Institut de Recerca en Economia Aplicada Regional i Pública
Research Institute of Applied Economics
Document de Treball 2022/01, 44 pàg.
Working Paper 2022/01, 44 pag.

Grup de Recerca Anàlisi Quantitativa Regional
Regional Quantitative Analysis Research Group
Document de Treball 2022/01, 44 pàg.
Working Paper 2022/01, 44 pag.

"An application of deep learning for exchange rate forecasting"

Oscar Claveria, Enric Monte, Petar Sorić and Salvador Torra

UBIREA
Research Institute of Applied Economics
AQR
Regional Quantitative Analysis Research Group

UBIREA | Institut de Recerca en Economia Aplicada Regional i Pública
UNIVERSITAT DE BARCELONA

WEBSITE: www.ub-irea.com • CONTACT: irea@ub.edu

AQR
Grup de Recerca Anàlisi Quantitativa Regional
Regional Quantitative Analysis Research Group

WEBSITE: www.ub.edu/aqr/ • CONTACT: aqr@ub.edu

Universitat de Barcelona
Av. Diagonal, 690 • 08034 Barcelona

The Research Institute of Applied Economics (IREA) in Barcelona was founded in 2005, as a research institute in applied economics. Three consolidated research groups make up the institute: AQR, RISK and GiM, and a large number of members are involved in the Institute. IREA focuses on four priority lines of investigation: (i) the quantitative study of regional and urban economic activity and analysis of regional and local economic policies, (ii) study of public economic activity in markets, particularly in the fields of empirical evaluation of privatization, the regulation and competition in the markets of public services using state of industrial economy, (iii) risk analysis in finance and insurance, and (iv) the development of micro and macro econometrics applied for the analysis of economic activity, particularly for quantitative evaluation of public policies.

Abstract

This paper examines the performance of several state-of-the-art deep learning techniques for exchange rate forecasting (deep feedforward network, convolutional network and a long short-term memory). On the one hand, the configuration of the different architectures is clearly detailed, as well as the tuning of the parameters and the regularisation techniques used to avoid overfitting. On the other hand, we design an out-of-sample forecasting experiment and evaluate the accuracy of three different deep neural networks to predict the US/UK foreign exchange rate in the days after the Brexit took effect. Of the three configurations, we obtain the best results with the deep feedforward architecture. When comparing the deep learning networks to time-series models used as a benchmark, the obtained results are highly dependent on the specific topology used in each case. Thus, although the three architectures generate more accurate predictions than the time-series models, the results vary considerably depending on the specific topology. These results hint at the potential of deep learning techniques, but they also highlight the importance of properly configuring, implementing and selecting the different topologies.

JEL Classification: C45, C58, E47, F31, G17.

Keywords: Forecasting, Exchange rates, Deep learning, Deep neural networks, Convolutional networks, Long short-term memory.

Oscar Claveria (corresponding author): AQR–IREA, University of Barcelona (UB). Department of Econometrics, Statistics and Applied Economics, University of Barcelona, Diagonal 690, 08034 Barcelona, Spain. Tel.: +34-934021825. Email: oclaveria@ub.edu

Enric Monte: Department of Signal Theory and Communications, Polytechnic University of Catalunya (UPC). Email: enric.monte@upc.edu

Petar Sorić: Faculty of Economics & Business University of Zagreb. Email: psoric2@net.efzg.hr

Salvador Torra: Riskcenter–IREA, University of Barcelona (UB). Email: storra@ub.edu

Acknowledgements and funding

This research was supported by the project PID2020-118800GB-I00 from the Spanish Ministry of Science and Innovation (MCIN) / Agencia Estatal de Investigación (AEI). DOI: http://dx.doi.org/10.13039/501100011033

1. Introduction

The exchange rate is a key macroeconomic variable. The fact exchange rates are used to determine the international competitiveness of a country, has made the forecast of their movements a relevant factor for both economic agents and policy makers. The predictability of exchange rates came to the fore since Meese and Rogoff (1983) showed that a random walk was frequently found to generate better exchange rate forecasts than economic models. This came to be known as the Meese-Rogoff puzzle. In her seminal paper, Rossi (2013) provided a critical review of the literature on exchange rate forecasting and found that the choice of predictor, the forecast horizon, the sample period, the model and the forecast evaluation method were key factors in determining the predictability of exchange rates.

Most of the vast literature related to exchange rate forecasting is centred around the exchange rate models (Jaworski, 2021). Notwithstanding, the results obtained by Meese and Rogoff (1983) has limited the number of studies focused on univariate time series. The development of new forecasting techniques in the last few years, especially in the field of machine learning (ML), and more specifically since the deep learning (DL) revolution fostered by advances in hardware in recent years, has caused a renewed interest exchange rate forecasting. This new scenario has led us to consider to what extent state-of-the-art DL architectures could improve the forecast accuracy of exchange rates predictions. We have assessed the performance of three different DL neural networks and compared them to different time-series models used as a benchmark. The three architectures we evaluate are a deep feed-forward network (DFNN), a convolutional neural network (CNN), and a long short-term memory (LSTM) network, which can be regarded as a particular type of recurrent neural network (RNN).

As opposed to most of the previous literature on this topic, which is focused on exchange rate models, in this paper we adopt a univariate time-series approach. For a recent assessment of statistical learning models to estimate exchange rate models see Colombo and Pelagatti (2020). In their study, the authors used regularised regression splines, random forests and support vector machines to provide a better understanding of the relationship between the exchange rate and economic fundamentals.

The remainder of the paper is structured as follows. Next section reviews the most recent literature on the application of DL networks with forecasting purposes. Section 3 presents the different DL architectures used in this study, their configuration and how the parameters are tuned. Section 4 describes the data and the design of the experiment. Empirical results are provided in Section 5. Finally, some concluding remarks are presented.

2. Literature review

The advent of DL has rekindled the interest for NN forecasting. The advances over the past two decades have proven the potential of NNs for time series forecasting, especially in situations where long series are available (Andrawis *et al.*, 2011; Ben Taieb *et al.*, 2012; Claveria *et al.*, 2015, 2016, 2017, 2020; Crone *et al.*, 2011; Feng & Zhang, 2014). In this context, more complex models can be estimated without being prone to overfitting. As noted by Bandara et al. (2020), recurrent architectures are naturally suited for modelling problems that need to capture the dependency in a sequential context, and are able to preserve knowledge while advancing through the subsequent time steps. These features, as well as their data-driven nature and their ductility for modelling tasks enable these methods to uncover complex relationships and perform forecasts without imposing restrictions regarding the underlying data generating process of the data.

As a result, several NN architectures such as multi-layer perceptron (MLP) and radial basis function (RBF) networks, which can be regarded as a special class of multi-layer feed-forward architecture with two layers of processing, have been widely used for economic forecasting (Aminian *et al.*, 2006; Claveria & Torra, 2014; Teräsvirta *et al.*, 2005; Stasinakis *et al.* 2014). A variation of RBF architectures are generalised regression neural networks (GRNNs), which were proposed by Specht (1991) and are also increasingly used as an alternative to MLP (Yan, 2012; Zimmermann et al., 2012).

Contrary to feed-forward networks, RNNs are models with a bidirectional data flow. While a feed-forward network propagates data linearly from input to output, RNNs also propagate temporal feedback from the outer layers to the lower layers. There are many recurrent architectures: fully recurrent, simple recurrent, bidirectional recurrent, etc. A special case of recurrent networks, which have been widely implemented are the Elman networks (Elman, 1990). Recently, LSTMs, are gaining interest, as they have proven particularly useful for forecasting (He *et al.*, 2021; Hewamalage *et al.*, 2021).

DL allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction, discovering intricate structures in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer (LeCun *et al.*, 2015). For a comprehensive review of DL in NNs, see Schmidhuber (2015). The need of long time series has made DL architectures increasingly popular in domains such as Natural Language Processing (Hinton *et al.*, 2012; Mikolov *et al.*, 2010), machine translation (Sutskever *et al.*, 2014), and speech recognition (Graves *et al.*, 2013). These methods are recently gaining popularity in time series research in different fields related to economics (Alaminos *et al.*, 2021; Bi *et al.*, 2021), especially in finance (Sezer *et al.*, 2020).

Exchange rates are high-frequency time series data that are easily available. Forecasting exchange rates has always been a topic of interest, both for researchers and analysists operating in the market. In her seminal paper, Rossi (2013) reviewed the methodologies and fundamentals proposed in the literature up to that point, and found that the predictability of exchange rates was conditional on several aspects. In spite of the the results of Meese and Rogoff (1983), who found that exchange rates could not be predicted on the basis of past exchange rate changes, recent advances in predictive techniques are refining exchange rate forecasts, thereby reopening the debate regarding the unpredictability of exchange rates (Alvarez-Diaz, 2008; Caporale & Spagnolo, 2004; Clements & Smith, 2001; Enders & Pascalau, 2015; Fernández-Rodríguez *et al.*, 2004; Gharleghi *et al.*, 2014; Gradojevic & Yang, 2006; Hong & Lee, 2003; Jamal & Sundar, 2011; Kiani & Kastens, 2008; Kirikos, 2000; Lee & Chen, 2006; Lin *et al.*, 2012).

Building on this premise that exchange rates exhibit pronounced mean-reversion, Ca' Zorzi and Rubaszek (2020) applied a battery of models both in a univariate time series setup and in a panel framework, finding that calibrated model and gradual adjustment autoregressive specifications produced more accurate predictions than the random walk or standard multi-step iterative autoregressive methods. Tripathi *et al.* (2021) used ensemble modelling for forecasting exchange rates of major currency pairs. Their model allowed combining univariate forecasts based on mean forecasting, NN and ARIMA modelling. The obtained results outperformed univariate specifications in most cases.

ML strategies for time series forecasting, which are particularly indicated for dealing with non-linearities, have opened up new possibilities for refining the prediction of exchange rates. Parot *et al.* (2019), Sermpinis *et al.* (2012, 2013), Ni and Ying (2009),

Yu *et al.* (2005), Nag and Mitra (2002), Zhang and Berardi (2001), and Lisi and Sciavo (1999) made exchange rates predictions using different types of NN architectures. Notwithstanding, as noted by Yilmaz and Arabaci (2021), the number of formal comparative studies in terms of exchange rate forecasting with DL models is quite limited. To cover this deficit the authors compared different time-series models and DNNs to predict three major exchange rate returns, finding evidence against the Meese-Rogoff puzzle by combining a linear time-series model with a LSTM recurrent network.

Chen *et al.* (2021) recently applied a LSTM network for exchange rate forecasting and obtained better forecasting results than with other ML procedures such as SVR and genetic algorithms. The authors developed a two-stage ML model for exchange rate prediction by introducing a preselected set of predictors in a LSTM exchange rate model to predict the Bitcoin exchange rate regardless of the previous exchange rate.

Similarly, Lara-Benítez *et al.* (2021) undertook a thorough comparison of different DL configurations for different datasets, including the *ExchangeRate* dataset, which records the daily exchange rate of eight countries of the OECD. The authors obtained the most accurate results with LSTM architectures.

Dautel *et al.* (2020) examined the potential of DL for exchange rate forecasting by systematically comparing LSTM networks and gated recurrent units (GRUs) 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 indicated the suitability of deep networks for exchange rate forecasting in general but also evidenced the difficulty of implementing and tuning corresponding architectures.

It is this last finding that led us to evaluate the performance of LSTM and other two DL techniques for exchange rate forecasting, making especial emphasis on the configuration of the architectures. Thus, in order to facilitate the implementation of the selected DL networks, in the next section we explicitly explain how the architectures have been designed, tuned and estimated.

3. Deep Learning Neural Networks

In this study we used three types of DL architectures, and several specifications of autoregressive moving average (ARMA) models used as a benchmark. The selected architectures were:

a) Deep Feed-forward Neural Network (DFNN) – This network is a multilayer perceptron. It is a densely connected network of several layers, where a set of past samples are used as the input vector of the network. This kind of network would be analogous to a moving average (MA) architecture. See section 3.1 for a detailed analysis of DFNNs.

b) Convolutional neural network (CNN) – This type of network is based on a layered structure of convolutions that act as a hierarchy of memories. The first layer performs a linear convolution over the input vector that comprises a window of the input time series, the second layer subsequently performs again a linear convolution over the outputs of the first layer, and so on until the last layer, being the last layer the forecast, denoted as r. The input vector is constructed from the input sequence by rearranging the observation sequence as a matrix, i.e., as a sequence of windowed inputs with overlap. The linear combination is implemented as a convolutional kernel that is applied to the input vector, as a sliding filter. This is further explained in subsection 3.2, which is dedicated to this particular type of DNN.

c) A long short-term memory recursive network (LSTM) – This is a type of RNN architecture based on LSTM cells in which the temporal dynamics of the input are modelled by means of recursive connections. This type of connections are characterised by the fact that the output of a unit, after a delay, can be the input of the same unit, along with the external inputs or the inputs of the lower layers. In particular the memory related to the input time series was structured in such a way that the units of the network observed a set of past samples including a set of past delayed outputs. See section 3.3 for a detailed analysis of LSTMs. For a comprehensive study on RNNs see Hewamalage *et al.* (2021).

These three architectures are particularly suited for time-series forecasting (Bandara *et al.*, 2020; Lara-Benítez *et al.*, 2021). Although the topologies are different and they also differ in the way they deal with the data, they have in common the use of a past window, and an internal representation of the temporal evolution of the time series in order to compute the forecast. For the sake of comparability, we used the same learning algorithm except for the LSTM, which uses a different algorithm due to the special properties of the units.

3.1. Deep Feedforward Neural Network (DFNN)

This is a type of DNN, in which there is a hierarchy of layers of neurons. The input vector connects to the first hidden layer, and the output of each hidden layer connects to subsequent layers, such as shown in Figure 2. In our application, the network consists of a window over the input time series and selects a subset of consecutives samples, that consist of the current observation *x(t)* and N_{LAG} past observations. Note that different windows might overlap.

The input vector is denoted as $x^0 = [x(t), x(t-1), \ldots., x(t - N_{LAG})]$. The vector will be the input to a first hidden layer of size L_1, i.e., the number of units in the first hidden layer. In Figure 2 there is an overview of how the network works.

The DFNN consists of a sequence of layers, with a connectivity that flows from the input at the bottom, to the output. The connections between layers are complete. These connections, which are represented in Figure 2 as a link, consist of a real number that weights the input from the lower layer, and is combined at each unit, represented as a circle, with all the other values of the output of the lower layer by means of a linear combination with the weights of the links.

Each unit performs of a dot product of the of the output vector of the previous layer (the one immediately below) with a set of weights, which are the links to the lower layer of units, and are denoted as *w*. The vector of weights is of the same dimensionality as the input vector. Note that in practice, the model of a neuron has a bias term, which means that the input vector is extended by adding a coordinate that has a constant, normally '1', and the vector *w*, which is variable has a bias term of 0 added at the same coordinate of the extended value of the input vector. Therefore, the bias term is incorporated into the model.

The weights are estimated iteratively by means of gradient search from the training data. That is, given an input vector *x*, the output of a given unit will be a scalar $z = w^T x$. The dot product is processed by a non-linearity which we will denote g(*z*). A summary of typical non-linearities is shown in Table 1.

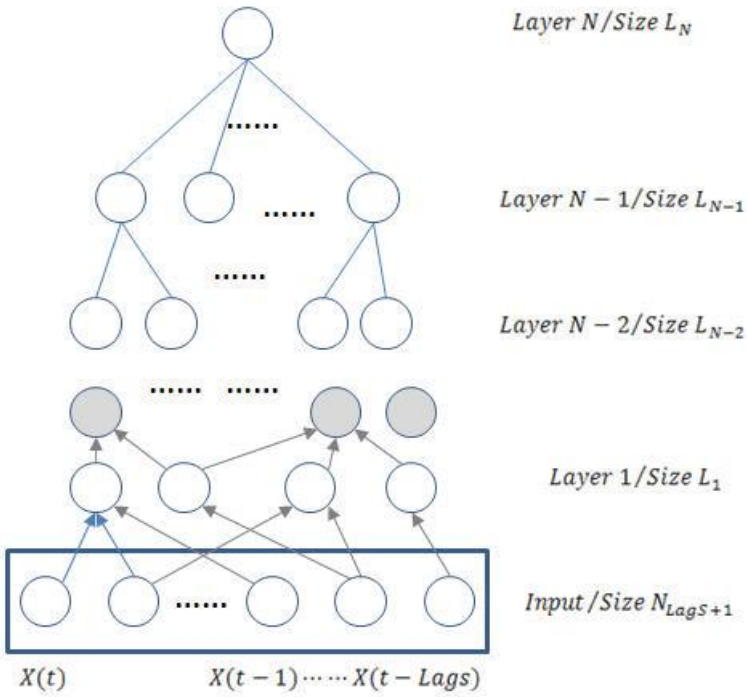

Figure 2. Diagram of a Deep Feedforward Network

Table 1. Summary of typical non-linearities

Non-Linearity	**Expression**
Sigmoid	$g(z) = \frac{1}{1+e^z}$
Tanh	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
Rectified Linear Unit (ReLu)	$g(z) = \begin{cases} z & z > 0 \\ 0 & z \leq 0 \end{cases}$

For notational convenience, we group the process performed in each layer by a matrix product with the input vector to the current layer, which is either the external input or the output of the previous layer. As a result, we will stack the weights $w_{k,i}^T$of the unit "*i*" in layer "*k*" as rows in a matrix, and then we will apply the non-linearity to each element of the vector resulting from the product.

Thus, we define the matrix W^k of weights at layer L_k as,

$$W^k = \begin{bmatrix} w_1^T \\ \vdots \\ w_{L_k}^T \end{bmatrix} \quad (1)$$

Where $W^k x^{k-1}$ is the product of the output of the layer *k-1* with the matrix W^k, which gives a vector of dimension equal to the number of units at layer *k*. In addition, each unit has a bias term, that is a constant term that does not depend on the input. This bias term will be denoted as b^k and is a vector with a dimensionality equal to the number of units at layer *k*.

Therefore, the basic equation computed in layer *k* is the following,

$$\mathrm{x}^{\mathrm{k}} = \mathrm{g}(W^k x^{k-1} + b^k) \tag{2}$$

Where g(.) is a non-linearity. In this research we used the ReLu, since it showed preferable properties regarding the computation of the gradient used to estimate the parameters W^k and b^k. The process for computing the output for a network of *N*+1 layers (including the input layer) consisted in a sequential application of Equation (2) for layer *k*=1 to layer *k*=*N*.

Equations of a DFNN

As an illustration of how the forecast is computed by means of this network, we show the procedure following the diagram shown in Figure 3. The DFNN described below consists of one input, three hidden and one output layer, with a topology defined as [10->100->100->100->1].

The output vector x^1 of the units in the first layer is computed by means of the following equation:

$$x^1 = \mathrm{g}\left(W^1 x^0 + b^1\right) \tag{3}$$

The input vector consists of a sliding window on the input time sequence defined as $x^0 = [x(t), x(t-1), \ldots., x(t - N_{LAG})]$ with $N_{LAG} = 9$. The weight matrix $W^1 \in \mathcal{R}^{10x100}$ connects the input vector x^0 (of dimension $N_{LAG} + 1 = 10$) to the units (of dimension L_1=100) of the first hidden layer. Therefore, $W^1 \in \mathcal{R}^{10x100}$ can be expressed as

$$W^1 = \begin{bmatrix} w_1^T \\ \vdots \\ w_{L_k}^T \end{bmatrix} = \begin{bmatrix} w_{1,1} \ldots w_{1,N_{LAG}} \\ \vdots \\ w_{L_1,1} \ldots w_{L_1,,N_{LAG}} \end{bmatrix} \tag{4}$$

And the offset vector of size equal to the number of units in the first layer $b^1 \in \mathcal{R}^{100}$

$$b^1 = \begin{bmatrix} b_1 \\ \vdots \\ b_{L_k} \end{bmatrix} \quad (5)$$

The output vector x^2 of the units in the second hidden layer are computed in an analogous manner:

$$x^2 = g(W^2 x^1 + b^2) \quad (6)$$

Where $W^2 \in \mathcal{R}^{100x100}$ and $x^2, b^2 \in \mathcal{R}^{100}$. The output of the third layer is computed as $x^3 = g(W^3 x^2 + b^3)$, where $W^3 \in \mathcal{R}^{100x100}$ and $x^3, b^3 \in \mathcal{R}^{100}$. And the final output of the DFNN is computed as $x^4 = g(W^4 x^3 + b^4)$, where $W^4 \in \mathcal{R}^{1x100}$ and $x^4, b^4 \in \mathcal{R}^{100}$.

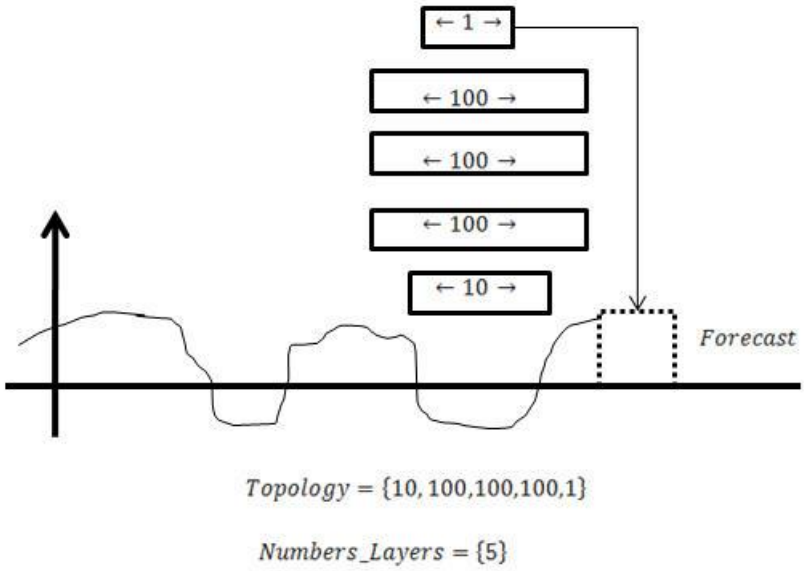

Figure 3. Diagram of a *DFNN* of 5 layers

Note: The input is a sliding window of length 10, and the output is the forecast of the value of the next sample

Estimation of the parameters of a DFNN

The estimation of the parameters is done by means of the Adam (Adaptive Moment Estimation) algorithm, which is a variant of backpropagation. This procedure computes the gradient of the complete set of weights W^k and b^k (for layers k=1 to k=N) with respect to a cost function. For ease of notation, we group all the parameters W^k and b^k into one unique variable which we will denote as **W**, which now incorporates a new column which is b^k. The gradient agitates the parameters so that the value of the cost function is decreased. The selected cost function was the mean square error (MSE):

$$\text{MSE}_{Train}(\mathbf{W}) = \frac{1}{T}\sum_t (x(t+1) - x^{out}(t))^2 \quad (7)$$

This cost function is the MSE between the output of the network $x^{out}(t)$ at time *t*, and the desired value at $t+1$. The mean is computed over an interval of *T* samples. The gradient is computed for all the parameters of the network, and basic updating equation is as follows:

$$\mathbf{W}^{n+1} = \mathbf{W}^n - \mu \nabla_W \text{MSE}_{Train}(\mathbf{W}^n) \quad (8)$$

Where the index '*n*' corresponds to the number of updates, and the estimation of the gradient $\nabla_W \text{MSE}_{Train}(\mathbf{W}^n)$ at each moment is done over a subset of the training database of *T* samples. Given that the training is done on Graphical Processing Units (GPU) with a limited memory, the training database is divided in batches that are feed to the GPU in order to do this sequential update. Note that the estimation of the gradient based on a limited amount of training data can give rise to a noisy estimation.

The Adam algorithm used for training the DFNN shows better convergence properties than the backpropagation algorithm, as it reduces the estimation error of the gradient $\nabla_W \text{E}_{Train}(\mathbf{W}^n)$ by performing an exponential window averaging of the current gradient and the second moment of the gradient. Furthermore, the Adam algorithm is implemented with a forgetting factor that allowed for the shrinking of weights associated with low gradients. The shrinking of weights improves the generalisation performance. A summary of the backpropagation algorithm and its variants, as well as the strategies to make a correct training is presented in Goodfellow *et al.* (2016).

Explored topologies for the DFNN

The design of the experiment is done in such a way that the effects of the architecture on the forecast performance can be assessed. The different structures of the architecture that were explored were:

i. Memory on the time series, which is determined by the number of past samples used for the forecast (*LAG*).
ii. The depth of the network, defined by the number of layers.
iii. The number of units at each layer.

Note that the complexity of the network, i.e., layers and units per layer, determines the flexibility of the topology to approximate a given function. As mentioned in Bengio *et al.* (2013), at the same degree of complexity in terms of number of units in the network, a topology with a greater number of layers has better capabilities in terms of generalisation, understood as the performance in terms of unused data to train the network parameters, The topologies explored in this paper were the had the following combinations:

- Inputs (i.e., N_{LAG}+1): [5, 10]
- Number of layers: [3, 4, 5, 12]
- Units at the hidden layers: [10, 50, 100]

Note that the vector is defined from $x(t)\ to\ x(t - N_{LAG})$. The number of layers, were explored for reasonable sizes, i.e., from 3 to 5, with a value of 12 in order to test if an extremely deep architecture provided better performance. The number of hidden layers, were explored from a size comparable to the size of the input to an extremely wide value of 100. The actual combinations are shown in Table 2.

Table 2. Combinations of DFNN architectures

3 layers	4 layers	5 layers	12 layers
1 input, 1 hidden and 1 output layers	1 input, 2 hidden and 1 output layers	1 input, 3 hidden and 1 output layer	1 input, 10 hidden and 1 output layer
[5->10->1]	[5->10->10->1]	[5->10->10->10->1]	[5->100->100->100->100->100->100->100->100->100->1]
[5->50->1]	[5->50->50->1]	[5->50->50->50->1]	
[10->10->1]	[10->10->10->1]	[10->10->10->10->1]	
[10->50->1]	[10->50->50->1]	[10->50->50->50->1]	[10->100->100->100->100->100->100->100->100->100->1]
[5->100->1]	[5->100->100->1]	[5->100->100->100->1]	
[10->100->1]	[10->100->100->1]	[10->100->100->100->1]	

3.2. Convolutional Neural Network (CNN)

CNNs can be regarded as a particular type of DFNN, where the estimation of the input vector to each of the layers is processed by means of a convolution operation, i.e. the input vector or output of the previous layers is sequentially processed by a convolution which creates the new output. This is a type of network that was introduced in the 90s for image recognition (Goodfellow *et al.* 2016), and can be employed with time series by using a 1D convolutional operator instead of the usual 2D convolution used in image recognition.

This approach has several advantages. First, instead of a complete matrix of dimensions $L_k \times L_{k-1}$, as it was the case with DFNNs, we assign to each unit a vector of smaller length, therefore the size of the parameters is $L_k \times L_{w,k}$, which is typically a smaller value than the completely connected matrix. This effect is illustrated in the diagram of Figure 5. In addition, the estimation of the parameters of the convolution to takes into account the sequential structure of the input and therefore captures time properties of the time series.

Formal specification of the CNN

Next, we explain how we specify a CNN in five consecutive steps. Given a time series $X = [x(0), x(1), \ldots., x(T)]$,

a- First, we rearrange the elements to form an extended $X^{Extended}$ matrix of the following form:

$$X^{Extended} = \begin{bmatrix} \mathrm{x}(0) & \mathrm{x}(1) & \cdots & \mathrm{x}(\mathrm{T}-\mathrm{N}_{\mathrm{LAG}}-1) & \mathrm{x}(\mathrm{T}-\mathrm{N}_{\mathrm{LAG}}) \\ \mathrm{x}(1) & \mathrm{x}(2) & & \mathrm{x}(\mathrm{T}-\mathrm{N}_{\mathrm{LAG}}-2) & \mathrm{x}(\mathrm{T}-\mathrm{N}_{\mathrm{LAG}}-1) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathrm{x}(\mathrm{N}_{\mathrm{LAG}}) & \mathrm{x}(\mathrm{N}_{\mathrm{LAG}}-1) & & \mathrm{x}(\mathrm{T}-1) & \mathrm{x}(\mathrm{T}) \end{bmatrix} \quad (9)$$

This process is illustrated in Figure 4.

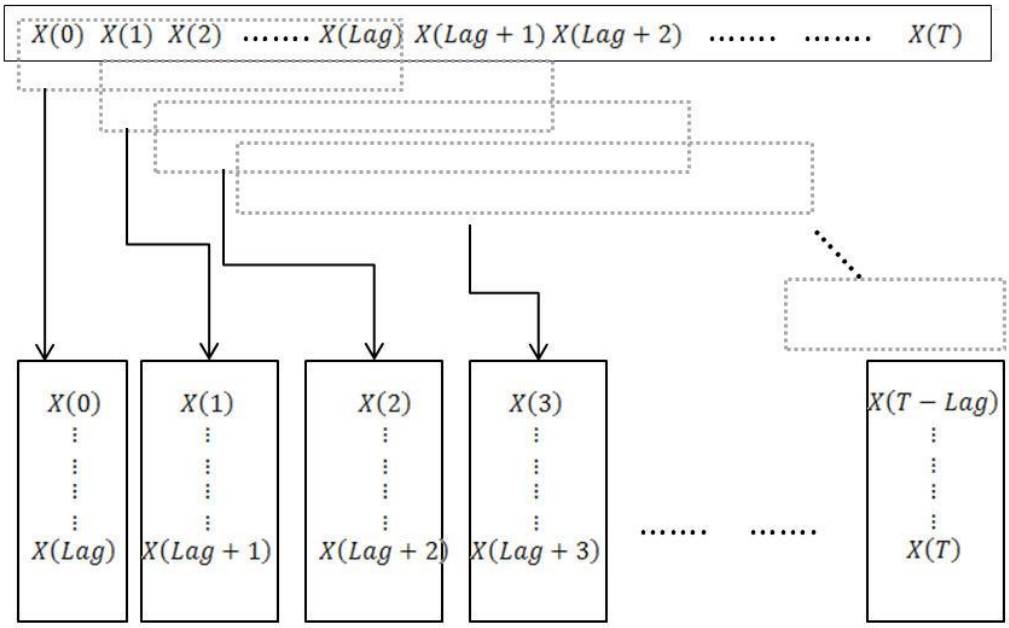

Figure 4. First Step. Rearrange the input time series into a time series of vectors of length N_{LAG}+1

b- Next, we group the elements of $X^{Extended}$ into a sequence of input matrices as illustrated in Figure 5 where the length of the input to the CNN is defined as L_CNN. Thus, we create a matrix of dimensions $N_{LAG} \times L\_CNN$, defined for the time 'n' as follows,

$$X_n^{input} = \begin{bmatrix} \mathrm{x(n)} & \mathrm{x(n+1)} & \cdots & \mathrm{x(n+}L\_CNN-1\mathrm{)} & \mathrm{x(n+}L\_CNN\mathrm{)} \\ \mathrm{x(n+1)} & \mathrm{x(n+2)} & & \mathrm{x(n+}L\_CNN-2\mathrm{)} & \mathrm{x(n+}L\_CNN-1\mathrm{)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathrm{x(n+N_{LAG})} & \mathrm{x(n+N_{LAG}-1)} & & \mathrm{x(n+N_{LAG}+}L\_CNN-1\mathrm{)} & \mathrm{x(n+N_{LAG}+}L\_CNN\mathrm{)} \end{bmatrix} \quad (10)$$

c- Third, we create a sequence of input matrices $X_1^{input} \ldots X_T^{input}$. Each matrix is part of the CNN input.

d- Next, the network scans each of the X_n^{input} matrices using as input in each timestamp a row of length N_{LAG} as shown in Figure 6. Note that the equation that gives the output of the m-th unit in the first layer in this case is of the form

$$x^1(m) = w_m^1 * X_n^{input} = \sum_{n=0}^{L_{CNN}} w(n)_m^k \, x^{input}(m-n) \quad (11)$$

Where x^{input} is the vectorisation of the segment of the matrix X_n^{input} under the convolution operation.

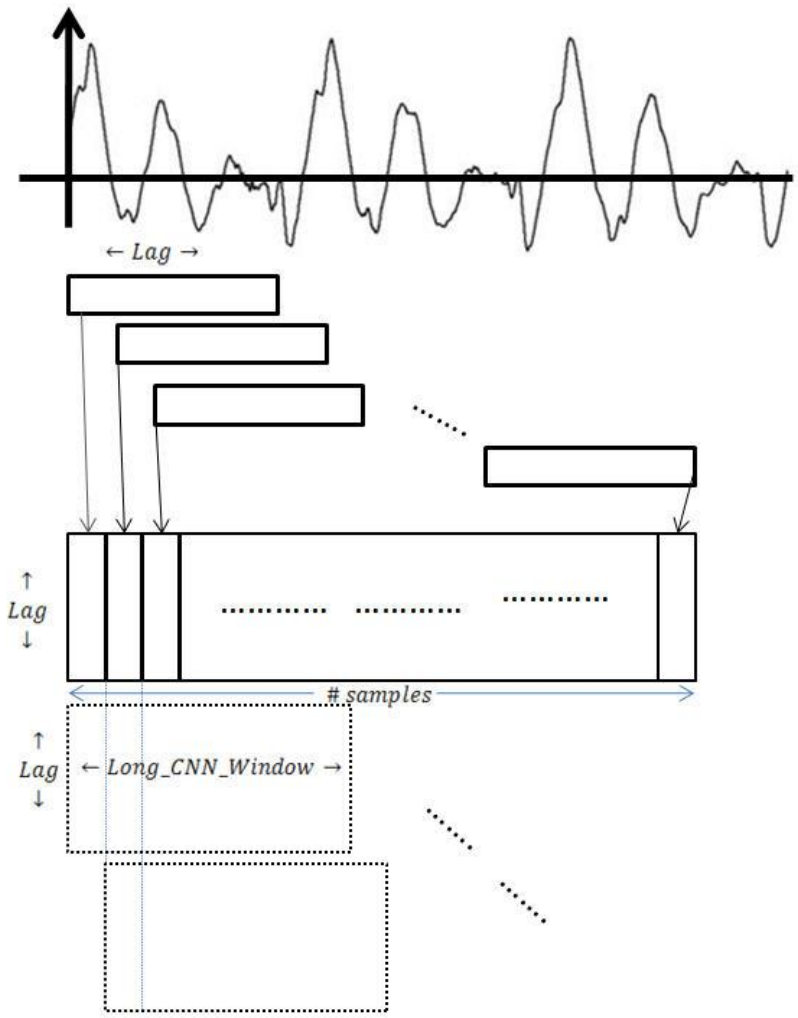

Figure 5. Segmentation of the time sequence of vectors of length Lag+1, into arrays of dimension $\boldsymbol{N_{LAG}}$+1 times L_CNN

Note: This is done by a stacking the values of a sliding window into an array and afterwards selecting subsets of the array of a shape Lag times the longitude of the CNN window

e- Finally, since the connectivity and the outputs of the network is dynamical, in the sense that the input is not totally connected to the first layer units, the input is sequentially scanned in order to create a sequential representation of the input at the hidden layer. This is illustrated in Figure 6, where a sliding array of size $L_k * N_{LAG}$ moves from left to right with increments of one column.

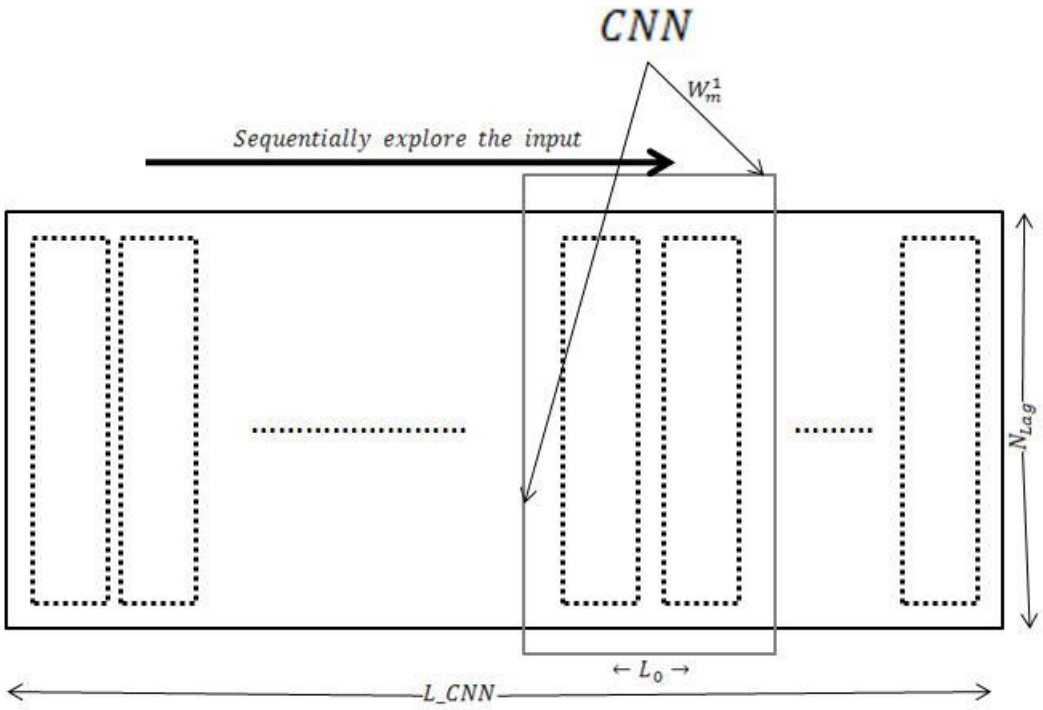

Figure 6. Illustration of the convolution operation, the weights of a given unit at the first hidden layer, the array is explored sequentially by a subarray of dimensions L_0 times N_{LAG}. The convolution at the external sides of the subarray is computed by zero padding

The idea behind CNNs is to substitute a matrix operation $W^{k}x^{k-1}$such as shown in Equation (2), by a convolution that computes each of the elements of the output vector x^k sequentially in time. Thus, the output vector at layer *k*, is computed as follows:

The m-th element of the vector x^k is computed as a convolution with elements of the *k*-1 layer. That is the output of unit 'm' is computed from the linear combination of the L_{k-1} outputs of the previous layer, combined linearly with the vector of weights w_m^k of length $L_{w,k}$ of the unit *m* at the layer *k*. Note that the length of the vector w_m^k is typically lower than the number of units at the lower layer, i.e. $L_{w,k} \leq L_{k-1}$.

The vector w_m^k represents the convolution kernel. The convolution process will be denoted by the '*' symbol as $x^k\,(m) = w_m^k * x^{k-1}$ and is computed as follows:

$$x^k(m) = w_m^k * x^{k-1} = \sum_{n=0}^{L_{k-1}} w(n)_m^k\, x^{k-1}(m-n) \tag{12}$$

Finally, the vector at layer *k* is created by the concatenation of the L_k convolutions at layer k, therefore:

$$x^k = [\,x^k\,(0), \ldots, x^k\,(L_k\,)] \tag{13}$$

The process described above is summarised in the diagram of Figure 7.

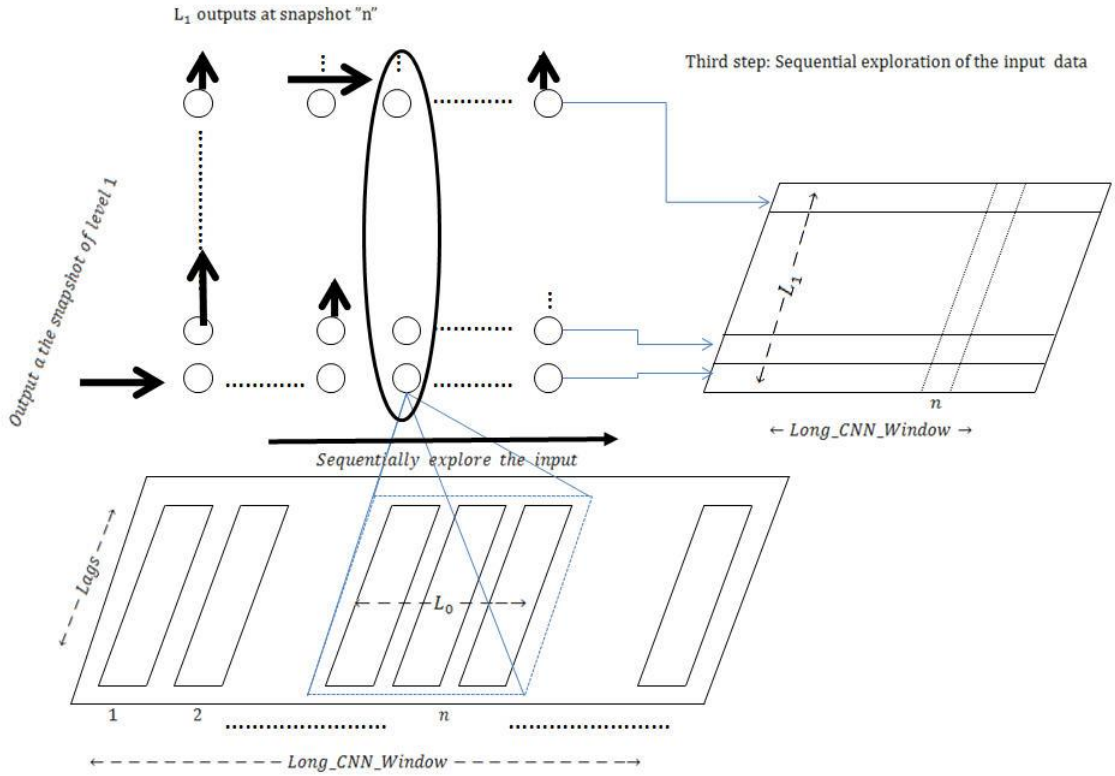

Figure 7. Third Diagram. The output of the CNN

Other comments related to CNNs

CNNs make a trade-off between the complexity of a matrix product per vector for a distribution over time of a vector product per vector in a sliding window. This dynamic aspect of the vector slider product allows the network to detect time patterns. These time patterns are also represented in a hierarchical way.

An aspect related to the convolution operator is the treatment of the extreme elements of the input vector to each layer and the step (stride) in the convolution. In this article we took a stride of one for all the layers and zero padded the extreme elements of the input vector to each layer.

Estimation of the parameters

The estimation of parameters is done by gradient search as explained before for DFNNs (see Section 3.1). That is the change in the weights is proportional to the derivative of MSE $_{Train}(\mathbf{W})$ with respect to each of the weights of the network, along with a smoothing term. Since the task at hand consists of time series prediction, it becomes a problem of function approximation. Hence, the cost function consisted of the quadratic approximation error in the training data base. That is,

$$\text{MSE}_{Train}(\mathbf{W}) = \frac{1}{T}\sum_t (x(t+1) - x^{out}(t))^2 \quad (14)$$

The net used for non-linearities the ReLu, and the stopping criterion relied also on the change on the performance on the validation database, and the regularization terms were applied in a similar way to that of the DFNN.

Topologies for the CNNs

The design of the experiment is done so that the effects of the architecture on the forecast performance can be assessed. The dimensions of the architecture that we explored were:

a- Memory on the time series, that is defined by the lag value N_{LAG}+1, i.e., the number of past samples used for the forecast, including the current one.
b- Window length L_CNN used to create the matrix $X^{Extended}$
c- Value of L_k used for performing the convolution.
d- The depth of the network, defined by the number of layers.
e- The number of units at each layer.

The topologies explored in this paper were the had the following combinations:

- Inputs (i.e., N_{LAG}+1): [10]
- Window (i.e., L_CNN): [10,15,20],
- Values of L_{k-1} :[5,10]
- Number of layers: [3,4]
- Units at the hidden layers: [100,200]

The coding of the architectures is "$N_{LAG}+1$, *L_CNN* input Layer, [Units per layers], L_k". Therefore, the code "9, 10, [100,100], 5" refers to N_{LAG}=9 and *L_CNN*=10, two hidden layers of 100 units each, and the values L_k for all the layers is 5. The combinations used in the paper are listed in Table 3.

Table 3. Topologies for the CNNs

3-layer architectures		4-layer architectures	
10->100->1		10->100->100->1	
9, 10, [100], 5	9, 10, [100], 10	9, 10, [100,100], 5	9, 10, [100,100], 10
9, 15, [100], 5	9, 15, [100], 10	9, 15, [100,100], 5	9, 15, [100,100], 10
9, 20, [100], 5	9, 20, [100], 10	9, 20, [100,100], 5	9, 20, [100,100], 10
10->200->1			
9, 10, [100], 5	9, 10, [100], 10		
9, 15, [100], 5	9, 15, [100], 10		
9, 20, [100], 5	9, 20, [100], 10		

3.3. Deep Recurrent Neural Network (DRNN) – Long short-term memory (LSTM)

RNNs are a family of layered networks that are composed of units that have feedback connections. As a result, the output of the unit is fed back with a delay and concatenated either with the external input or with the output of the lower layer units (see left part of Figure 8).

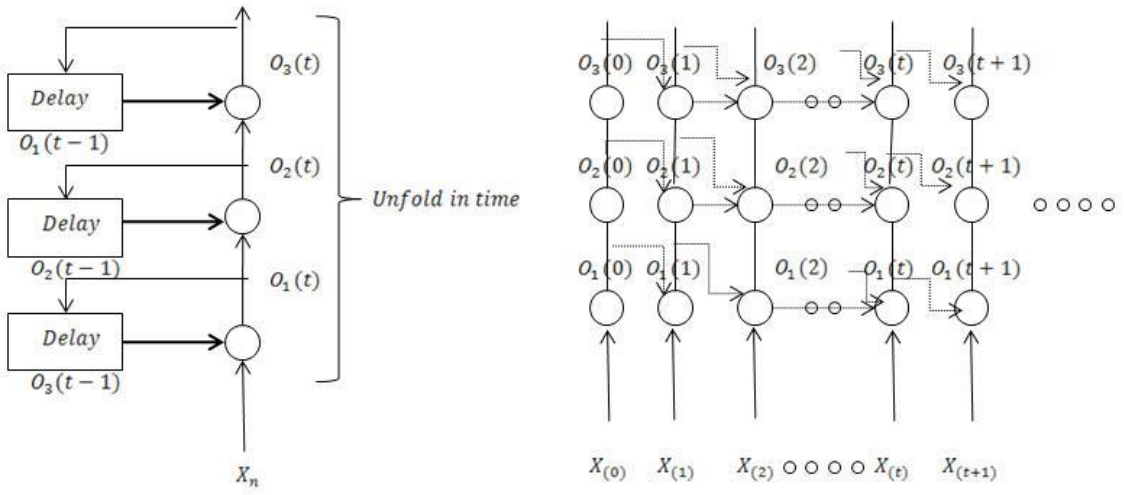

Figure 8. Left, network of one input and three layers, in where we present the form of the feedback. Right, the same network temporarily unfolded. i.e., the states at each time stamp are depicted as columns

The possible set of feedback connections include not only feedback in the same unit but also from units at higher layers to units at lower layers. This type of architecture captures the dynamical aspects of the input time series, since the feedback of each unit keeps a track of the past features of the time series and is used for computing the future values depending on a window over the past samples. These cycles allow the network to have memory of the past.

We decided to restrict ourselves to the structure summarised in Figure 8, and discarded the option of allowing feedback connections from one unit to any of the lower ones. This decision is justified for two reasons, the first being that the possible combinations of feedback in one unit with units in the lower layers results in an excessive number of configurations that makes comparisons between architectures unfeasible, computationally wise. Secondly, since the architecture we propose has been successfully used in language translation systems, in which the temporal sequence consists of sentences formed by words, and in which there is temporal dependence of the current observation with observations from the past (Sutskever *et al.*, 2014).

This type of application shares with that of prediction the fact that the value to be estimated depends conditionally on values from the past. This dependence with the past in is not fixed in any case, since the sample from the past with the greatest influence on the current sample is not at a fixed distance.

We did not introduce an attention mechanism (Bahdanau *et al.*, 2014), because the size of the temporal windows employed in this paper does not allow for a correct implementation of this mechanism. Attention mechanisms are useful when the time dependency between the output and a given term in the past is unknown and can have a wide span of temporal variation.

Finally, as explained in Siegelmann and Sontag (1991), the structure we use is equivalent to a universal Turing machine and therefore has the capacity of a universal function approximator. The best way to understand the way RNNs process information is by taking the point of view of the unfolding in time of a dynamical system. By unfolding in time the internal states of the network, we obtain a repetition of the network at each moment in the temporal evolution of the series. In Figure 8, we show the unfolding of the network. For computational purposes, the network is replicated as many times as we have samples in the input series, and then operations are applied. As for the gradient calculation, we apply what is known as backpropagation through time. This process of network time unfolding justifies the database partition shown in Figures 9a and 9b.

Dynamics of RNNs in terms of units

Next, we explain how to generate the network input vector X_n^{input}. We start by describing the structure of the network and its temporal dynamics in terms of input/output of units, and we will leave for later the internal details of each unit. Since the dynamics of the recursive part may give rise to instabilities and problems related to the exponential growth or vanishing of the gradients, the units in RNNs are more complex than those in DFNNs or CNNs (see Equation 2). The type of units used in this article are the LSTM (Hochreiter & Schmidhuber, 1997), which are described below.

The use of a special type of unit is justified in order to prevent problems that were found in topologies that had recursive connections but conventional units. The use of these units gave rise to instabilities, vanishing gradients and exploding gradient.

To prevent these phenomena, LSTM were proposed as gated networks in which the inner connections prevent the gradient from vanishing or exploding, and also allowed information to be accumulated. The flow of information will be described first at the unit level and then at the structure level. Note that the fact that there is a feedback term in each unit provides a memory mechanism that allows dealing with the dynamic aspects of the time series.

Temporal structure of the outputs of a single unit

Each unit is a function 'h' that computes the output from past outputs and the current input. This function 'h' will correspond to the LSTM described in more detail later in this section. The output $o(n)$ of a given unit in the network depends on a function $h(.;W)$ through the values of the current input $x(n)$, the delayed output $o(n-1)$ and with the weights W. Thus, the form of the function can be expressed as:

$$o(n) = h(x(n), o(n-1); W) \tag{15}$$

which in turn can be unfolded according to the output of the preceding moment as,

$$o(n) = h(x(n), h(x(n-1), o(n-2); W); W) \tag{16}$$

This feedback can be unfolded using the past values of the input time series until the initial value *x(0)*. Thus, the feedback in turn can be extended to the initial sample:

$$o(n) = h(x(n), h(\dots\dots\dots h(x(0), o(-1); W); W); W) \quad (17)$$

where the initial output $o(-1)$ is set to zero.

Therefore, the structure makes the current value depend on the whole previous history of observations $[x(n), x(n-1), \dots., x(0)]$. Note that in the case of units in layers above the first, the input *x(n)* can be substituted by the outputs of the lower layer.

Temporal structure of the outputs in a several layer network

a- **Case of one unit per layer**. In Figure 9a we show a network of one input and three layers, in where each unit is represented with a one-unit delay feedback. While in Figure 9b we show how the network unfolds in time. That is, each column corresponds to a snapshot at a given moment, which depends on the outputs of the previous snapshot. Note that the time unfolding is a display of the dynamics of the network, and the network continues to be the layered layout shown in Figure 9a.

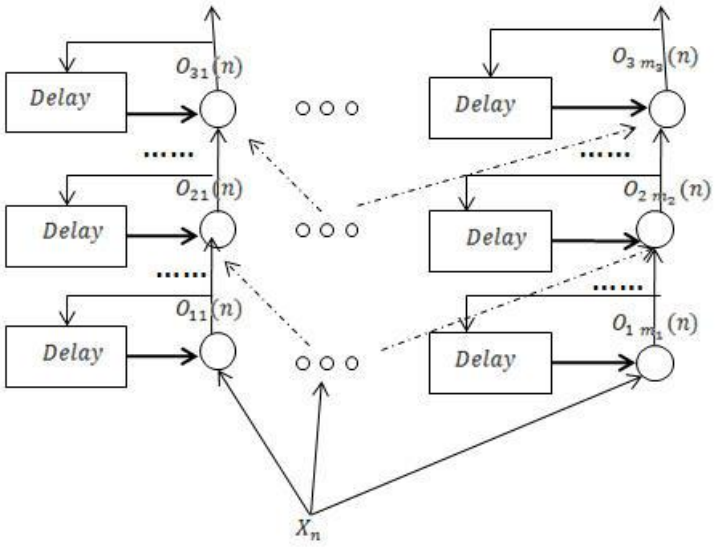

Figure 9a. Network of one input and three layers, and several units at each.

b- **Case of m_k units per layer.** RNNs can have more than one unit at each layer, and a similar diagram is shown in Figure 9b, where we show the diagram of a three-layer network with more than one unit per layer, and at the right the time unfold of the network.

RNNs are based on a recursive diagram such as shown in Figures 8 and 9, where each unit has an input either from the original time series or from a lower layer and feedback of its own output after a delay of one sample. As it can be seen in Figure 9a, each unit has two inputs, the current observation vector at time *n*. For a given output of the unit *j* of layer *k*, $o_{k,j}(n)$, the inputs of the unit are:

- In the case of the first layer: $X(n) = [x(n), x(n-1), \ldots., x(n - N_{LAG-1})]$
- In the case of the hidden layer *k*: $[o_{k-1,1}(\mathrm{n}), o_{k-1,2}(\mathrm{n}), \ldots., o_{k-1,m_{k-1}}(\mathrm{n})]$, where m_{k-1} denotes the number of units at the layer *k*-1, i.e., at the previous layer.
- In any of the above cases, the output value delayed one unit, that is $o_{k,j}(\mathrm{n}-1)$, is appended to the input vector of the unit.

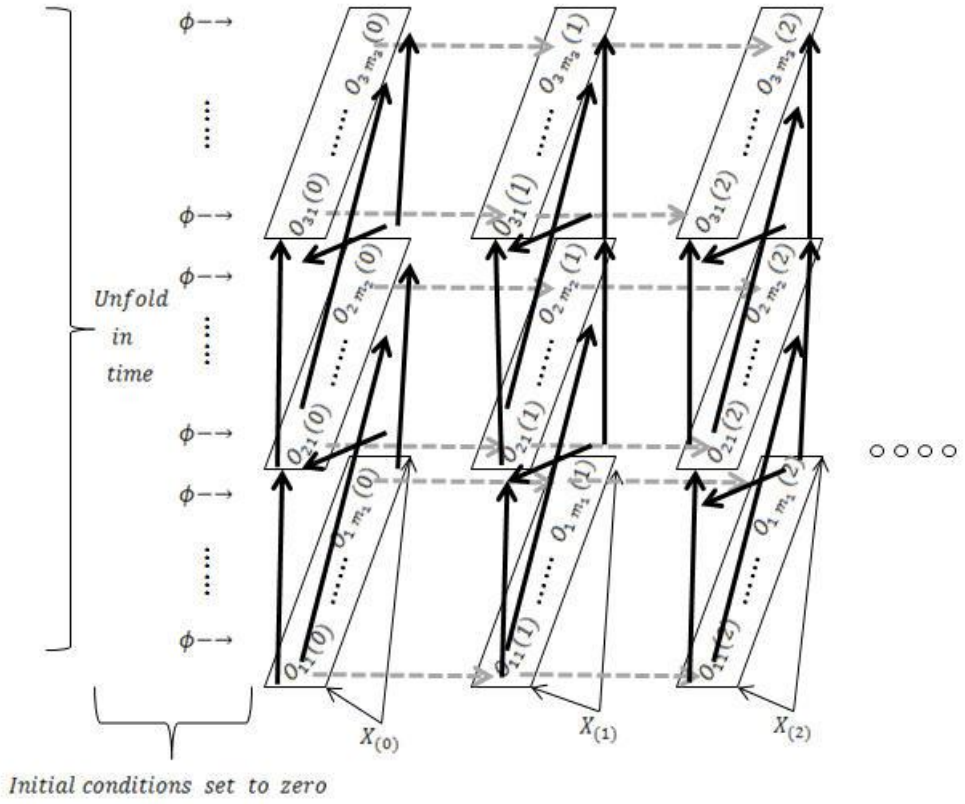

Figure 9b. Output of a neuron in the first layer. Time series of the output for one neuron time series unfolded in time

Procedure for calculating the output of the first layer

The original time sequence is organised as described in Figure 4, generating a sequence of vectors of dimension $N_{LAG} + 1$ by means of the following steps:

a- First, rearranging the elements to form an extended $X^{Extended}$ matrix of the form:

$$X^{Extended} = \begin{bmatrix} \mathrm{x(0)} & \mathrm{x(1)} & \cdots & \mathrm{x(T-N_{LAG}-1)} & \mathrm{x(T-N_{LAG})} \\ \mathrm{x(1)} & \mathrm{x(2)} & & \mathrm{x(T-N_{LAG}-2)} & \mathrm{x(T-N_{LAG}-1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathrm{x(N_{LAG})} & \mathrm{x(N_{LAG}-1)} & & \mathrm{x(T-1)} & \mathrm{x(T)} \end{bmatrix} \quad (18)$$

b- As shown in Figure 5, the array $X^{Extended}$ is subdivided into a time sequence of matrices X_n^{input} of dimension $\mathrm{N_{LAG}} \times L\_LSTM$ indexed by the time stamp *n*. The variable L_LSTM is the size of the input window that is used for computing the output at the current moment. Thus, the neuron sequentially scans the input vector sequence of the form:

$$X_n^{input} = \begin{bmatrix} \mathrm{x(n)} & \mathrm{x(n+1)} & \cdots & \mathrm{x(n+L\_LSTM-1)} & \mathrm{x(n+L\_LSTM)} \\ \mathrm{x(n+1)} & \mathrm{x(n+2)} & & \mathrm{x(n+L\_LSTM-2)} & \mathrm{x(n+L\_LSTM-1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathrm{x(n+N_{LAG})} & \mathrm{x(n+N_{LAG}-1)} & & \mathrm{x(n+N_{LAG}+L\_LSTM-1)} & \mathrm{x(n+N_{LAG}+L\_LSTM)} \end{bmatrix} \quad (19)$$

c- Next, as shown in Figure 10, the input vector X_n^{input} is explored sequentially by each neuron, and generates for instance in the case of the first neuron a sequence $[o_{1,1}(\mathrm{n}), o_{1,1}(\mathrm{n}+1), \ldots., o_{1,1}(\mathrm{n}+L\_LSTM)]$. The combination of the output of all the units m_1 of the first layer gives rise to a matrix O_1 that consists of the concatenation of the outputs of each unit. This matrix is of size $(\mathrm{L}_{LSTM}+1) \times m_1$ and is of the form:

$$O_1 = \begin{bmatrix} o_{1,1}(\mathrm{n}) & o_{1,1}(\mathrm{n}+1) & \ldots & o_{1,1}(\mathrm{n}+L\_LSTM-1) & o_{1,1}(\mathrm{n}+L\_LSTM) \\ o_{1,2}(\mathrm{n}) & o_{1,2}(\mathrm{n}+1) & & o_{1,2}(\mathrm{n}+\mathrm{L}\_LSTM-1) & o_{1,2}(\mathrm{n}+L\_LSTM) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ o_{1,m_1}(\mathrm{n}) & o_{1,m_1}(\mathrm{n}+1) & & o_{1,m_1}(\mathrm{n}+L\_LSTM-1) & o_{1,m_1}(\mathrm{n}+L\_LSTM) \end{bmatrix} \quad (20)$$

Procedure for calculating the output of the upper layers

The procedure follows the diagram illustrated in Figure 10, in which in layer *k* there is an input array O_k of size $(\mathrm{L}_{LSTM}+1) \times m_k$, where m_k is the number of units at layer *k*.

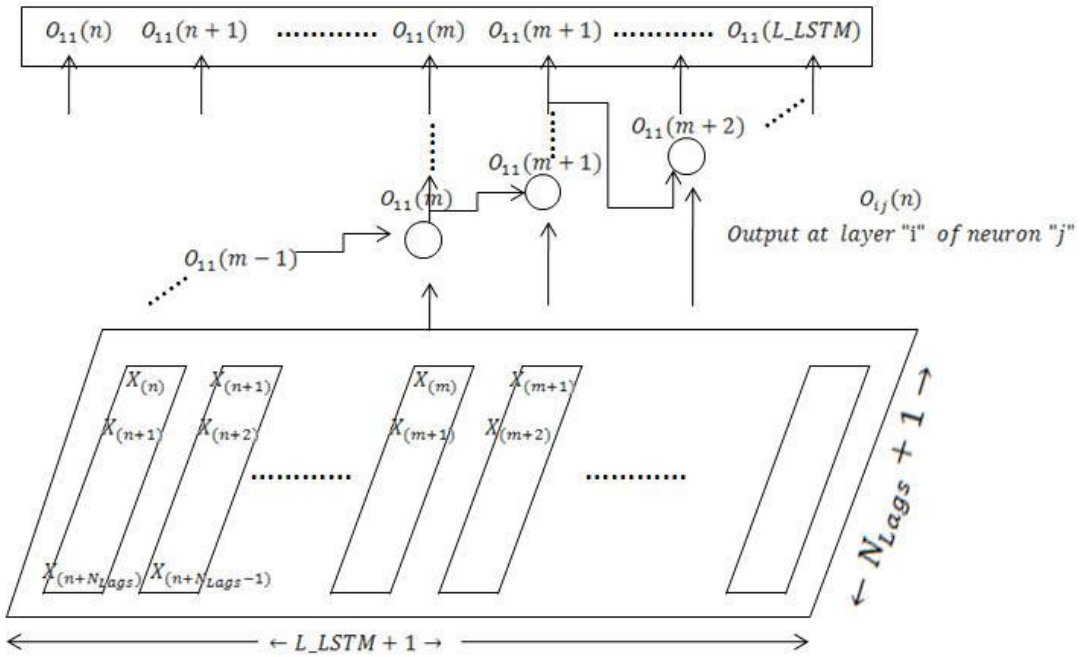

Figure 10. Internal structure of a LSTM unit. Four neurons compute an internal state s_t and the output of the unit o_t , which are feed back with a delay of one unit of time

Estimation of the parameters via back propagation through time

Parameter estimation is performed by backpropagation, which is an algorithm that computes the gradient for a single input value. This method can be extended for the case of feed-forward networks, where unfolding is done over time. This unfolding consists of creating as many replicas of the network as values of the input sequence. In the case of sequences of large length or indefinite duration, it is customary to create a maximum horizon, i.e., a maximum number of replicas. Once the replicated version of the network has been constructed, connections between networks associated with consecutive points in time are made at feedback points. Therefore, the inputs of each network will consist of the external input and the output of the network associated with the previous interval. Once the structure has been completed, the entire structure is trained simultaneously, so that the structure associated with the unfolding in time will have the entire time sequence as input, together with the outputs associated with each network of the replica. In this sense, the backpropagation algorithm is applied globally to determine the gradient associated with the set of weights.

Summary of the LSTM units

We selected the LSTM as unit for the recurrent term of the network. An alternative type of unit is the GRU, which has a similar structure. Empirically, it has been found that both types of recurrent units yield similar performance in applications with complex temporal structure such as translation. These properties are mentioned in Jozefowicz *et al.* (2015), where a comprehensive set of experiments were done.

The LSTM units are designed in order to avoid problems associated with RNNs which are due to the fact that the gradient either vanishes or explodes during the training phase, giving rise to nets that suddenly suffer what was known as 'catastrophically forgetting'. The idea of these units is to allow paths for passing the gradient through time, and also allowing for the preservation of the past events that might be useful for the computation of future outputs. The interesting feature of this kind of units is that the weights that control these two mechanisms are learned from examples. A diagram of this kind of units is depicted in a diagram shown in Figure 11, where we show the external form, and in Figure 12, where we show the internal components of the unit.

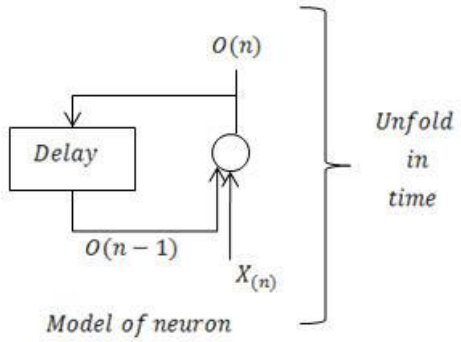

Figure 11. Internal connections of the four units that make up the LSTM.
Note: Note that the vectors x_t and o_{t-1} are concatenated and copied as input of the four internal neurons of the LSTM unit. Also, note that the weights and non-linearities associated with each of the neurons are detailed in the text.

In order to be able to model the dynamics of a recurrent unit and be able to compute the gradient of the backpropagation algorithm through time, the LSTM unit has an inner structure that is made up of four individual neurons and an internal state, as shown in Figure 11. These four internal neurons have the structure of a conventional artificial neuron:

$$output\_unit = Non\_linear\_function(W\ unitinput\_unit + bias) \quad (21)$$

Three of these units, i.e., the forgetting, the gate and the state unit, control de internal state of the LSTM unit. The output of the unit, as shown below, is itself controlled by the internal state. Note that all weights of the three units are computed from examples by gradient search, through the back propagation through time.

The diagram in Figure 12, shows the connection of the inner units of the LSTM. Note that the vectors x_t and o_{t-1} are concatenated and copied as input of the four internal neurons of the LSTM unit. Also note that there are 'product' and 'addition' modules, which operate at a vector level.

The forgetting unit at time 't', denoted as f_t , multiplies the value of the previous state s_{t-1} and therefore functions as a regularization mechanism, that modulates the influence of the previous state. As the non-linear function associated with this unit is a sigmoid, the output of this unit will be in the range [0,1].

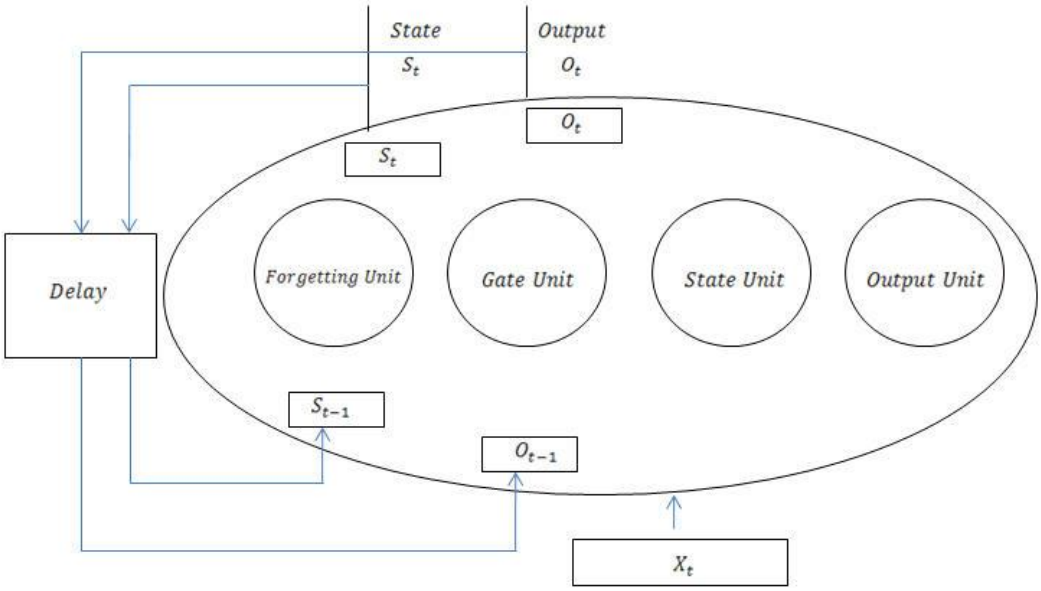

Figure 12. Internal structure of a LSTM unit. Four neurons compute an internal state s_t and the output of the unit o_t , which are fed back with a delay of one unit of time.

Note in Figure 13 that this unit is complementary of the gate unit that controls the influence of the estimation of the new state. The output of the forgetting unit at time *t*, is denoted as f_t and is computed by means of a neuron with a sigmoidal non-linearity, $\sigma(z)=1/(1+e^z)$ and the weights are W^f. The final equation for computing the forgetting term can formalized as:

$$f_t = \sigma\left(W^f\left[x_t \quad o_{t-1}\right] + b^f\right) \tag{21}$$

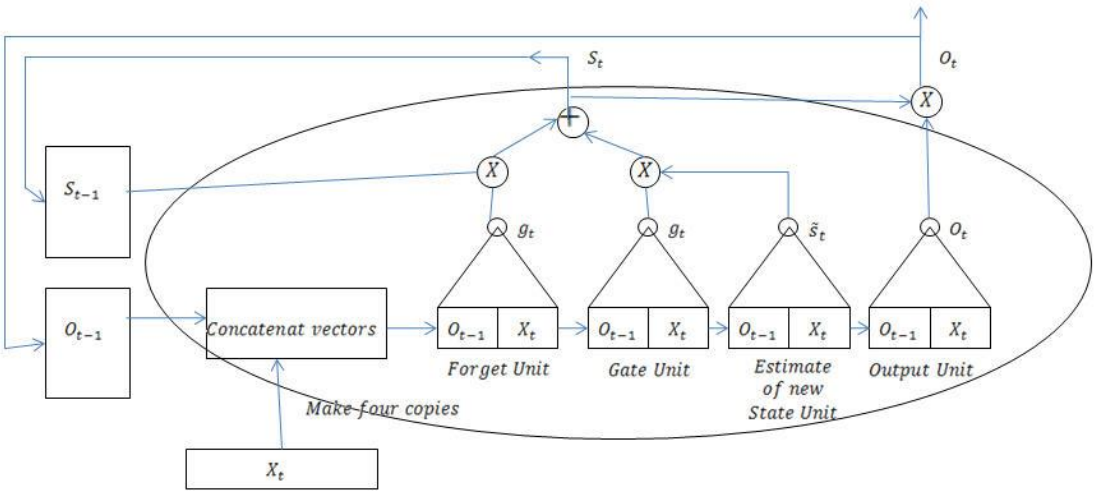

Figure 13. Internal connections of the four units that make up the LSTM.
Note: Note that the vectors x_t and o_{t-1} are concatenated and copied as input of the four internal neurons of the LSTM unit. Note that the weights and non-linearities associated with each of the neurons are detailed in the text.

The gate unit, denoted as g_t , controls the flow of the estimation of the new internal state of the LSTM unit. This estimation of the new internal state is denoted as $\tilde{s}_t$. Both are computed with a neuron with a non-linear function, in the case of the gate unit is a sigmoid with a range in [0,1], which regulates the extent to which the value of the estimate of the new internal state is retained:

$$g_t = \sigma\left(W^g\left[x_t \quad o_{t-1}\right] + b^g\right) \tag{23}$$

On the other hand, the estimate of the new internal state uses a hyperbolical tangent (tanh) function define as $\tau(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, and a set of weights W^s and bias b^s:

$$\tilde{s}_t = \tau\left(W^s\left[x_t \quad o_{t-1}\right] + b^s\right) \tag{24}$$

The tanh function gives an output in the range [-1,1], which afterwards is used to control the level and sign of the estimated output o_t .

The link between the gate output and the estimate of the new internal state output is multiplicative, as shown in Figure 13. The value of the new internal state is computed as follows:

$$s_t = s_{t-1} * f_t + g_t * \tilde{s}_t \tag{25}$$

This equation allows for controlling the output by means of two estimated values, the first is a local in time forgetting weight f_t , which determines up to which point the

past state of the unit has to be preserved, and the other is the gate weight g_t which controls if the estimated new state is allowed to be preserved. The parameters of the three terms are trained from examples by means of a gradient search algorithm, therefore the estimation is done in such a way that the resulting new state is useful for the computation of the output at the current output.

The output of the unit is computed by a set of independent weights and a sigmoid non-linearity as,

$$\tilde{o}_t = \sigma\left(W^O\left[x_t \quad o_{t-1}\right] + b^O\right) \tag{26}$$

The key point in the working of the neuron that allows controlling the flow of information in time when the computation is unfold, is the fact that the output is multiplied by the internal state, multiplied by a tanh non-linearity, which in turn allows scaling and a possible change of sign. Therefore, the final output is

$$o_t = \tilde{o}_t * \tau(s_t) \tag{27}$$

Note that the feedback term related to the state of the unit s_t allows conveying to the future information that may be relevant to determine a value several time units later, and the decision of whether this information is used is determined by the two terms f_t and g_t .

Explored topologies for the LSTM network

The dimensions of the architecture that we explored were:

i. Memory on the time series, that is defined by the Lag value N_{LAG}+1, i.e., the number of past samples used for the forecast, including the current one.
ii. Window length L_LSTM used to create the matrix $X^{Extended}$
iii. The number of units at each layer.

The structure of the networks are of three layers, with only one hidden layer. The combination of parameters explored were the following:

- N_{LAG}+1= [10, 15, 20]
- L_LSTM= [5, 10]
- Units in the hidden layer = [100, 200]

4. Design of the experiment

In this study, we use the daily US/UK foreign exchange rate provided by the Federal Reserve Bank of St. Louis (https://fred.stlouisfed.org/series/DEXUSUK). Data is not seasonally adjusted. The motivation for focusing in this particular variable is twofold. On the one hand, the increase in uncertainty and the consequent volatility in investment markets associated with the days leading up to Brexit, made it an ideal setting for the analysis. On the other hand, since DL models require long time series the daily price of the dollar in relation to the pound is one of the longest time series available. In Figure 14 we graph the evolution of the exchange rate from January 4, 1971 to Januray 31, 2020, the day before Brexit.

Figure 14. Daily US/UK foreign exchange rate

To evaluate the performance of the three types of DL architectures against a benchmark, we have chosen several specifications of autoregressive moving average (ARMA) models that mirrored the orders of the input (MA) term and the order of the recursive term (AR) used in the other DL architectures. See Figure 15.

The criterion for deciding the orders of the ARMA model was to have a benchmark with similar memory structure of the DL architectures. The similarity is in the sense of size of input values, i.e., MA component and size of the feedback, and AR component. The combinations that were explored were M = [1, 6, 11] and K = [0, 5, 10, 15].

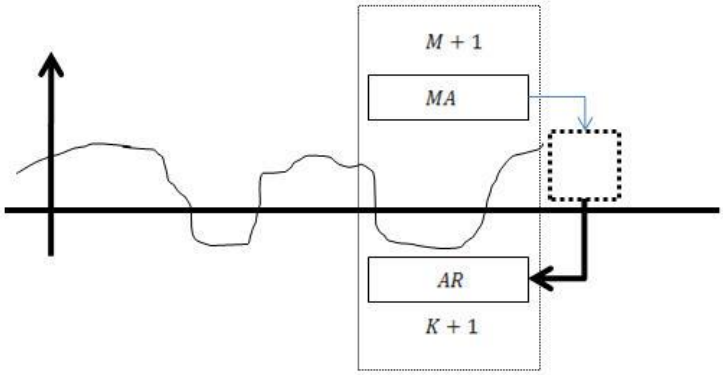

Figure 15. Diagram showing the terms in the ARMA structure

Since we are interested in the performance of DL techniques, we have designed an out-of-sample forecasting experiment, using 35 working days after Brexit, until March 20, 2020. Given that the computational effort of training the networks is high, we opted for a sequential partition of the dataset with a validation sub-sample with a fixed size and an increasing segment used for training.

The design of the partition of the dataset was done in such a way that the final experiment was as close as possible to a recursive out-of-sample forecasting exercise. With that aim we applied a four-fold partition of the data (see Figure 15). Each partition, had a fixed size validation part, and an increasing segment for training. As explained before, the justification for this approximation is related to the computational requirements to train these types of networks.

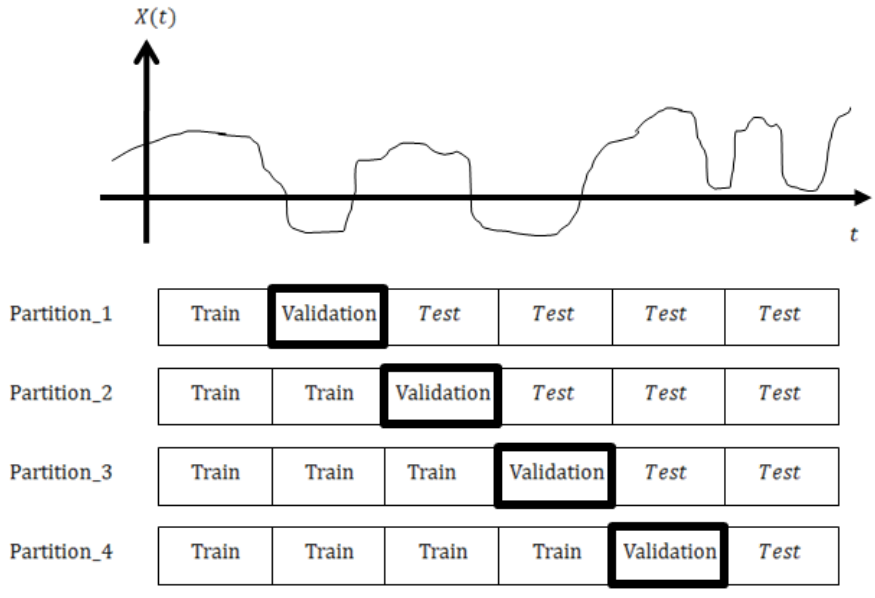

Figure 15. Partition of the data base into 4 time segments and sub partitions of each segment into Train, validation, test and the post- Brexit period.

Table 4. Summary of the partition intervals with the number of days used in the experiments.

		Partition 1:	Days interval	Partition 2:	Days interval	Partition 3:	Days interval	Partition 4:
Train	Begin	25/1/71		25/1/71		25/1/71		25/1/71
	End	22/3/79	2978	25/5/87	5964	26/7/95	8948	26/9/03
Validation	Begin	30/3/79		2/6/87		3/8/95		6/10/03
	End	25/5/87	2978	26/7/95	2976	26/9/03	2976	29/11/11
Test	Begin	2/6/87		3/8/95		6/10/03		7/12/11
	End	30/1/20	11930	30/1/20	8946	30/1/20	5960	30/1/20

Note: Note that a fourth interval was used for the post Brexit interval, from 2020-01-31 to 2020-03-27 with a total of 40 samples.

Although the three DL architectures have a different number of units, the number of inputs is common to all three, which allows comparing the observed memory. Additionally, the updating of parameters in all the structures is done by gradient search computed from a loss function. We have used both generalisation terms and early stopping to avoid overfitting (i. e. the model adjusts extremely well the performance on the training data but has a poor performance on unseen data). See, for instance, Goodfellow *et al.* (2016) or Bishop (2006).

The estimation of the parameters of a DL networks can be a complex task due to the fact that the ratio of the number of observations to the number of free parameters can be extremely low. Therefore there is a need for regularising the gradient estimation. The procedures we used to implement the regularisation were:

- Batch normalization – done after each layer, which consists on subtracting the mean of the outputs and normalising at each batch. The effect is an improvement of the estimation of the gradient.
- Dropout procedure – Which consists of a random selection of the units to be used to calculate the output of each layer during training. We used a probability of 50%.
- Early stopping of the training – The early stopping criterion is based on detecting the point on which the performance on the validation partition ceases to improve (see section Partition of the database). That is at each updating moment 'n' the validation error MSE $_{Validation}(\mathbf{W})$ is computed, and the stopping criterion is the point at which the validation MSE stops decreasing. On the experiments we have detected that the validation MSE has a convex behaviour, that consists of a first phase of decreasing values, afterwards a flat phase, which for the experiments at

hand could be as long as the decreasing phase, followed by a phase where the validation MSE increases.

- L2 penalizations on the weights – Consists on changing the performance index by adding a term on the norm of the weights so that there is trade-off between the accuracy on the training data and the norm of the weights. This is because a small L2 norm on the weights, prevents overfitting and is equivalent to having an effective lower number of degrees of freedom, that is

$$\text{MSE}_{Train}(\mathbf{W}) = \frac{1}{T}\sum_t (x(t+1) - x^{out}(t))^2 + \lambda \|\mathbf{W}\|^2 \quad (28)$$

- The Adam algorithm – By applying an exponential window on the gradient and moment estimation also implements an indirect regularisation effect, because the parameters of small absolute value and noisy reduced to zero.

All models were implemented in the pytorch framework (Paszke, 2019), and the experiments were done in a i7 workstation with 24 G of RAM with a Nvidia GPU NVIDIA Quadro K5200 8GB. Some of the experiments were done in the google colab (Bisong, 2019).

5. Empirical results

In this section, we analyse the accuracy of the out-of-sample forecasts obtained with the proposed DL architectures and the ARMA models used as a benchmark: 20 DFNNs, 30 CNNs, 10 LSTM NNs, and 11 ARMA models. We have computed the mean absolute percentage error (MAPE) during the out-of-sample period. Results are presented in Tables 5 to 8. Table 9 contains the MAPE of the best model for each architecture.

While the lowest MAPE was obtained with a DFNN of 3 layers of size 100 and 4 lags, overall, the MAPE values obtained with DFNNs and LSTM NNs are the ones showing the highest dispersion, as opposed to ARMA models, and CNNs to a lower extent.

Table 5. Forecast accuracy of DFNNs

Model	Lags	Layers	Size of layers	MAPE
DFNN(5,10,1)	4	3	10	7.065
DFNN(5,10,10,1)	4	4	10	4.241
DFNN(5,10,10,10,1)	4	5	10	3.409
DFNN(5,50,1)	4	3	50	2.781
DFNN(5,50,50,1)	4	4	50	1.296
DFNN(5,50,50,50,1)	4	5	50	2.718
DFNN(5,100,1)	4	3	100	1.243
DFNN(5,100,100,1)	4	4	100	3.118
DFNN(5,100,100,100,1)	4	5	100	3.218
DFNN(5,100,...,100,1)	4	12	100	17.706
DFNN(10,10,1)	9	3	10	2.947
DFNN(10,10,10,1)	9	4	10	1.828
DFNN(10,10,10,10,1)	9	5	10	4.178
DFNN(10,50,1)	9	3	50	2.660
DFNN(10,50,50,1)	9	4	50	1.415
DFNN(10,50,50,50,1)	9	5	50	2.528
DFNN(10,100,1)	9	3	100	1.722
DFNN(10,100,100,1)	9	4	100	2.223
DFNN(10,100,100,100,1)	9	5	100	6.358
DFNN(10,100,...,100,1)	9	12	100	14.977

Table 6. Forecast accuracy of CNNs

Model	Lags	Layers	Kernel size	Window	MAPE
CNN(10,100,1)	9	3	5	10	2.636
CNN(10,100,1)	9	3	10	10	3.221
CNN(10,100,1)	9	3	5	15	9.125
CNN(10,100,1)	9	3	10	15	2.920
CNN(10,100,1)	9	3	5	20	5.727
CNN(10,100,1)	9	3	10	20	4.373
CNN(10,100,100,1)	9	4	5	10	1.990
CNN(10,100,100,1)	9	4	10	10	3.092
CNN(10,100,100,1)	9	4	5	15	2.879
CNN(10,100,100,1)	9	4	10	15	5.254
CNN(10,100,100,1)	9	4	5	20	3.354
CNN(10,100,100,1)	9	4	10	20	6.329
CNN(10,200,1)	9	3	5	10	5.778
CNN(10,200,1)	9	3	10	10	3.886
CNN(10,200,1)	9	3	5	15	8.600
CNN(10,200,1)	9	3	10	15	3.844
CNN(10,200,1)	9	3	5	20	9.129
CNN(10,200,1)	9	3	10	20	5.612
CNN(11,100,1)	10	3	5	10	4.362
CNN(11,100,1)	10	3	10	10	3.545
CNN(11,100,1)	10	3	5	15	7.855
CNN(11,100,1)	10	3	10	15	2.604
CNN(11,100,100,1)	10	4	5	10	2.324
CNN(11,100,100,1)	10	4	10	10	4.161
CNN(11,100,100,1)	10	4	5	15	5.434
CNN(11,100,100,1)	10	4	10	15	4.308
CNN(11,200,1)	10	3	5	10	4.207
CNN(11,200,1)	10	3	10	10	5.504
CNN(11,200,1)	10	3	5	15	4.851
CNN(11,200,1)	10	3	10	15	4.947

Table 7. Forecast accuracy of LSTM NNs

Model	Layers	Input size	MAPE
LSTM(10,100,1)	3	5	11.651
LSTM(10,200,1)	3	5	25.435
LSTM(15,100,1)	3	5	9.146
LSTM(15,100,1)	3	10	7.991
LSTM(15,200,1)	3	5	12.948
LSTM(15,200,1)	3	10	2.007
LSTM(20,100,1)	3	5	9.101
LSTM(20,100,1)	3	10	8.438
LSTM(20,200,1)	3	5	7.256
LSTM(20,200,1)	3	10	1.966

Table 8. Forecast accuracy of ARMA models

Model	MAPE
ARMA(1,0)	3.645
ARMA(1,5)	3.672
ARMA(1,10)	3.672
ARMA(1,15)	3.670
ARMA(6,0)	3.673
ARMA(6,5)	3.666
ARMA(6,10)	3.687
ARMA(6,15)	3.671
ARMA(11,0)	3.673
ARMA(11,5)	3.631
ARMA(11,10)	3.660

Table 9. Best topology for each architecture

Model	MSE	MAPE
DFNN(5,100,1)	0.001	1.243
CNN(10,100,100,1)	0.077	1.990
LSTM(20,200,1)	0.002	1.966
ARMA(11,5)	0.070	3.631

These results somehow hint at the potential of DL techniques, but also highlight the importance of properly configuring the different architectures as well as implementing them appropriately. Our results are in line with those obtained by Chen *et al.* (2021), Dautel *et al.* (2020), Dodevski *et al.* (2018), Lara-Benítez *et al.* (2021), Qu and Zhao (2019), Ranjit *et al.* (2018), Sun *et al.* (2018), Yilmaz and Arabaci (2021) and Zhang (2018), who obtained the most accurate exchange rates predictions with variations of the LSTM architecture. Similarly, Galeshchuk and Mukherjee (2017) obtained a similar result with CNNs. In that sense, Byrne et al. (2018) showed that models embedding a high degree of coefficient variability yield forecast improvements at horizons beyond one month.

In spite of the fact that deep networks seem particularly suitable for exchange rate forecasting, Dautel *et al.* (2020) and Pascanu *et al.* (2013) have already noted the difficulty of implementing and tuning corresponding architectures.

5. Concluding remarks

This study examined the performance of several state-of-the-art deep learning techniques for exchange rate forecasting: deep feedforward networks, convolutional networks and a long short-term memory architecture. The main aim was to clearly explain how to configure the different architectures, as well as the tuning of the parameters and the regularisation techniques used to avoid overfitting. With that purpose, we designed an out-of-sample forecasting experiment and evaluated the accuracy of different topologies to predict the US/UK foreign exchange rate in the days after the Brexit took effect.

All three deep learning networks provided better forecasts that time-series models used as a benchmark, nevertheless the accuracy of the predictions varied remarkably depending on the chosen topology in each case. Thus, these results hint at the potential of deep learning techniques, but they also highlight the importance of properly configuring, implementing and selecting the different topologies.

Acknowledgements and funding

This research was supported by the project PID2020-118800GB-I00 from the Spanish Ministry of Science and Innovation (MCIN) / Agencia Estatal de Investigación (AEI). DOI: http://dx.doi.org/10.13039/501100011033

References

Alaminos, D., Salas, M. B., & Fernández-Gámez, M. A. (2021). Quantum computing and deep learning methods for GDP growth forecasting. *Computational Economics*, Forthcoming.

Alvarez-Diaz, M. (2008). Exchange rates forecasting: Local or global methods? *Applied Economics*, 40(15), 1969–1984.

Aminian, F., Dante, E., Aminian, M., & Waltz, D. (2006). Forecasting economic data with neural networks. *Computational Economics*, 28(1), 71–88.

Andrawis, R. R., Atiya, A. F., El-Shishiny, H. (2011). Forecast combinations of computational intelligence and linear models for the NN5 time series forecasting competition. *International Journal of Forecasting*, 27(3), 672–688.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

Bandara, K., Bergmeir, C., & Smyl, S. (2020). Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach. *Expert Systems with Applications*, 140, 112896.

Ben Taieb, S., Bontempi, G., Atiya, A. F., & Sorjamaa, A. (2012). A review and comparison of strategies for multiple-step ahead time series forecasting based on the NN5 forecasting competition. *Experts Systems with Applications*, 39(8), 1950–1957.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), 1798–1828.

Bi, J. W., Li, H., & Fan, Z. P. (2021). Tourism demand forecasting with time series imaging: A deep learning model. Annals of Tourism Research, 90, 103255.

Bishop, C. M. (2006). *Pattern recognition and machine learning (Information science and statistics)*. Springer.

Bisong, E. (2019). *Building machine learning and deep learning models on Google Cloud platform*. Apress, Berkeley, CA.

Byrne, J. P., Korobilis, D., & Ribeiro, P. J. (2018). On the sources of uncertainty in exchange rate predictability. International Economic Review, 59(1), 329–357.

Ca' Zorzi, M., & Rubaszek, M. (2020) Exchange rate forecasting on a napkin. *Journal of International Money and Finance*, 104, 102168.

Caporale, G. M., & Spagnolo, N. (2004). Modelling East Asian exchange rates: A Markov-switching approach. *Applied Financial Economics*, 14(4), 233–242.

Chen, W., Xu, H., Jia, L., & Gao, Y. (2021). Machine learning model for Bitcoin exchange rate prediction using economic and technology determinants. *International Journal of Forecasting*, 37, 28–43.

Claveria, O., & Torra, S. (2014). Forecasting tourism demand to Catalonia: Neural networks vs. Time series models. *Economic Modelling*, 36(1), 220–228.

Claveria, O., Monte, E., & Torra, S. (2015). Common trends in international tourism demand: Are they useful to improve tourism predictions? Tourism Management Perspectives, 16, 116–122.

Claveria, O., Monte, E., & Torra, S. (2016): Modelling cross-dependencies between Spain's regional tourism markets with an extension of the Gaussian process regression model. *SERIEs*, 7(3), 341–357.

Claveria, O., Monte, E., & Torra, S. (2017). Data pre-processing for neural networks-based forecasting: Does it really matter? *Technological and Economic Development of Economy*, 23(5), 709–725.

Claveria, O., Monte, E., & Torra, S. (2020). Time series features and machine learning forecasts. *Tourism Analysis*, 25(4), 463–472.

Clements, M. P., & Smith, J. (2001). Evaluating forecasts from SETAR models of exchange rates. *Journal of International Money and Finance*, 20(1), 133–148.

Colombo, E., & Pelagatti, M. (2020). Statistical learning and exchange rate forecasting. *International Journal of Forecasting*, 36(4), 1260–1289.

Crone, S. F., Hibon, M., Nikolopoulos, K. (2011). Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction. *International Journal of Forecasting* 27(3), 635–660.

Dodevski, A., Koceska, N., & Koceski, S. (2018). Forecasting exchange rate between Macedonian denar and euro using deep learning. *Journal of Applied Economics and Business*, 6(2), 50–61.

Enders, W., & Pascalau, R. (2015). Pretesting for multi-step-ahead exchange rate forecasts with STAR models. *International Journal of Forecasting*, 31(2), 473–487.

Fernández-Rodríguez F., Sosvilla-Rivero S., & Andrada-Félix J. (2004) Nearest-neighbour predictions in foreign exchange markets. In: Chen SH., Wang P.P. (eds.) *Computational Intelligence in Economics and Finance*. *Advanced Information Processing*. Springer, Berlin, Heidelberg.

Galeshchuk, S., & Mukherjee, S. (2017). Deep networks for predicting direction of change in foreign exchange rates. *Intelligent Systems in Accounting, Finance and Management*, 24(4), 100–110.

Gharleghi, B., Shaari, A. H., & Shafighi, N. (2014). Predicting exchange rates using a novel "cointegration based neuro-fuzzy system". *International Economics*, 137, 88–103.

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

Gradojevic, N., & Yang, J. (2006). Non-linear, non-parametric, non-fundamental exchange rate forecasting. *Journal of Forecasting*, 25(4), 227–245.

Graves, A., Mohamed, A., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 6645–6649

He, K., Ji, L., Wu, C. W. D., & Tso, K. F. G. (2021). Using SARIMA–CNN–LSTM approach to forecast daily tourism demand. *Journal of Hospitality and Tourism Management*, 49, 25–33.

Hewamalage, H., Bergmeir, C., & Bandara, K. (2021). Recurrent *ne*ural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1), 388–427.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., & Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6), 82–97.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.

Hong, Y., & Lee, T. H. (2003). Inference on predictability of foreign exchange rates via generalized spectrum and nonlinear time series models. *The Review of Economics and Statistics*, 85(4), 1048–1062.

Jamal, A. M. M., & Sundar, C. (2011). Modeling exchange rates with neural networks. Journal of Applied Business Research (JABR), 14(1), 1–5.

Jaworski, K. (2021). Forecasting exchange rates for Central and Eastern European currencies using country-specific factors. Journal of Forecasting, 40(6), 977–999.

Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. Proceedings of the 32nd International Conference on Machine Learning, Lille, France. JMLR: W&CP volume 37.

Kiani, K. M., & Kastens, T. L. (2008). Testing forecast accuracy of foreign exchange rates: Predictions from feed forward and various recurrent neural network architectures. *Computational Economics*, 32(4), 383–406.

Kirikos, D. G. (2000). Forecasting exchange rates out of sample: Random walk vs Markov switching regimes. Applied Economics Letters, 7(2), 133–136.

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

Lara-Benítez, P., Carranza-García, M., & Riquelme, J. C. (2021). An experimental review on deep learning architectures for time series forecasting. International Journal of Neural Systems, 31(3), 2130001.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444.

Lee, H. Y., & Chen, S. L. (2006). Why use Markov-switching models in exchange rate prediction? *Economic Modelling*, 23(4), 662–668.

Lin, C., Chiu, S., & Lin, T. (2012). Empirical mode decomposition-based least squares support vector regression for foreign exchange rate forecasting. *Economic Modelling*, 40, 76–80.

Lisi, F., & Schiavo, R. A. (1999). A comparison between neural networks and chaotic models for exchange rate prediction. *Computational Statistics and Data Analysis*, 30(1), 87–102.

López-Suárez, C. F., & Rodríguez-López, J. A. (2011). Nonlinear exchange rate predictability. *Journal of International Money and Finance*, 30(5), 877–895.

Meese, R. A., & Rogoff, K. (1983). Empirical exchange rate models of the seventies: Do they fit out of sample? *Journal of International Economics*, 14(1–2), 3–24.

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., & Khudanpur, S. (2010). Recurrent neural network based language model. *Interspeech*, 2(3), 1045–1048.

Nag, A. K., & Mitra, A. (2002). Forecasting daily foreign exchange rates using genetically optimized neural networks. *Journal of Forecasting*, 21(7), 501–511.

Ni, H., & Yin, H. (2009). Exchange rate prediction using hybrid neural networks and trading indicators. *Neurocomputing*, 72(13–15), 2815–2823.

Nikolsko-Rzhevskyy, A., & Prodan, R. (2012). Markov switching and exchange rate predictability. *International Journal of Forecasting*, 28(2), 353–365.

Park, C., & Park, S. (2013). Exchange rate predictability and a monetary model with time-varying cointegration coefficients. *Journal of International Money and Finance*, 37, 394–410.

Parot, A., Michell, K., & Kristjanpoller, W. D. (2019). Using artificial neural networks to forecast exchange rate, including VAR-VECM residual analysis and prediction linear combination. *Intelligent Systems in Accounting, Finance and Management*, 26(1), 3–15.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. 30th *International Conference on Machine Learning, ICML* (3), 1310–1318.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E. DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 8026–8037.

Qu, Y., & Zhao, X. (2019). Application of LSTM neural network in forecasting foreign exchange price. *Journal of Physics: Conference Series*, 1237, 042036.

Ranjit, S., Shrestha, S., Subedi, S., & Shakya, S. (2018). Comparison of algorithms in foreign exchange rate prediction. In Proceedings on 2018 IEEE 3rd international conference on computing, communication and security, ICCCS 2018 (pp. 9–13), Institute of Electrical and Electronics Engineers Inc.

Rossi, B. (2013). Exchange rate predictability. *Journal of Economic Literature*, 51(4), 1063–1119.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61, 85–117.

Sermpinis, G., Dunis, C., Laws, J., & Stasinakis, C. (2012). Forecasting and trading the EUR/USD exchange rate with stochastic Neural Network combination and time-varying leverage. *Decision Support Systems*, 54(1), 316–329.

Sermpinis, G., Theofilatos, K., Karathanasopoulos, A., Georgopoulos, E. F., & Dunis, C. (2013). Forecasting foreign exchange rates with adaptive neural networks using radial-basis functions and particle swarm optimization. *European Journal of Operational Research*, 225(3), 528–540.

Sezer, O. B., Gudelek, M. U., & Ozbayoglu, A. M. (2020). Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied Soft Computing*, 90, 106181.

Shen, F., Chao, J., & Zhao, J. (2015). Forecasting exchange rate using deep belief networks and conjugate gradient method. *Neurocomputing*, 167, 243–253.

Siegelmann, H. T., & Sontag, E. D. (1991). Turing computability with neural nets. Applied Mathematics Letters 4.6 (1991): 77–80.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 27, 3104–3112.

Tenti, P. (1996). Forecasting foreign exchange rates using recurrent neural networks. *Applied Artificial Intelligence*, 10(6), 567–582.

Teräsvirta, T., Van Dijk, D., & Medeiros, M. C. (2005). Linear models, smooth transition autoregressions, and neural networks for forecasting macroeconomic time series: A re-examination. International Journal of Forecasting 21 (4), 755–774.

Tripathi, M., Kumar, S., & Kumar Inani, S. (2021) Exchange rate forecasting using ensemble modeling for better policy implications. *Journal of Time Series Econometrics*, 13(1), 43–71.

Sun, S., Wei, Y., & Wang, S. (2018). AdaBoost-LSTM Ensemble Learning for Financial Time Series Forecasting. In: Shi Y. *et al*. (Eds.) *Computational Science* – ICCS 2018. ICCS 2018. Lecture Notes in Computer Science, vol 10862. Springer, Cham.

Yilmaz, F. M., & Arabaci, O. (2021). Should deep learning models be in high demand, or should they simply be a very hot topic? A comprehensive study for exchange rate forecasting. *Computational Economics*, 57(1), 79–98.

Yu, L., Wang, S., & Lai, K. K. (2005). A novel nonlinear ensemble forecasting model incorporating GLAR and ANN for foreign exchange rates. *Computers & Operations Research*, 32(10), 2523–2541.

Zhang, B. (2018). Foreign exchange rates forecasting with an EMD-LSTM neural networks model. *Journal of Physics: Conference Series*, 1053, 12005.

Zhang, G. P., & Berardi, V. L. (2001). Time series forecasting with neural network ensembles: An application for exchange rate prediction. *Journal of the Operational Research Society*, 52(6), 652–664.

UBIREA

Institut de Recerca en Economia Aplicada Regional i Públic
Research Institute of Applied Economics

WEBSITE: www.ub-irea.com • **CONTACT:** irea@ub.edu

AQR

Grup de Recerca Anàlisi Quantitativa Regional
Regional Quantitative Analysis Research Group

WEBSITE: www.ub.edu/aqr/ • **CONTACT:** aqr@ub.edu

Universitat de Barcelona
Av. Diagonal, 690 • 08034 Barcelona