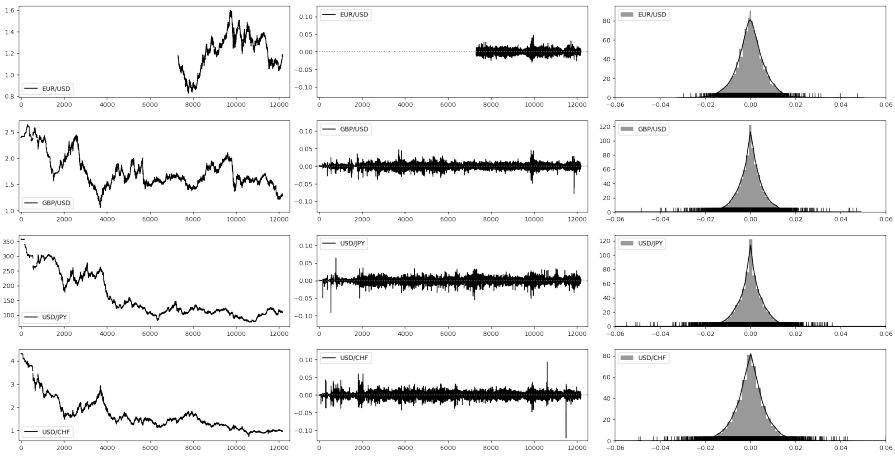

Fig. 9 Prices, one-day returns, and a combination of histograms, KDE, and rug plots of the one-day percentage returns for the four foreign exchange rate time series.

Table 2 Statistical properties of the one-day percentage returns of selected currencies.

	EUR/USD	GBP/USD	USD/JPY	USD/CHF
Observations	4687	11708	11702	11708
Mean	0.0000	-0.0000	-0.0001	-0.0001
Standard Deviation	0.0063	0.0060	0.0065	0.0073
Minimum	-0.0296	-0.0784	-0.0907	-0.1221
25 % Quantile	-0.0034	-0.0029	-0.0030	-0.0038
Median	0.0000	0.0001	0.0000	0.0000
75 % Quantile	0.0035	0.0029	0.0031	0.0036
Maximum	0.0473	0.0470	0.0646	0.0930
Skewness	0.1511	-0.3216	-0.5540	-0.2305
Kurtosis	2.2591	6.9514	8.6128	12.3697

4.2 Data preprocessing

In order to prepare the data for analysis, we divide each time series into study periods, scale the training data, and create input sequences and target variable values.

4.2.1 Features

Exchange rates represent the price of one unit of currency denominated in another currency, whereby we consider the USD as denominator.

Let P_t^c denote the price of a currency c at time t in USD. The one-day percentage return can then be calculated as the percentage change of the price from time t to the following trading day:

$$r_t^c = \frac{P_t^c}{P_{t-1}^c} - 1$$

Before model training, we scale the returns to the interval $[l, u]$ using min-max-scaling. To avoid data leakage, we perform the scaling for each study period individually, which ensures that the scaler is fitted to the training data and has no access to the trading (or test) data.

We use the time series of scaled returns as the sole feature, with the input at time t consisting of the sequence of returns of the previous τ trading days:

$$X_t^c = \{\tilde{r}_{t-\tau}^c, \tilde{r}_{t-\tau+1}^c, \tilde{r}_{t-\tau+2}^c, \ldots, \tilde{r}_{t-1}^c\}$$

4.2.2 Targets

We formulate the prediction task as a binary classification problem. The focus on directional forecasts is motivated by recent literature (Takeuchi, 2013; Fischer and Krauss, 2018). We then define the target variable Y_t^c such that values of one and zero indicate non-negative and negative returns, respectively.

$$Y_t^c = \begin{cases} 1 & \text{if } r_t^c \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

4.2.3 Length of the Input Sequences

Previous studies found foreign exchange rates to exhibit long-term memory (van de Gucht et al., 1996). This suggests the suitability of GRUs and LSTMs with their ability to store long-term information, provided they receive input sequences of sufficient length. We chose an input sequence length of τ =240, which follows from two of the most recent studies (Fischer and Krauss, 2018; Shen et al., 2018). The LSTM, GRU, and simple RNN (SRNN) that we consider as benchmark model regard each sequence of 240 observations as one single feature and make use of the relative order of data points. On the contrary, a FNN, which we also consider as benchmark, regards the 240 observations as distinct features.

4.2.4 Training and Trading Window

To test the predictive performance of different forecasting models, we employ a sliding-window evaluation, which is commonly used in previous literature (Krauss et al., 2017; Tomasini and Jaekle, 2011; Dixon et al., 2016). This approach forms several overlapping study periods, each of which contains a training and a test window. In each study period, models are estimated on the training data and generate predictions for the test data, which facilitate model assessment. Subsequently, the study period is shifted by the length of one test period as depicted in Figure 10. Such evaluation is efficient in the sense that much data is used for model training while at the same time predictions can be generated for nearly the whole time series. Only the observations in the first training window cannot be used for prediction.

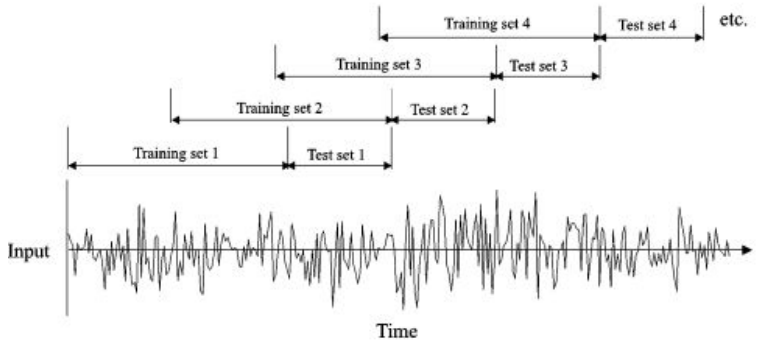

Fig. 10 Sliding window evaluation: Models are trained in isolation inside each study period, which consists of a training set and a trading (test) set. The models are trained only on the training set, predictions are made on the test set, which is out-of-sample for each study period. Then, all windows are shifted by the length of the test set to create a new study period with training set and out-of-sample test set. (From Giles et al. (2001).)

4.2.5 Loss function

The models are trained to minimize the cross-entropy between predictions and actual target values. In the binary case, the cross-entropy for an individual prediction and the corresponding target value is given by $-(y_t \log(\hat{y}_t) + (1 - y_t) \log(1 - \hat{y}_t))$, and the overall training loss is the average of the individual cross-entropy values. That way, the training process can be interpreted as a maximum likelihood optimization, since the binary cross-entropy is equal to the negative log-likelihood of the targets given the data. The loss function for study period S including trading set T_S (with cardinality $|T_S|$) is represented by L_S [1]:

$$L_S(y_{T_S}, \hat{y}_{T_S}) = -\frac{1}{|T_S|} \sum_{t \in T_S} (y_t \log(\hat{y}_t) + (1 - y_t) \log(1 - \hat{y}_t))$$

4.2.6 Activation Functions

We train the FNN using a rectified linear unit (relu) activation function: $relu(x) = \max(0, x)$Using relu activations improves gradient flow, reduces the training time (Glorot et al., 2011) and has become the state-of-the art in deep learning (LeCun et al., 2015; Clevert et al., 2016; Ramachandran et al., 2017).

For the recurrent neural networks, activation functions in the recurrent layers are applied as described in chapter 2: The SRNN uses hyperbolic tangent activations, while the LSTM and the GRU use sigmoid gates and hyperbolic tangent activations as input and output activations. More precisely, we follow Chollet et al. (2018a) and use a segment-wise linear approximation of the sigmoid function to enhance computational efficiency.

All networks use a sigmoid function as their output activation to model the conditional probability of non-negative returns given the training data, $\mathbb{P}(Y_t^c = 1 | X_t^c = \{r_{t-\tau}^c, \ldots, r_{t-1}^c\})$ Goodfellow et al. (2016).

4.2.7 Regularization

One drawback of neural networks is their vulnerability to overfitting (Srivastava et al., 2014). Regularization is a way to protect against overfitting, and can be implemented in several ways including penalizing model complexity or monitoring the model's performance on unseen data. We employ two regularization techniques:

Dropout: helps the network's neurons to generalize and avoid large co-dependence between different neurons (Hinton et al., 2012). To that end, a dropout layer randomly masks the connections between some neurons during model training. We use dropout on the non-recurrent connections after all hidden

[1] L_S simplifies to $L_S(y_{T_S}, \hat{y}_{T_S}) = -\frac{1}{|T_S|} \sum_{t \in T_S} \log(\hat{y}_t)$ in the binary classification task with labels (0,1).

layers as in Zaremba et al. (2014) with various dropout rates. For example, a dropout rate of 25 percent implies that each neuron in the previous layer is dropped with probability 25 percent; on average, a quarter of the neurons of that layer are masked.

Early stopping: refers to holding back a certain part of the training data to trace the forecasting error of a network during training (e.g., after each epoch). The validation set error enables us to stop network training conditional on the validation loss.

4.3 Hyperparameter Tuning

Neural networks and their underlying training algorithms exhibit several hyperparameters that affect model quality and forecast accuracy. Examples include the number of hidden layers and their number of neurons, the dropout rate or other regularization parameters, as well as algorithmic hyperparameters such as the learning rate, the number of epochs, the size of mini-batches, etc. (Goodfellow et al., 2016). Hyperparameter tuning is typically performed by means of empirical experimentation, which incurs a high computational cost because of the large space of candidate hyperparameter settings. We employ random search (Bengio, 2012) for hyperparameter tuning considering the following search space:

- Number of hidden layers: 1, 2, 3, 4
- Number of neurons per hidden layer: 25, 50, 100, 200, 400, 800, 1600
- Dropout: 0 to 60 percent, in steps of 10 percent
- Optimizer and learning rate: Adam and RMSprop with various learning rates
- Batch size: 16, 32, 64, 128, 256

5 Evaluation

We consider three measures of forecast accuracy: logarithmic loss (Log loss) as this loss function is minimized during network training; predictive accuracy (Acc.) as the most intuitive interpretation of classification performance; and the area under the receiver operator characteristic curve (AUC).

In addition to assessing classification performance, we employ a basic trading model to shed light on the economic implications of trading on model forecasts. The trading strategy is as follows: for each observation t in the test period, buy the currency in the numerator of the currency pair if a non-negative return is predicted with probability of at least 50 percent (and realize that day's net profit); sell that currency otherwise (and realize that day's net profit multiplied by -1). The position is held for one day. This would yield the following realized daily return $\tilde{r}_t^c$ of the basic trading model:

$$\tilde{r}_t^c = \begin{cases} r_t^c & \text{if } \hat{y}_t \geq 0.5 \\ -r_t^c & \text{otherwise} \end{cases}$$

	Acc.	AUC	Returns	SD	SR
EUR/USD	0.4744	0.4718	-0.0202	0.0060	-0.0188
GBP/USD	0.5010	0.4971	0.0481	0.0059	0.0310
USD/JPY	0.4940	0.4888	0.0488	0.0063	0.0280
USD/CHF	0.4873	0.4839	0.0131	0.0071	0.0014
Weighted Avg.	0.4921	0.4880	0.0307	0.0064	0.0161

Table 3 Results from a nave forecast by time series and aggregated (average weighted by length of time series).

As each test set consist of 240 trading days (roughly one year), the annualized net returns of this strategy in study period S are approximated by:

$$R_S = \prod_{t \in T_S} (1 + \tilde{r}_t^c) - 1$$

As a measure of risk, the standard deviation (SD) of the series of realized trading strategy returns is considered, and the Sharpe ratio (SR) is computed as a measure of risk-adjusted returns. These three metrics are used to compare the different models' predictions economically.

6 Empirical Results

In order to set results of different neural networks models into context, we compute a nave benchmark forecast the prediction of which at time t simply equals the true target at time $t-1$:

$$\hat{y}_t = y_{t-1}$$

The results of this benchmark can be found in table 3, both per time-series as well as aggregated across time series. Note that we cannot compute the log loss for this benchmark since log(0) is undefined and the nave benchmark predicts 0 whenever the previous day's true returns are negative.

The nave forecast gets the predictions right a bit under half the time and if the trading strategy defined in section **??** were applied, it would result in small positive net returns.

Results from training the FNN, SRNN, LSTM, and GRU on the four selected foreign exchange rate time series are displayed in Table 4 and visualized in Figure 11 by means of violin plots.

Table 4 suggests three conclusions. First, in terms of the training loss (Log Loss), the gated recurrent networks LSTM and GRUs perform slightly better than the FNN and SRNN for each time series. This general observation also holds roughly true for the accuracy for three of the four time series, but not for the EUR/USD exchange rate. Second, economic measures of forecast

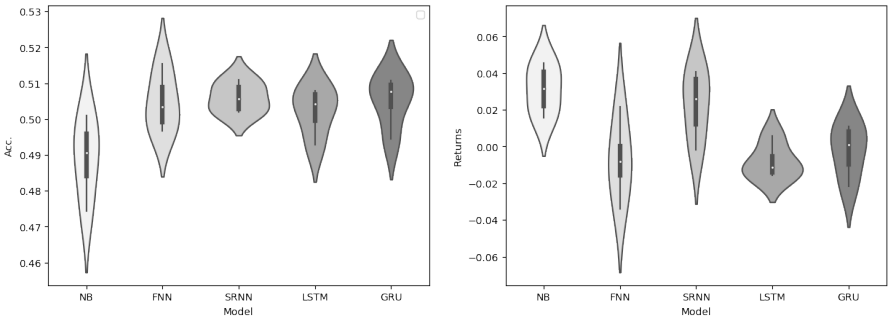

Fig. 11 Accuracy and trading strategy returns of the nave benchmark and the four deep learning models.

Time Series	Model	Log Loss	Acc.	AUC	Returns	SD	SR
EUR/USD	FNN	0.6953	0.5155	0.5202	0.0218	0.0060	0.0186
	SRNN	0.7114	0.5019	0.5003	0.0406	0.0060	0.0240
	LSTM	0.6948	0.4928	0.5005	-0.0138	0.0060	-0.0073
	GRU	0.6948	0.4944	0.5103	-0.0216	0.0060	-0.0131
GBP/USD	FNN	0.6964	0.5068	0.5035	-0.0094	0.0059	-0.0034
	SRNN	0.7064	0.5110	0.5116	0.0166	0.0059	0.0098
	LSTM	0.6943	0.5066	0.5021	-0.0088	0.0059	-0.0041
	GRU	0.6945	0.5064	0.4930	-0.0056	0.0059	-0.0021
USD/JPY	FNN	0.7001	0.4966	0.4995	-0.0340	0.0063	-0.0255
	SRNN	0.7100	0.5030	0.4955	-0.0019	0.0063	-0.0081
	LSTM	0.6956	0.5019	0.5077	-0.0157	0.0063	-0.0143
	GRU	0.6945	0.5091	0.5089	0.0075	0.0038	0.0092
USD/CHF	FNN	0.6977	0.4999	0.4982	-0.0068	0.0071	-0.0019
	SRNN	0.7016	0.5081	0.5057	0.0356	0.0071	0.0196
	LSTM	0.6936	0.5079	0.5080	0.0056	0.0071	0.0044
	GRU	0.6941	0.5108	0.5109	0.0108	0.0071	0.0057
Weighted Avg.	FNN	0.7026	0.5062	0.5061	-0.0126	0.0064	-0.0071
	SRNN	0.7115	0.5103	0.5073	0.0195	0.0064	0.0090
	LSTM	0.6993	0.5076	0.5088	-0.0072	0.0064	-0.0050
	GRU	0.6992	0.5107	0.5085	0.0014	0.0057	0.0024

Table 4 Results for the four neural networks by currency pair and model type.

performance paint a different picture. None of the models is able to produce a large positive return. Both in terms of returns and risk-adjusted returns, the SRNN performs competitive and not inferior to more advanced network architectures in form of GRU and LSTM. This is an interesting result in that several previous forecast comparisons observe a different result. We discuss the ramifications of our results in Section 5. Third, the deep learning models all perform better than the benchmark in terms of accuracy and area under

the ROC curve. However, the net returns resulting from applying the selected trading strategy are smaller in most cases.

7 Conclusion

The paper reported results from an empirical comparison of employing different deep learning frameworks for financial time series prediction. One insight of the analysis was that exchange rates are highly non-stationary. Even training in a rolling window setting cannot always ensure that training and trading set follow the same distribution. Another characteristic of the studied exchange rate was their leptokurtic distribution of returns. For example, the average kurtosis of the chosen exchange rate returns in this study is 8.60 compared to 2.01 for the stock returns in Fischer and Krauss (2018). This resulted in many instances of returns close to zero and few, but relatively large deviations and could have lead to the models exhibiting low confidence in their predictions.

The results, in term of predictive accuracy, are in line with other studies Fischer and Krauss (2018). However, they exhibit a large discrepancy between the training loss performance and economic performance of the models. This becomes especially apparent in Figure 11. When the models were trained, the assumption was that there existed a strong correlation between training loss and accuracy as well as profits. The detected gap between statistical and economic results suggests that this assumption is not true. Leitch and Tanner (1991) find that only a weak relationship exists between statistical and economic measures of forecasting performance for predictions (regression) of interest rates. A similar problem might exist between the log loss minimized during training and the trading strategy returns in this study.

Hyperparameter tuning in the chosen experimental setting turned out to be cumbersome. The dynamic training approach described in Section 4 and depicted in Figure 10 has one huge drawback: 576 individual models are trained on very limited training data each. Applying the same hyperparameters to whole time series that last over 46 years (18 in the EUR/USD case) constrains the models' capacity. Isolated hyperparameter tuning for each study period would be desirable but is not feasible in this setting as it included 144 such study periods (15 study periods for the EUR/USD series and 43 each for the GBP/EUR, USD/JPY, and USD/CHF series.)

As any empirical study, the paper exhibits limitations and these could be addressed in future research. One way of addressing the issue of low confidence predictions could be to use scaled prices as inputs, either with the same targets as in this experiment or to predict price levels in a regression and then transform the outputs to binary predictions by comparing them to the previous day's price. Hussain et al. (2008) find scaled prices as inputs slightly outperform scaled returns, but the majority of the literature uses returns.

From the discrepancy between statistical and economic results it becomes clear that a more advanced trading strategy needs to be developed if the goal is the model's application for maximization of (risk-adjusted) profits. One

example of such a trading strategy is the work of Fischer and Krauss (2018) who construct a strategy only trading a number of top and bottom pairs from a large set of 500 binary predictions on stock performance. This particular strategy would, of course, require training on many more time series. A possible solution for better interaction between model and economic performance is furthermore to develop a combination of a custom loss function and suitable output activation function instead of using binary cross-entropy with a sigmoid output activation function. That way, the model could directly optimize for either returns or risk-adjusted returns. A rough idea would be to use a weighted tanh output activation to simultaneously predict the relative size and direction of the trade[2] in combination with using the negative of either trading strategy returns or Sharpe ratio as loss function.

For better hyperparameter tuning, the obvious solution is to focus on a single (or very few) model(s) and time series. This is often the case in practical applications but might have lower scientific value. Efforts to automate large deep learning processes are under way Feurer et al. (2015), but tuning a large number of individual models remains computationally costly.

Lastly, an often applied technique to improve forecasting is ensembling. Here, many different models are trained and their individual predictions are combined either by (weighted) averaging or using a so-called meta-learner on top of the first level predictions, which determines the share of each individual models in the ensemble. Since the goal of this study was to explore recurrent neural networks in particular, ensembling was not pursued, but it should certainly be part of any practical application.

LSTM and GRUs have become the state-of-the-art in many fields Vaswani et al. (2017) and are still developed further to improve certain aspects or apply them to very specific problems. A number of recent proposals for prediction of sequential data augments or even aims to supplant recurrent neural networks.

Such expansions of recurrent neural network methods include combining RNNs with convolutional neural networks when the data is both spatial and temporal Karpathy and Li (2014) or even applying image classification to plots of time series data; giving models access to an external memory bank (Neural Turing Machine(s) Graves et al. (2014)); employing recurrent encoder-decoder structures, or modeling time dependencies in a non-recurrent way Vaswani et al. (2017).

Machine learning research is moving increasingly fast and new ideas for improvements or augmentations of algorithms keep appearing. On the other hand, some technologies become practical only many years after their emergence. The best example o this is LSTM, an algorithm that was little appreciated in the first decade of its life but is one of the cornerstones of machine learning another ten years later. It is intriguing to imagine what might be possible in another decade.

[2] The values in $[-1, 1]$ would imply trades between going short with 100 percent of the portfolio and going long with the whole portfolio in the other extreme.

References

Chollet et al F (2018a) Keras TensorFlow Backend (https://github.com/keras-team/keras/blob/master/keras/backend/tensorflow_backend.py#L1487)

Chollet et al F (2018b) Keras: The Python Deep Learning library. Astrophysics Source Code Library p ascl:1806.022, URL http://adsabs.harvard.edu/abs/2018ascl.soft06022C

Bengio Y (2012) Practical Recommendations for Gradient-Based Training of Deep Architectures. In: Montavon G, Orr GB, Mller KR (eds) Neural Networks: Tricks of the Trade: Second Edition, Lecture Notes in Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 437–478, DOI 10.1007/978-3-642-35289-$8_26, URLhttps://doi.org/10.1007/978-3-642-35289-8_26$

Chen K, Zhou Y, Dai F (2015) A LSTM-based method for stock returns prediction: A case study of China stock market. In: 2015 IEEE International Conference on Big Data (Big Data)(BIG DATA), pp 2823–2824

Cho K, van Merrienboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv:14061078 [cs, stat] URL http://arxiv.org/abs/1406.1078, arXiv: 1406.1078

Chung J, Gulcehre C, Cho K, Bengio Y (2014) Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv:14123555 [cs] URL http://arxiv.org/abs/1412.3555, arXiv: 1412.3555

Clevert DA, Unterthiner T, Hochreiter S (2016) Fast and accurate deep network learning by exponential linear units (ELUS). Archive pre-print p 15

Czech KA, Waszkowski A (2012) Foreign exchange market efficiency: Empirical results for the USD/EUR market. e-Finanse: Financial Internet Quarterly 8(3):1–9, URL https://www.econstor.eu/handle/10419/147044

Di Persio L, Honchar O (2017) Recurrent neural networks approach to the financial forecast of Google assets. International Journal of Mathematics and Computers in Simulation 11:7–13

Dixon MF, Klabjan D, Bang J (2016) Classification-Based Financial Markets Prediction Using Deep Neural Networks. SSRN Scholarly Paper ID 2756331, Social Science Research Network, Rochester, NY, URL https://papers.ssrn.com/abstract=2756331

Fama EF (1970) Efficient Capital Markets: A Review of Theory and Empirical Work*. The Journal of Finance 25(2):383–417, DOI 10.1111/j.1540-6261.1970.tb00518.x, URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.1970.tb00518.x

Feurer M, Klein A, Eggensperger K, Springenberg J, Blum M, Hutter F (2015) Efficient and Robust Automated Machine Learning. In: Cortes C, Lawrence ND, Lee DD, Sugiyama M, Garnett R (eds) Advances in Neural Information Processing Systems 28, Curran Associates, Inc., pp 2962–2970

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

Gers F, Schmidhuber J (2000) Recurrent nets that time and count. In: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium, IEEE, Como, Italy, pp 189–194 vol.3, DOI 10.1109/IJCNN.2000.861302, URL http://ieeexplore.ieee.org/document/861302/

Gers FA, Schmidhuber J, Cummins F (1999) Learning to Forget: Continual Prediction with LSTM. Neural Computation 12:2451–2471

Giles CL, Lawrence S, Tsoi AC (2001) Noisy Time Series Prediction using Recurrent Neural Networks and Grammatical Inference. Machine Learning 44(1):161–183, DOI 10.1023/A:1010884214864, URL https://doi.org/10.1023/A:1010884214864

Glorot X, Bordes A, Bengio Y (2011) Deep Sparse Rectifier Neural Networks. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp 315–323, URL http://proceedings.mlr.press/v15/glorot11a.html

Goodfellow I, Bengio Y, Courville A (2016) Deep Learning. MIT Press, URL http://www.deeplearningbook.org

Graves A (2012) Supervised Sequence Labelling. In: Graves A (ed) Supervised Sequence Labelling with Recurrent Neural Networks, Studies in Computational Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 5–13, DOI 10.1007/978-3-642-24797-$2_2, URL https://doi.org/10.1007/978-3-642-24797-2_2$

Graves A (2013) Generating Sequences With Recurrent Neural Networks. arXiv preprint arXiv:13080850 URL http://arxiv.org/abs/1308.0850, arXiv: 1308.0850

Graves A, Wayne G, Danihelka I (2014) Neural Turing Machines. arXiv:14105401 [cs] URL http://arxiv.org/abs/1410.5401, arXiv: 1410.5401

Greff K, Srivastava RK, Koutnk J, Steunebrink BR, Schmidhuber J (2017) LSTM: A Search Space Odyssey. IEEE Transactions on Neural Networks and Learning Systems 28(10):2222–2232, DOI 10.1109/TNNLS.2016.2582924, URL http://arxiv.org/abs/1503.04069, arXiv: 1503.04069

van de Gucht LM, Dekimpe MG, Kwok CCY (1996) Persistence in foreign exchange rates. Journal of International Money and Finance 15(2):191–220, DOI 10.1016/0261-5606(96)00001-0, URL http://www.sciencedirect.com/science/article/pii/0261560696000010

Hakkio CS, Rush M (1989) Market efficiency and cointegration: an application to the sterling and deutschemark exchange markets. Journal of International Money and Finance 8(1):75–88, DOI 10.1016/0261-5606(89)90015-6, URL http://linkinghub.elsevier.com/retrieve/pii/0261560689900156

Härdle WK, Leopold S (2015) Applied Multivariate Statistical Analysis, 4th edn. Springer

Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR (2012) Improving neural networks by preventing co-adaptation of feature detectors. arXiv:12070580 [cs] URL http://arxiv.org/abs/1207.0580, arXiv: 1207.0580

Hochreiter S (1998a) Recurrent Neural Net Learning and Vanishing Gradient. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6(2):107–116

Hochreiter S (1998b) The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 06(02):107–116, DOI 10.1142/S0218488598000094, URL http://www.worldscientific.com/doi/abs/10.1142/S0218488598000094

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

Hsu MW, Lessmann S, Sung MC, Ma T, Johnson JE (2016) Bridging the divide in financial market forecasting: machine learners vs. financial economists. Expert Systems with Applications 61:215–234, DOI 10.1016/j.eswa.2016.05.033, URL http://linkinghub.elsevier.com/retrieve/pii/S0957417416302585

Huck N (2009) Pairs selection and outranking: An application to the S&P 100 index. European Journal of Operational Research 196(2):819–825, DOI 10.1016/j.ejor.2008.03.025, URL http://www.sciencedirect.com/science/article/pii/S0377221708003160

Hussain AJ, Knowles A, Lisboa PJG, El-Deredy W (2008) Financial time series prediction using polynomial pipelined neural networks. Expert Systems with Applications 35(3):1186–1199, DOI 10.1016/j.eswa.2007.08.038, URL http://www.sciencedirect.com/science/article/pii/S0957417407003442

Jozefowicz R, Zaremba W, Sutskever I (2015) An empirical exploration of recurrent network architectures. In: Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37, JMLR.org, ICML'15, pp 2342–2350, URL http://dl.acm.org/citation.cfm?id=3045118.3045367

Kamijo K, Tanigawa T (1990) Stock price pattern recognition-a recurrent neural network approach. In: 1990 IJCNN International Joint Conference on Neural Networks, IEEE, San Diego, CA, USA, pp 215–221 vol.1, DOI 10.1109/IJCNN.1990.137572, URL http://ieeexplore.ieee.org/document/5726532/

Karpathy A, Li FF (2014) Deep Visual-Semantic Alignments for Generating Image Descriptions. arXiv:14122306 [cs] URL http://arxiv.org/abs/1412.2306, arXiv: 1412.2306

Kiani KM, Kastens TL (2008) Testing Forecast Accuracy of Foreign Exchange Rates: Predictions from Feed Forward and Various Recurrent Neural Network Architectures. Computational Economics 32(4):383–406, DOI 10.1007/s10614-008-9144-4, URL https://doi.org/10.1007/s10614-008-9144-4

Krauss C, Do XA, Huck N (2017) Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500. European Journal of Operational Research 259(2):689–702, DOI 10.1016/j.ejor.2016.10.031, URL http://www.sciencedirect.com/science/article/pii/S0377221716308657

Kuan CM, Liu T (1995) Forecasting exchange rates using feedforward and recurrent neural networks. Journal of Applied Econometrics 10(4):347–364

LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444, DOI 10.1038/nature14539, URL https://www.nature.com/articles/nature14539

Leitch G, Tanner JE (1991) Economic Forecast Evaluation: Profits Versus the Conventional Error Measures. The American Economic Review 81(3):580–590, URL https://www.jstor.org/stable/2006520

Nielsen MA (2015) Neural Networks and Deep Learning. Determination Press, URL http://neuralnetworksanddeeplearning.com

Olah C (2015) Understanding LSTM Networks (http://colah.github.io/posts/2015-08-Understanding-LSTMs/). URL http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Ramachandran P, Zoph B, Le QV (2017) Searching for Activation Functions. arXiv:171005941 [cs] URL http://arxiv.org/abs/1710.05941, arXiv: 1710.05941

Rather AM, Agarwal A, Sastry VN (2015) Recurrent neural network and a hybrid model for prediction of stock returns. Expert Systems with Applications 42(6):3234–3241, DOI 10.1016/j.eswa.2014.12.003, URL http://www.sciencedirect.com/science/article/pii/S0957417414007684

Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323(6088):533–536, DOI 10.1038/323533a0, URL https://www.nature.com/articles/323533a0

Saad E, Prokhorov D, Wunsch D (1998) Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks. IEEE Transactions on Neural Networks 9(6):1456–1470

Shen G, Tan Q, Zhang H, Zeng P, Xu J (2018) Deep Learning with Gated Recurrent Unit Networks for Financial Sequence Predictions. Procedia Computer Science 131:895–903, DOI 10.1016/j.procs.2018.04.298, URL http://www.sciencedirect.com/science/article/pii/S1877050918306781

Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. J Mach Learn Res 15(1):1929–1958, URL http://portal.acm.org/citation.cfm?id=2670313

Takeuchi L (2013) Applying Deep Learning to Enhance Momentum Trading Strategies in Stocks. Technical Report, Stanford University

Tenti P (1996) Forecasting foreign exchange rates using recurrent neural networks. Applied Artificial Intelligence 10(6):567–582, DOI 10.1080/088395196118434, URL http://www.tandfonline.com/doi/abs/10.1080/088395196118434

Tomasini E, Jaekle U (2011) Trading systems: a new approach to system development and portfolio optimisation, reprinted edn. Harriman House, Petersfield, oCLC: 934736951

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is All you Need. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds) Advances in Neural Information Processing Systems 30, Curran Associates, Inc., pp 5998–6008, URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Werbos PJ (1990) Backpropagation through time: what it does and how to do it. Proceedings of the IEEE 78(10):1550–1560, DOI 10.1109/5.58337

Wu JL, Chen SL (1998) Foreign exchange market efficiency revisited. Journal of International Money and Finance 17(5):831–838, DOI 10.1016/S0261-5606(98)00028-X, URL http://linkinghub.elsevier.com/retrieve/pii/S026156069800028X

Xiong R, Nichols EP, Shen Y (2015) Deep Learning Stock Volatility with Google Domestic Trends. arXiv:151204916 [q-fin] URL http://arxiv.org/abs/1512.04916, arXiv: 1512.04916

Zaremba W, Sutskever I, Vinyals O (2014) Recurrent Neural Network Regularization. arXiv:14092329 [cs] URL http://arxiv.org/abs/1409.2329

Zhao Z, Rao R, Tu S, Shi J (2018) Time-Weighted LSTM Model with Redefined Labeling for Stock Trend Prediction. In: 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI), pp 1210–1217, DOI 10.1109/ICTAI.2017.00184, URL doi.ieeecomputersociety.org/10.1109/ICTAI.2017.00184

IRTG 1792 Discussion Paper Series 2019

For a complete list of Discussion Papers published, please visit http://irtg1792.hu-berlin.de.

001 "Cooling Measures and Housing Wealth: Evidence from Singapore" by Wolfgang Karl Härdle, Rainer Schulz, Taojun Xie, January 2019.

002 "Information Arrival, News Sentiment, Volatilities and Jumps of Intraday Returns" by Ya Qian, Jun Tu, Wolfgang Karl Härdle, January 2019.

003 "Estimating low sampling frequency risk measure by high-frequency data" by Niels Wesselhöfft, Wolfgang K. Härdle, January 2019.

004 "Constrained Kelly portfolios under alpha-stable laws" by Niels Wesselhöfft, Wolfgang K. Härdle, January 2019.

005 "Usage Continuance in Software-as-a-Service" by Elias Baumann, Jana Kern, Stefan Lessmann, February 2019.

006 "Adaptive Nonparametric Community Detection" by Larisa Adamyan, Kirill Efimov, Vladimir Spokoiny, February 2019.

007 "Localizing Multivariate CAViaR" by Yegor Klochkov, Wolfgang K. Härdle, Xiu Xu, March 2019.

008 "Forex Exchange Rate Forecasting Using Deep Recurrent Neural Networks" by Alexander J. Dautel, Wolfgang K. Härdle, Stefan Lessmann, Hsin-Vonn Seow, March 2019.

IRTG 1792, Spandauer Strasse 1, D-10178 Berlin
http://irtg1792.hu-berlin.de

This research was supported by the Deutsche Forschungsgemeinschaft through the IRTG 1792.