International Journal of Computational Economics and Econometrics

ISSN online: 1757-1189 - ISSN print: 1757-1170
https://www.inderscience.com/ijcee

Reservoir computing vs. neural networks in financial forecasting

Spyros P. Georgopoulos, Panagiotis Tziatzios, Stavros G. Stavrinides, Ioannis P. Antoniades, Michael P. Hanias

DOI: 10.1504/IJCEE.2021.10041721

Article History:

Received:	10 March 2021
Accepted:	13 July 2021
Published online:	30 November 2022

Reservoir computing vs. neural networks in financial forecasting

Spyros P. Georgopoulos, Panagiotis Tziatzios and Stavros G. Stavrinides*

School of Science and Technology,
International Hellenic University,
Thermi Campus,
Thessaloniki, Greece
Email: spgeorgop@gmail.com
Email: p.d.tziatzios@gmail.com
Email: s.stavrinides@ihu.edu.gr
*Corresponding author

Ioannis P. Antoniades

Physics Department,
Aristotle University of Thessaloniki,
Thessaloniki, Greece
Email: iantoniades@auth.gr

Michael P. Hanias

Department of Physics,
International Hellenic University,
Kavala Campus,
St. Lucas, Kavala, Greece
Email: mhanias@physics.ihu.gr

Abstract: Stock market prediction techniques are a major research area, thus, extracting time-dependent patterns for the existing predictive models is of major significance. In this work, we compare forecasting performance of the nonlinear model of recurrent neural networks (RNN) in two implementations, LSTM and CNN-LSTM, to the relatively novel approach of reservoir computing (RC), and in specific, the particular class of the echo state networks (ESN). This comparison focuses on exploiting data latent dynamics, in performing efficient training and high quality predictions of the evolution of real-world financial data. Applying a multivariate scheme to a stock market index without any stationarity techniques, a definite precedence of the ESN-RC over both types of RNN's in computational efficiency as well as prediction quality, emerges. Finally, the implemented approach is friendly to the trader, since specific values of a stock market timeseries provide with a frame allowing for in time forecasting, under real-world circumstances.

Keywords: deep learning; neural networks; reservoir computing; machine learning; time series analysis; financial-economic forecasting; algorithmic comparisons.

Reference to this paper should be made as follows: Georgopoulos, S.P., Tziatzios, P., Stavrinides, S.G., Antoniades, I.P. and Hanias, M.P. (2023) 'Reservoir computing vs. neural networks in financial forecasting', *Int. J. Computational Economics and Econometrics*, Vol. 13, No. 1, pp.1–22.

Biographical notes: Spyros P. Georgopoulos is a Physicist and holds a MSc in Photonics and another one in Data Science. His research interests include computational physics related to ultra-fast laser interaction with materials and laser processing, as well as time series analysis using deep learning techniques with applications in finance and economics. He currently works as a data scientist.

Panagiotis Tziatzios is a Mathematician with a MSc in Data Science. His research interests include nonlinear time series analysis and forecasting, as well as study of chaotic systems. He is currently working as a software engineer in developing web applications.

Stavros G. Stavrinides is a Physicist with a MSc in Electronics and a PhD in Chaotic Electronics. His research interests include the design of analogue and mixed-signal electronic circuits, chaotic electronics/synchronisation and their applications (with emphasis on security and secure communications), memristors nonlinear time series analysis and econophysics. He is currently with the School of Science and Technology at the International Hellenic University, Greece. He has authored or co-authored more than 90 journal and conference papers and he has participated, as a researcher, in several Greek-nationally and internationally (EU, NATO) funded projects. Finally, he is an IEEE senior member.

Ioannis P. Antoniades obtained his Bachelor in Physics with Honours from the University of Chicago, USA in 1992 and his PhD in Monte Carlo and Molecular Dynamics Atomistic Simulations of Grain Boundaries in Alloys from the Aristotle University of Thessaloniki, Greece, Department of Physics, Solid State Section. His current research interests and activity focuses on complex systems statistics, modelling, simulation and analysis with a broad range of applications including financial systems (econophysics), solid state, electronics (memristive systems, MOSFETs) and social systems. He has 31 peer reviewed publications (15 journals, 2 book chapters, 14 in conference proceedings) and has participated in several national, European and NATO funded research programs.

Michael P. Hanias is an Associate Professor in Physics Department at the International Hellenic University, Greece. He holds a PhD in Semiconductor Physics. His research interests include nonlinear properties of semiconductors, chaos theory, neural networks, time series prediction and econophysics. He has authored more than 100 journal and conference papers, and has participated in several national and international funded projects.

1 Introduction

Decision making and budget planning are of crucial importance and vital, in establishing and maintaining a healthy and sturdy business in the financial credit sector (Balli et al., 2020; Alghalith and Floros, 2020; Mejdoub and Arab, 2019). Therefore, forecasting various indicators or characteristic features in the form of time series or patterns, has emerged as a key-field study, during the last decades. Stock market appears to be a domain where, next to the traditional information advantage, apt forecasting provides with clear precedence. To this direction the importance of machine learning is apparent. The possible outcomes and variations of machine learning applications are fully capable to cover a wide spectrum of needs and ideas. More specifically, by the beginning of the 21st century neural networks have been at the focus for addressing problems such as fraud detection, customer segmentation, sales prediction, as well as, stock market prediction, to mention a few (Heaton et al., 2016; Tsiptsis and Chorianopoulos, 2011; Roy et al., 2018; Nelson et al., 2017; Chen et al., 2015; Sherstinsky, 2020).

Neural networks and similar models are extensively studied as deep investigation tools with specialisation on time series prediction, apparently holding a promise to address real world problems. Such problems are highly related to financial time series analysis that pertains to the development of any business or association.

Most of the published works (Qian and Chen, 2019; Qi and Zhang, 2008; Chan et al., 1977; Vaisla and Bhatt, 2010; Köhler and Lorenz, 2005) in the field, tend to deal with and legitimately point out the necessity of *stationarity* techniques. This work aims to demonstrate the reservoir computing (RC) capability not only in accurately fitting real datasets, but also in extracting knowledge about the low-dimensional dynamics of a system out of noisy temporal data. To this end, no stationarity techniques need to be applied in pre-processing raw data. The only pre-processing is a mere normalisation of the time series values, in order to map it to a particular range (usually the unit interval).

The presented hereby work aims to shed some light on the performance of two versions of RNNs, specifically the classical long-short term memory (LTSM) and the convolutional neural network (CNN) extended long short-term memory (LSTM) versus the more contemporary algorithms of RC, under no de-trending pre-processing. Initially, a typical deterministic chaotic macroeconomic model, namely the Vosvrda system, is studied, in order to tune our models and gain the necessary feedback for a more complete comparison regarding a real-world, typical stock-market series. Then, utilising the S&P-500 index, a multivariate prediction scheme is addressed. In this scheme, given the closing price at day t together with the opening price at day $t+1$, the system outputs a forecast for the closing value of day $t+1$, leaving a 'betting' window of several hours (opening to closing) to invest or draw money. Such an approach fully fits the needs of brokers. The emerging results show that RC outperforms both versions of RNN, when applied to real-world financial data, both in terms of computational efficiency and prediction accuracy.

2 Fundamental theoretical elements

There are many approaches in implementing artificial intelligence (AI), ranging from simple learning algorithms to neural implementations and deep learning techniques. In the lines below, a brief theoretical description regarding the classic LSTM and the CNN-LSTM recurrent neural networks is presented. Additionally, a theoretical description of RC with focus on echo state networks (ESNs) is apposed.

2.1 Recurrent neural networks

The idea behind neural networks and furthermore deep learning techniques is mimicking the human brain. It is common knowledge that human brain does not think in a manner of frames or from scratch. For instance, as the reader goes through the lines of this paper, one understands each word based both on understanding and the short memory of previous words. Additionally, the reader's developed training on understanding similar lines and readings allows for achieving improved understanding. This approach is in a coarse way the central idea behind RNNs.

In terms of temporal dynamics, this approach of reasoning usage stems from the fact that the connections between the nodes of an RNN form a directed graph, along a temporal sequence. The ability to feed sequential input, producing sequential output, by sharing knowledge between the nodes, is the essence of a RNN system and has resulted in breakthrough achievements and improvements in many domains of AI, like natural language processing (NLP), image captioning, speech recognition, time series forecasting, etc. to mention a few (Hori et al., 2017; Rao et al., 2017; Wang et al., 2016; Mao et al., 2014; Maknickienė et al., 2011; Dunis and Huang, 2002)

Briefly, in the case of an RNN strongly affected by the decision that was taken in time instant $t-1$ (the system has two inputs one for the past and one for the present state), every decision that is being taken at time instant t is a combined response based on the past and the present input, just like the way humans make decisions. This feedback loop, connecting every neuron to its past, is the main difference between an RNN and any other feed-forward network. In Figure 1 the general structure of a recurrent neural network is presented. The sample values (signals), represented by vector $X(t)$, are propagated forward to the hidden neuron layers. The recurrent character of the network stems from the fact that every neuron in the layer stores the input from all the previous steps and merges this information with the input of the current step. Consequently, it also captures some information regarding the correlation between the current data step and the previous steps. The decision at a time step $t-1$ affects the decision taken at time t. Finally, the last hidden layer passes on the values to the output layer, where we get as an output $(Y(t+1))$, which the desired vector of values in the desired range.

The basic function of an RNN cell is presented in Figure 2 and it has been taken from Olah (2015). As already mentioned, the feedback scheme connecting every neuron to its past is the main reason that RNN differs from every other feed-forward network, and it can be graphically and mathematically defined for a cell, as follows:

$$h_t = \phi(W\mathbf{x}_t + Uh_{t-1}) \tag{1}$$

where h_t corresponds to the hidden state at time t and x_t is the input at the same time; x_t is modified by a weight matrix W and then added to the hidden state of the previous time step t−1, multiplied by a hidden state matrix U bearing the name transition matrix. The weight matrices' values neglect the importance given by the model to the current and the previous state. Finally, ϕ is a logistic sigmoid function or a tanh function, depending on the nature of the dataset and the task the RNN has been built for. Either way, it needs to be a nonlinear and differentiable function.

Figure 1 A schematic representation of the general structure of a recurrent neural network (see online version for colours)

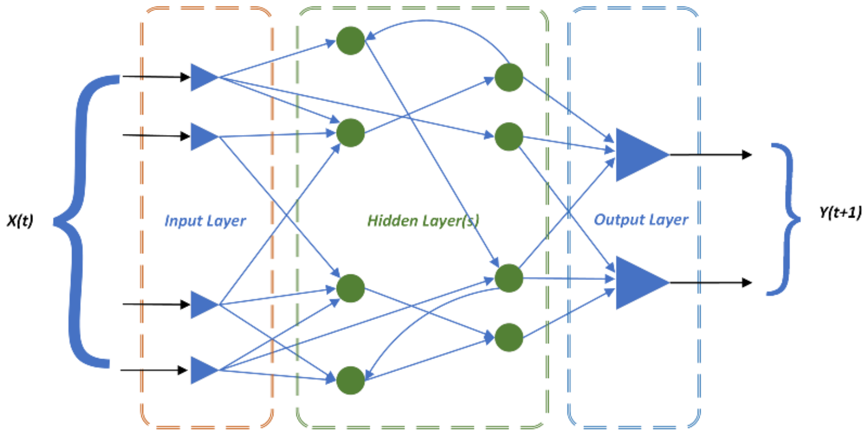
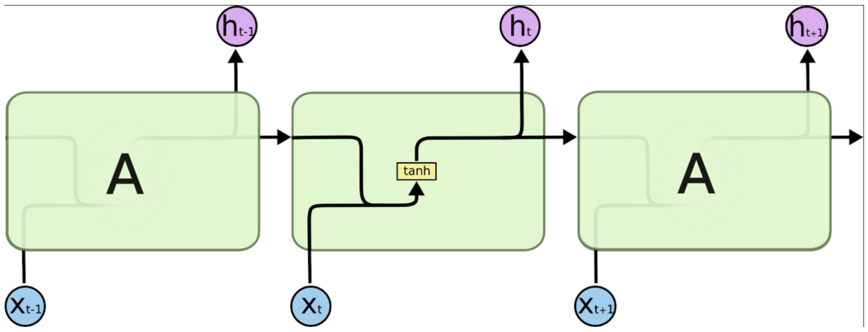


Figure 2 The basic RNN architecture, consisting of only one layer; the tanh function at time instant t (see online version for colours)



2.1.1 Long short-term memory

Given that RNN enable the modelling of time-depended data issues such as stock market, stock prediction, text generation, and others, we cannot avoid referring to the main obstacle in this type of models; the decay of information through time. This major problem known as 'vanishing gradient problem', mostly occurs when dealing with large time series of data. It was first discovered by Sepp Hochreiter in 1991, who contributed to the development of neural network and deep learning as this is considered today (Hochreiter and Schmidhuber, 1997).

The gradient descent algorithm is supposed to find the global minimum of the cost function and propagate back the calculated error in order to update the weights of the network in a beneficial way, in terms of the suitable metric used. The case with the RNNs is that not just the neurons prior to the output layer, need to adjust their weights accordingly, but all of the neurons of the previous time-steps. When working with long sequences, the derivatives of ϕ function become increasingly smaller. Consecutive multiplications with the same small values finally result into increasingly lower gradients. After several steps, the gradients may vanish completely, which is catastrophic for the network, because from that point and on, there is no more training and thus no more learning. So, since the output is based on earlier inputs, the system will end up with training neurons at time t with inputs not trained at all. Similarly, it is also common for gradients having large values to exhibit exploding gradient problems, resulting into increasinlgy higher values, further leading the system to become unstable. The exploding vanishing problem can be solved by clipping the gradient at a pre-defined threshold. On the other hand, the vanishing problem is way more complex to solve. To address this problem, the LSTM block was proposed, once again, by Sepp Hochreiter and his PhD supervisor Jurgen Schmidhuber in Hochreiter and Schmidhuber (1997). Their chain like structure can be seen in Figures 3 and 4; note that both schemas were taken from Olah (2015). The four layers of information processing displayed in those two figures, compared to the only one of the basic RNN depicted in Figure 2 is obvious.

Figure 3 The LSTM basic architecture, depicting all four layers of operations from the input to the output (see online version for colours)

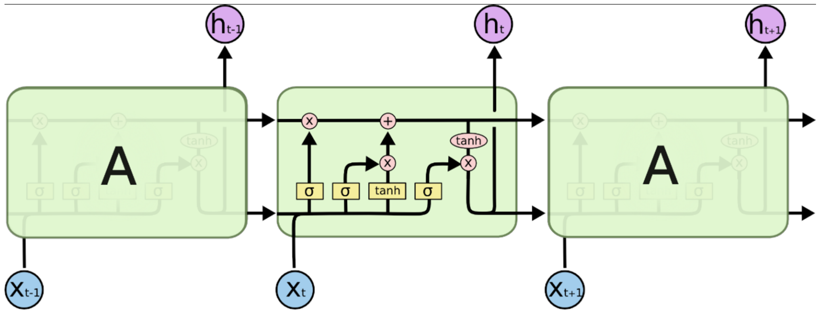
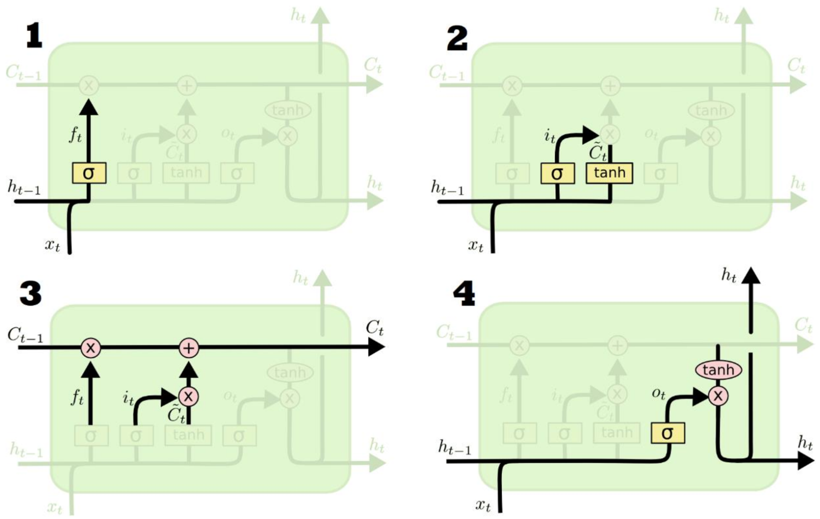


Figure 4 The processing parts within a single LSTM cell (see online version for colours)



The internal mechanisms used, called gates, achieve regulation of the information flow by learning which parts of the sequences of flowing data are important to keep. This occurs in cells that transmit only the relevant information needed for the task.

Let $U = [0,1]$ to be the LSTM unit interval and then let ± U = [–1, 1]. Let also $x^{(t)}$ be the sequence of the inputs. Every LSTM cell can be described by the hidden state h and the cell state c. The complete mathematical function for a cell will be denoted as Ξ and, as seen below, can take three arguments and returns two outputs:

$$(h^{(t)}, c^{(t)}) = \Xi(h^{(t)}, c^{(t)}, \mathbf{x}^{(t)}) \tag{2}$$

with every argument $\in \pm\, U$ except for the input value $x^{(t)}$ which belongs to the coordinate space $\mathbb{R}^d$

The outputs are leaving the cell at time t and being fed into the same cell at time $t+1$ along with an element from the sequence of $x^{(t)}$. As the mathematical equations illustrate below, the hidden state and the input element are fed into three gates that describe the operations inside every cell. It is obvious that the sigmoid function σ, implies that the output of every gate is a value $\in U$ (Karim et al., 2018; Livieris et al., 2020):

$$\begin{aligned}
&forget^{(t)}(\mathbf{x}^{(t)}, h^{(t-1)}) = \sigma(\mathbf{W}^T_{forget,x}\mathbf{x}^{(t)}, w_{forget,h}h^{(t-1)} + b_{forget}) \\
&input^{(t)}(\mathbf{x}^{(t)}, h^{(t-1)}) = \sigma(\mathbf{W}^T_{input,x}\mathbf{x}^{(t)}, w_{input,h}h^{(t-1)} + b_{input}) \\
&output^{(t)}(\mathbf{x}^{(t)}, h^{(t-1)}) = \sigma(\mathbf{W}^T_{output,x}\mathbf{x}^{(t)}, w_{output,h}h^{(t-1)} + b_{output}) \\
&update^{(t)}(\mathbf{x}^{(t)}, h^{(t-1)}) = \tanh(\mathbf{W}^T_{x}\mathbf{x}^{(t)}, w_{h}h^{(t-1)} + b)
\end{aligned}$$

where $W_{forget,x}$, $W_{input,x}$, $W_{output,x}$, W_x, $w_{forget,h}$, $w_{input,x,h}$, $w_{output,h}$ and b_{forget}, b_{input}, b_{output}, b are weight vectors and biases that are produced during the training process.

As regards the flow of information within the cell, the forget gate outputs a number in the range [0, 1]. with values near zero being the unwanted information that needs to be forgotten, and the values close to one the ones to be kept. The next step combines two procedures. The sigmoid layer called the 'input gate' decides the cell values that need to get updated, and the tanh layer fulfills this update by creating a vector $c_t^{(t)}$ consisting of these new values, to be added to the cell state. The multiplication of $c_t^{(t-1)}$ with the $forget^{(t)}$ will result in an updated new $c_t^{(t)}$ that has finally forgotten everything the cell wanted to get rid of. The last stage is associated with the output of the cell. As seen in the last part of the graph, the first sigmoid filters the parts of the cell to be pushed forward to the output, along with a tanh function that scales the information values between [–1, 1] to ensure that the output contains all the parts needed.

In mathematical terms, the new cell and the hidden states at time t can be written as follows:

$$\begin{aligned}
&c^{(t)} = forget^{(t)} \cdot c^{(t-1)} + input^{(t)} \cdot update^{(t)} \in \pm U, \\
&h^{(t)} = output^{(t)} \cdot \tanh(c^{(t)}) \in \pm U
\end{aligned}$$

The above can be generalised for a number of n LSTM cells forming an LSTM layer, which we denote as Ξ_n corresponding to the concatenation of the n cells that produce and use their own weights and biases.

Thanks to the aforementioned gate-based mechanisms, LSTM has proved to be the cure regarding the vanishing and exploding gradient problems, producing high quality results and is considered well-suited to regression tasks related to time series analysis.

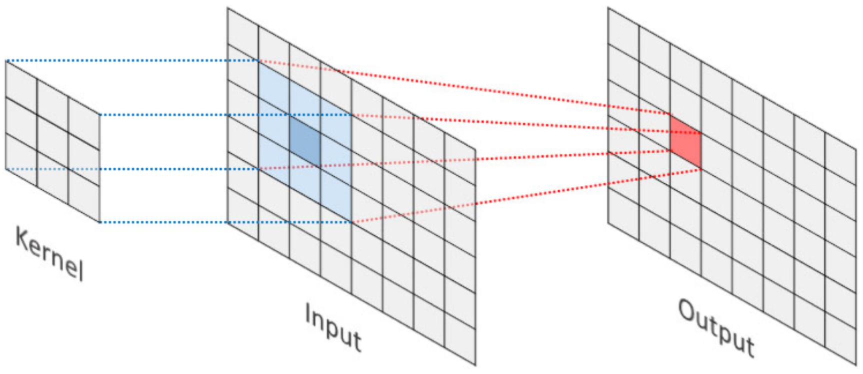
2.1.2 Convolutional neural networks

A CNN is a class of deep learning networks that among others, is most commonly applied in image classification tasks, such as fraud detection, search engines and recommendation systems (Zhang et al., 2018; Wang et al., 2016; Heryadi and Warnars, 2017; Fu et al., 2016; Zheng et al., 2018; Han et al., 2018). Its name indicates the use and exploitation of the convolution mathematical linear operation and thus is considered to be a mathematical construction. These mathematical operations take part in the first building block of a CNN which is called 'convolution'. This first layer is related to feature extraction procedures, where an array of small numbers, called the kernel, is being applied as a filter across the tensor, which is another array of numbers representing the input data, further producing an output tensor, called the feature map (Wang, 2015).

The filter (set of weights) is chosen to be smaller in size than the input and this results in several multiplications – for multiple times at different points with the input. In specific, the filter is applied systematically to each overlapping part or filter-sized patch of the input data, from left to right and top to bottom, as depicted in Figure 5. It is evident that if the model's filter is designed to detect a specific type of pattern in the input, then recognition of this pattern anywhere in the image becomes detectable indeed.

The second building block of a CNN is the pooling layer, which is responsible for progressively reducing the spatial size of the representation, further reducing the parameter number and thus, the computational complexity of the network. Pooling layers achieve a reduction of the data dimensions by combining neuron-clusters outputs to a single neuron in the next layer. Finally, the produced vector, i.e., the produced flatten matrix, is being fed into a final layer called the 'fully connected layer'. This connects every neuron of a layer to every other neuron in another layer just like a traditional multi-layer perceptron. In this work, we are not interested in the common architecture of CNNs mainly used for image processing tasks, rather than the first layer that is responsible for feature extraction.

Figure 5 Example of a filter applied to a 2D input creating, a feature output map (see online version for colours)



2.2 Reservoir computing

RC is a broad subject combining the fields of artificial neural nets and complex dynamical systems. In general, a reservoir computer can be implemented by a large number of interacting nonlinear dynamical systems (consisting the reservoir), driven by some external input signal and issuing an output signal. RC started out as a RNN-based framework suitable for temporal/sequential information processing (Jaeger and Haas, 2004; Verstraeten et al., 2007; Lukoševičius and Jaeger, 2009). The reservoir transforms nonlinearly an input time series into a very high-dimensional space, such that the features of the inputs can be effectively extracted by a simple learning algorithm. In other words, our quest is to build a model capable of approximating the state space of the dynamical system, an approach known in bibliography as empirical system identification (Cuchiero et al., 2020). Therefore, instead of RNNs, we make use of RC to model complex dynamical systems.

The main characteristic of RC is that the weights of the connections between units (nodes), within the reservoir, are not trained. Only the weights of the output to reservoir units layer are trained by linear regression or another simple machine learning algorithm. Compared to standard RNNs, this simple and fast training process greatly reduces the computational cost of learning, which is the major advantage of RC (Jaeger, 2002). Additionally, RC has been successfully applied to many computational problems, such as temporal pattern classification, prediction, and generation. It has to be noted that in order to achieve excellent computational performance, it is necessary to appropriately represent sample data and optimally design the reservoir.

A particular class of RC, namely the ESN, has been developed within the concept of computational predictive models introduced by Jaeger (2012). It was designed based on the idea of feeding any input signal $u(t)$ to a random dynamical system, the reservoir. The reservoir is made up of a very large number (several hundreds or thousands) of randomly connected neurons (nodes). In the training step, the reservoir state is read and mapped to the desired output Y_{true} by adjusting the weight of the output layer to the reservoir in a *single* training step. It is apparent that such an approach results into massive reduction of computational time and effort, since the training procedure takes place only at the readout section, while the reservoir connected nodes remain unchanged. The basic structure of an RC is shown in Figure 6.

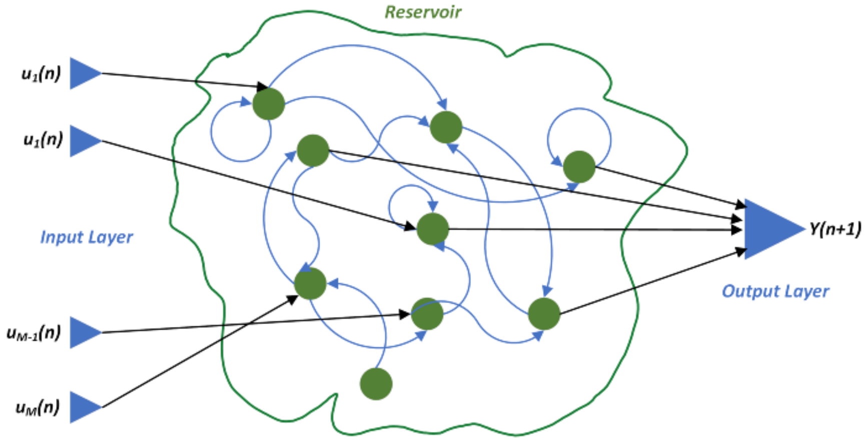
The internal state of a reservoir containing M nodes at a time step n is described by a vector variable $x(n) = (x_1, x_2, ..., x_M)$, where each component $x_i(n)$ denotes the internal state of node i at time step n. The evolution of $x(n)$ is defined by the equation (3), which determines the value of x at the next time step $(n + 1)$, depending on the value of x at the previous time step.

$$x(n+1) = (1-\alpha)x(n) + \alpha * \tanh(Wx(n) + W^{in}u(n+1)) \tag{3}$$

It is apparent from the above equation that the status update process is assigned to the tanh function. Moreover, each node state $x(n + 1)$ is derived from its current state $x(n)$, combined with the input data $W^{in}u(n + 1)$, and a nonlinear expression describing the interaction with the state of all the reservoir nodes, which is represented by the term $Wx(n)$. W represents the (time independent) random weight matrix of neuron connections and ensures the network's recurrent loops, leading to the essence of this model, i.e., memory retain. The feature of $memory\ retain$ in an ESN is similar to the LSTM memory of its previous time-steps and comes with the name of $echo\ state$

property. This feature is directly related to the spectral radius ρ of the matrix W, which is defined as the largest eigenvalue of the matrix. Spectral radius ρ is a physical parameter of the reservoir depending on the connection weight distribution among nodes and is of high significance. A proper value of ρ ensures that the output of the reservoir computer stays bounded at all times, i.e., it does not explode to infinity or collapse to zero. According to the literature, and for ensuring the specific echo state property, ρ needs to be less than 1 (Jaeger, 2001). Finally, α represents the *leaking rate* of the system, defining the speed at which the reservoir updates its dynamics in terms of how quickly the input is being fed (leaked) into the reservoir; α ranges from 0 to 1.

Figure 6 A schematic representation of a typical RC scheme (see online version for colours)



Regarding the training process, the weight matrix W_{out}, contains the weights that are connecting the reservoir nodes with the desired output node(s). As already mentioned, the novelty of ESN is that the training part takes place directly in the output layer. Since reservoir is a linear combination of the output weights with the reservoir states, it is apparent that:

$$Y_{true} = W_{out} * X, \tag{4}$$

with W_{out}, being the output-weights matrix. Additionally, it should be underlined that $Y_{true} \in R^{N_y * T}$, while $X \in R^{1*Nu*N_x}$. The later is the concatenated matrix $[1, u(t), x]$, where x is the column vector that refers to the reservoir states for every time-step t of the training process. N_y refers to the size of the target data points; N_u is the size of the input signal; and T is the number of data points in the training dataset.

The purpose during the training stage is to minimise the root mean squared error (RMSE) between the reservoir computer output series and the true series. The system described in equation (4) is an over-determined one, since T is much bigger than the product $(1 * N_u * N_x)$, and there are a plethora of ways to solve it. The most common way is by using what is known as the *Tikhonov regularisation* (Lukoševičius, 2012), which is based on ridge regression and gives the following solution to our system:

$$W_{out} = Y_{true} * X^T (X * X^T + \beta I)^{-1}, \tag{5}$$

where β is the regularisation coefficient and I is the identity matrix. Equation (5) shows that training is achieved in just one step, just by solving it.

3 Implementation

In this section the implementations of the forecasting models, theoretically described above, are presented. In all cases, one step forecasting over a 200 step testing set has been attempted. Moreover in all cases the first 100 data points were not part of the training set due to a technique commonly used in the ESN implementations. The idea behind this technique is to feed the ESN with a part of the data sequence to wash out the initial random state. Thus, these first 100 data points were ruled out of the training sets of LSTM and CNN-LSTM to ensure that all of three models will be trained in the exact same training sets. This is also shown in the corresponding sections, in Figures 7 and 9 by the name 'initial size' and a red straight line. In the same figures the train and testing areas are also displayed.

3.1 The LTSM RNN implementation

The LSTM model utilised in this work was built using a sliding window technique. The idea behind rolling techniques is to over-sample input data and create new sequences of observations. The impact of a sliding window is conducting regressions over and over again with sub-samples of the original full sample. In our case, the LSTM model was fed with a number of points (n) and then a prediction for the n + 1 day was attempted. Then, the RNN was re-fed with the [2, n + 1] points of the original time series and then the prediction of the $(n+2)^{\text{th}}$ point was attempted, and so on.

As already mentioned, all the data fed to the LSTM model were normalised by being linearly mapped to the unit interval [0, 1], without distorting the differences in the initial data, according to:

$$x_{normalised} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \tag{6}$$

The optimisation of an LSTM can be performed in a plethora of ways. In this work the model that was implemented, consisted of an input layer, one hidden layer and a final output layer. The units (neurons) of every layer, contributing to the data processing, decision and error propagation, were chosen to be 20 for each layer, while the sliding window was set to a frame of two points. Moreover, every layer was carrying a 20% dropout.

Dropout is a popular tactic, aiming in regularising any deep neural network in a way that the over-fitting risk is eliminated. It is a computationally cheap way to improve model performance and ensure the validity of the results. According to this, some of the input units, along with some units located at the hidden layers, are randomly excluded from activation and weight updates, during the network's training stage.

The activation function used was the tanh function, while the optimising function was set to $Adam$ (adaptive moment estimation), a function that updates network weights; the most commonly used in time series prediction. $Adam$ has low memory requirements and is appropriate for noisy, as well as non-stationary datasets.

Finally, the batch size used to train our model was set to 10. Batch size represents the number of sequences that are trained together. Small sized batches ensure the reliability of the fitting, minimising randomness as much as possible.

3.2 The CNN-LSTM model implementation

The CNN-LSTM architecture utilises CNN layers for feature extraction on the input time series data, combined with LSTM layers to carry out the sequence prediction task. The implementation of the model is exactly the same in terms of strategy with the previous described LSTM model, and the novelty only relates to the CNN layers added before the LSTM ones. Specifically, two convolution layers, consisting the convolution kernel, convolve with the inputs in order to produce the output matrix, using 256 and 128 filters, respectively. These kernel filter parameters in each convolution layer are not to be decided rather than just provide a number for them. The filter values are learned automatically by the neural network through the training process. One-dimensional (1D) pooling layer right after the convolutional one was utilised. That is a max pooling layer of size one was added, aiming to reduce the amount of parameters and computation in the network, and hence to also control over-fitting.

3.3 The RC implementation

In the case of a RC implementation, an ESN model was utilised. A similar to the above described process was followed, but with some differentiation included. The input data were fed into the ESN without being normalised to a specific interval (raw data). Tuning of the the model parameters was performed by thorough grid-search, out of which the optimal parameters occurred and are presented in the next sections.

The reservoir, whose size is proportional to the number of state variables of the system, was composed of 150 nodes. Based on a trial and error approach, the leaking rate was set to 0.1 (10%), defining the speed of the dynamics of the reservoir nodes and hence, as its name suggests, it affected the volume information leaking out of the reservoir. Thus, further affecting the short-term memory of the network.

The values of the elements of the matrix of input weights (W_{in}) were chosen from a uniform distribution in the interval [–1, 1], whereas the matrix of the reservoir weight matrix W were chosen from a uniform distribution in the interval [0, 1] with a sparsity level of 10%.

Finally, during the training stage, the spectral radius ρ was optimally set to 0.7 (based on trial and error scheme), which was pursuing minimisation of both the convergence time and the round mean square (RMS) of the fitting between the RC output and the real data. It is noted that spectral radius ρ sets the scaling in W and consequently the scaling of the dynamics of the reservoir.

4 Comparative results

In this section the comparison of the performance demonstrated three AI approaches, namely the LSTM RNNs, the CNN-LSTM and the RC-ESNs, while predicting the evolution of time series, is presented. Implementing the above described networks, the comparison of their results in the case of a typical nonlinear, chaotic system (the Vosvrda macroeconomic model) and real-world financial data (the S&P-500 stock market index), appears. In both cases, expanding our understanding of the systems via parameter tuning, towards the minimisation of errors and high quality predictions, takes place.

The data used for the whole process is the Vosvrda macroeconomic equation system that was generated in the MATLAB environment and the S&P-500 stock market index, downloaded from https://finance.yahoo.com. The implementation of neural networks (LSTM and CNN-LSTM) was carried out using Python programming language and the open-source software library of TensorFlow 2.4. On the other hand, the RC algorithm was developed in MATLAB R2018b environment.

4.1 The Vosvrda macroeconomic model

Chaos theory demonstrates impressive, interdisciplinary applications, and induces self-organisation/similarity feedback loops and self-repetition patterns, which are governed by deterministic laws, highly correlated and sensitive to initial conditions.

The Vosvrda system of equations, refers to an ideal macroeconomic model, proposed by Vosvrda (2001) and Vosvrda et al. (2001). The same system was initially proposed as a microeconomic model (Bouali, 1999; Buali, 2002). It has been found that it demonstrates deterministic chaotic behaviour (Pribylova, 2009; Hanias et al., 2019) and it has attracted the interest of the econophysics community (Antoniades et al., 2020). This system was utilised as a paradigm for introducing a data series into our forecasting models, namely the LTSM and RC, and it is mathematically defined by the following set of nonlinear ODEs:

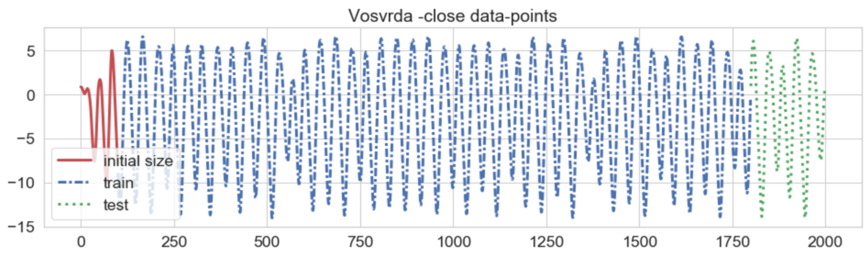
$$\left.\begin{aligned}\dot{S}(t) &= \alpha Y * (t) + p * S(t) * (k - Y(t) * Y(t)) \\ \dot{Y}(t) &= u * (S(t) + F(t)) \\ \dot{F}(t) &= m * S(t) - r * Y(t)\end{aligned}\right\} \text{ the Vosvrda system} \qquad (7)$$

where dotted quantities denote first time derivatives.

In its macroeconomic interpretation, $S(t)$ stands for the household savings, $Y(t)$ is the GDP and the $F(t)$ represents the foreign capital inflow. Regarding the micro-economic version of the model, $Y(t)$ are a firm's earnings, $S(t)$ are the reinvested capital and $F(t)$ the firm's debt. Regarding the parameters in the case of the macroeconomic version, α is the variation of the marginal propensity to savings, p is the ratio of the capitalised profit, u is the output/capital ratio, k is the potential GDP (which here is set to 1, as a unit of GDP – Y, S, F then represent the percentage part of the potential GDP), m is the capital inflow/savings ratio and r is the debt refund/output ratio.

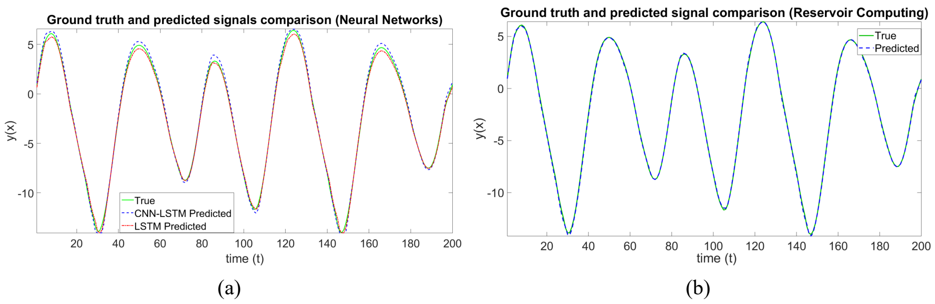
For this work, the aforementioned parameters were set to have the following values: α = 0.1, p = 0.01, u = 0.032, m = 0.2, r = 0.25. Parameter values are per unit time. Arbitrary time units were used. Note that the only physical explanation for the selection of these values was that these were ensuring that the system was demonstrating a deterministic chaotic behaviour.

Figure 7 The original Vosvrda chaotic time series (see online version for colours)



Notes: The parts used for the initialisation, training and testing procedures of the RNN and RC models are clearly shown.

Figure 8 The Vosvrda system test data points (green line) plotted versus the predicted values, (a) of the optimised LSTM RNN model (blue dashed line) and the CNN-LSTM RNN (red dotted line) (b) the RC (ESN) model (blue dashed line) (see online version for colours)



Like all deterministic chaotic systems, the Vosvrda system displays all the accompanying characteristic features, like high sensitivity to initial conditions and parameter values, thus resulting into tremendously different outcomes for almost identical inputs. As a result, predicting its temporal evolution by a single variable of the system, is an issue. Although it is a low-dimensional dynamical system, a fact that enables forecasting with more conventional methods of nonlinear analysis, still presents an interesting first approach to compare forecasting capabilities of the neural network models (LSTM and CNN-LSTM) against RC.

Therefore, a chaotic time series solution of the Vosvrda system, using a Runge-Kutta (4, 5) method, was calculated, and is plotted in Figure 7. In this graph, the length of the series used for the initialisation of the neural and RC networks was set to 100 data points, the training set consisted of the next 1,400 data points and finally, the last 200 points served as the testing set.

In the plots appearing in Figure 8(a), the comparison between the Vosvrda system time series testing area of the 200 points (green line), with the predicted signal of the LSTM RNN (blue dashed line) and the CNN-LSTM RNN (red dotted line) models, are illustrated. Likewise, in the plots appearing in Figure 8(b), the same comparison between the original (green line) and predicted by the ESN-RC (dashed blue line), is displayed. There is no doubt that in all three cases of deep learning models, these operate properly and predict high quality results, as the prediction lines follow the true time series behaviour without any forward shift. It is apparent from Figure 8(b) that especially in the case of the ESN model (RC), deviations from the true signal are minor and hardly distinguishable.

In Table 1 a comparison of the forecasting performance between the ESN and the two neural network models is presented. Towards this, the utilised metrics were the values of the root mean square error (RMSE), the mean absolute error (MAE), the absolute number and the percentage of the correctly predicted trends, and the training times. Apparently, predicting a *trend* means to predict whether the next series point will move to a higher or lower value compared to the present value.

Table 1 Comparative performance results of the LSTM, CNN-LSTM RNN and ESN RC in the case of the Vosvrda chaotic system

Model	*RMSE*	*MAE*	*Training time*	*Trends*	*Trends %*
LSTM	0.171	0.158	3.5 min.	193/199	96.9
CNN-LSTM	0.31	0.22	5 min.	191/199	95.9
ESN	0.1056	0.0781	8 sec.	196/199	98.5

These metrics show that a high level of accuracy, in terms of next series point prediction, has been achieved by the testing models, with the ESN having clearly having the lead. Furthermore, a big advantage of the ESN compared to the LSTM and CNN-LSTM networks, is the significantly reduced time needed for the training procedure (approx. 8 sec. compared to 3.5 min. and 5 min., respectively). This is something expected, as the training process of the reservoir computing algorithms takes place just on the weights of the output layer, with reservoir weights being fixed. Moreover, in the RC there is no need for back propagation, thus the training takes place in just one step. Besides, the training process of a neural network is slow, as it requires small batch size to ensure stable outputs and a large number of epochs to be trained, which is computationally inefficient, especially when dealing with a large amount of data.

Regarding the comparison between the LSTM and CNN-LSTM networks, LSTM seems to be achieving slightly lower errors than the same model with the convolution layer. As already mentioned, the basic concept behind the idea of the CNN-LSTM is that the convolutional mathematical operations can detect and extract specific complex patterns of the time series that an LSTM network alone could not. However, in the Vosvrda case things are different due to the lack of unusual complex patterns. Thus, the CNN-LSTM extra layers may be just induce complexity to the network that finally prevents it from generalising in unseen data better than an LSTM stand-alone network. The training time is also less in the LSTM case and this can be easily explained by the fact that the convolutional layer and especially the max pooling layer of the CNN-LSTM induce an extra computational effort in terms of multiple linear algebra operations, to flatten the signal in order to be fed to LSTM layers of the network.

Finally, all three models achieve a very high percentage of correctly predicted trends. The ESN achieved 196 out of 199, thus a 98.5% of correctly predicted trends; while LSTM RNN succeeded in 193 out of 199 trends, reaching a percentage of 97%. In the last place CNN-LSTM network predicted 191 correct trends. It is noted that successful time series trend prediction is a key metric for noisy financial time series prediction, especially for the stock market, which is the subject of the next subsection, and it was the reason why this metric was also included in the prediction of chaotic system time series.

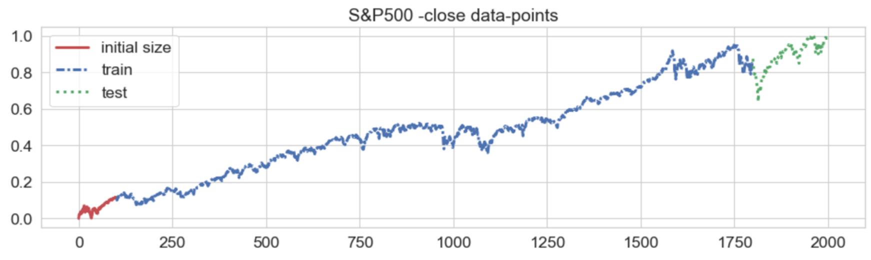
4.2 The S&P 500 stock market index

It is obvious that stock prices can be affected (indicatively) by political decisions, international level changes of policies, migration, physical phenomena and a plethora of other factors, hard to foresee or take into account. Apparently, all these factors combined, make up a financial complex system corresponding to a nonlinear dynamical system, whose behaviour is governed by chaotic dynamics. Therefore, financial data is essentially noisy data, as it comes from a highly-dimensional dynamical systems, which although clearly containing short-term and long-term temporal correlations, their evolution appears almost random and difficult to predict (Black, 1986).

It has been already mentioned above that one of the aims of this work is to underline and delve into the performance of RC over non-stationary data series. In an intuitive sense, stationarity implies that the statistical properties of a series should remain constant over time; thus, it ensures that statistical properties like the mean, variance, and correlations accurately describe the data over the whole extent of the time series. Consequently, for properly studying and analysing any time series (linear or nonlinear), this should not exhibit any trends (upward or downward) or seasonal effects; at the same time, mean, variance or co-variance should be constant over time.

Although non-stationary time series render analysis, evaluation and predictive modelling, it is the case usually encountered in financial systems. So in this subsection, RC (specifically the ESN algorithm) is compared to an LSTM RNN and a CNN-LSTM network, in the complete absence of any popular stationarity techniques, except for a mere normalisation procedure, as applied in the previous section for the Vosvrda model. The general methodology has already been described; thus, prediction of 199 data points of the stock market index S&P 500 would be attempted.

Figure 9 The S&P-500 time series used for the compared models, with the initialisation, training and testing parts clearly appearing (see online version for colours)

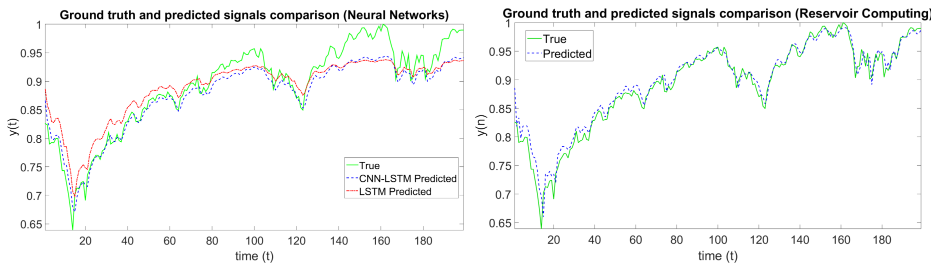


In Figure 9 the normalised within the [0, 1] domain, data series of the closing value for S&P 500, is plotted. This graph regards a period of nine years, from 12 December 2010 to 12 December 2019. As in the previous case, the time series is broken into an initialisation (100 data points – red part of the plot), a training (1,400 data points – the blue part of the plot) and a testing part (200 points – the green part of the plot). The feature of non-stationarity is visually apparent, since the examined time series demonstrates a clear trend.

The training of all models on the S&P 500 series followed a multivariate approach, where for each day t, the opening price, the day's lowest and highest values and closing price, together with the opening price of day $t+1$, created a kind of vector, which served as the input to the model, making up the multivariate training dataset. The prediction target was the closing price of day $t+1$.

These input variables are enough to establish a multivariate feed for our models as our aim is to compare them for a fixed number of certain inputs and not combine inputs from different time series to get the best of our algorithms. A case where different stock marketing indexes combined provide several inputs to our model could also be a future work plan.

Figure 10 (a) The S&P-500 test data points plotted versus the predicted values of the optimised LSTM and the CNN-LSTM model (b) The S&P-500 test data points plotted versus the predicted values of the optimised RC-ESN model (see online version for colours)



(a) (b)

Besides, the aforementioned changes on the way data is fed into the networks at each step, it was necessary to adjust some specifications of the models. In specific, in all networks, trials showed that the number of neurons for the LSTM/CNN-LSTM and the reservoir size for the ESN, had to be increased. This adjustment was judged to be necessary, since S&P 500 represents a significantly higher-dimensional dynamical system. Equivalently, it requires a higher-dimensional network, in order to properly model it. Therefore, the neurons of each layer were increased to 100 for the neural networks, while the reservoir size was set to 500, for the ESN. Additionally, regarding the LSTM RNN and the CNN-LSTM, there was no need for a sliding window this time, since the input data of the training set was now multivariate, therefore no further need to over-sample the input with more sequences, existed.

Table 2 Comparative performance results of the LSTM, the CNN-LSTM RNN and the ESN RC for the real world data of the S&P-500 index

Model	*RMSE*	*MAE*	*Training time*	*Trends*	*Trends %*
LSTM	0.03	0.0249	6 min.	124/198	62.6
CNN-LSTM	0.025	0.02	10 min.	126/198	64.1
ESN	0.0117	0.0085	10 sec.	136/198	68.7

It has to be mentioned that the main cause behind the multivariate feeding approach was our intention to create a prediction realistic framework. Since the values of the previous day, combined with the opening value of the current day are utilised as the input to our models, a window of several hours is provided to a potential stock market analyst, which is enough to consult with his clients on price changes and act accordingly.

Prediction results regarding the three models, in the case of the S&P-500 data series, appear in Figure 10. In these graphs the testing area of 199 points appear and regard the closing value prediction for the S&P-500. Contrary to the previous case, this time the differences between the prediction outputs of the models are even visually apparent. The ESN model clearly outperforms the neural networks.

All the relevant results are comparatively presented in Table 2, together with the required training time for the three models. Performance differences between the neural network models and the RC model are now much more evident, rendering RC as a much more effective network for predicting highly-complex and noisy financial time series. The ESN achieved a 68.6% of correctly predicted trends; while LSTM and CNN-LSTM succeeded in 62.6% and 64.1% of trends, respectively. Again the RMSE and the MAE are very improved in the case of the RC. Furthermore, it becomes obvious that in the case of RC (ESN), solving the over-determined system in equation (4) results into a massive reduction of training time. Compared to neural network models, that exhibit back propagation, minimising cost function processes and re-adjusting weight actions, a significant increase in necessary computational resources is evident.

In addition to this, what is also interesting is the comparison between the LSTM and CNN-LSTM models. Table 2 shows that concerning the error metrics used, the two models are quite similar to each other regarding accuracy, with CNN-LSTM having the lead. This lead is also illustrated in Figure 10(a), where the blue dashed line representing the CNN-LSTM predictions are more close to the spikes of dips and sudden increases of the true (green line) data of S&P-500. This is interesting and can be explained by the fact that the CNN's convolution and pooling layers duty, was to strengthen the model in terms of pattern recognition processes before the data is fed into the following LSTM layers; something that can be assumed seen in the figure. Note that this difference is absent in the Vosvrda time series predictions, as the behaviour of studied time series presents no fluctuations; thus both the LSTM and CNN-LSTM prediction resembles in a clear and similar way.

5 Discussion and concluding remarks

The scope of this paper was to delve into nonlinear time series prediction with focus on the financial sector. By developing a multi-variate scheme suitable for point to point or day to day prediction of prices and trends, a comparison between neural networks (LSTM and CNN-LSTM RNN) and RC (ESN) was attempted.

To this direction a chaotic macroeconomic model, namely the Vosvrda equation system, was initially utilised. This served in testing our algorithms and optimising their parameters, while maximising the output results to the greatest possible extent. The result of this stage led to a better understanding of the LSTM, the CNN-LSTM and the ESN dynamics and capabilities. In this case, all models proved to be successful in terms of error metrics, achieving low root mean square and absolute errors, along with high percentage of correct predicted trends. The vast reduction in training time appears to be the greatest asset for RC (ESN), compared to the two neural network models. The ESN was able to produce slightly better results than the other models, but within only a very small fraction of the training time (8 sec. vs. 3.5 min. for the LSTM and 5 min. for the CNN-LSTM).

Regarding the main challenge of this work, that of forecasting the values of real financial data, we opted to compare the two deep learning models in the case of a classic stock market index, namely the S&P-500. The designed strategy towards the implementation of the neural networks and the RC algorithms was based on the capability of the later to perform well even with non-stationary inputs. In specific, the ESN maps the inputs to a high-dimensional state space; thus being able to capture the dynamical behaviour of the time series more efficiently, as this higher dimension mapping contributes to a nonlinear decay. As a result, common stationarity techniques are completely absent in this paper.

Moreover, a multivariate scheme, in terms of stock values and time, was also employed and aimed in the first hand to establish a more stable and robust system. On the other hand this approach provided with a framework where a potential individual would have the ability to access/predict information for the closing value of S&P-500, after the opening value is announced.

The emerging comparison results showed that mean absolute and RMSEs were quite lower in the case of RC (ESN), while the correctly predicted trends in the case of RC reached up to 68.6% compared to a 64.1% of the CNN-LSTM and a 62.6% of the LSTM model. It should be mentioned that these lower percentages, compared to Vosvrda's results, are well expected, since S&P-500 is a real stock market index governed by fluctuations and noise; thus it is quite hard for any model to extract time patterns.

Concluding in brief, RC ESN model clearly outperforms the recurrent neural network LSTM model and the convolution boosted version of LSTM – the CNN-LSTM – both in the case of a chaotic time series (coming from a typical chaotic set of equations) and a real world stock market index. Next to better trend and absolute value predictions, training time of RC is incomparably smaller than that of neural networks (about 80 times less), which further results in significantly less power consumption due to decreased CPU usage and computational efficiency. Finally, since all these results emerged in the case of no pre-processing of the data, it is demonstrated that stationarity techniques are not needed for apt forecasting in the case of RC.

Finally, regarding future work, it would be worthwhile to use our RC model and the gained knowledge on reservoir systems as well as their supremacy to target other variables or even other computed financial metrics (such as the sharpe ratio) of a portfolio of stock marketing indexes.

References

Alghalith, M. and Floros, C. (2020) 'Futures hedging with stochastic volatility: a new method', *International Journal of Computational Economics and Econometrics*, Vol. 10, No. 2, pp.203–207.

Antoniades, I., Stavrinides, S., Hanias, M. and Magafas, L. (2020) 'Complex network time series analysis of a macroeconomic model', in *Chaos and Complex Systems*, pp.135–147, Springer, Switzerland.

Balli, F., Pierucci, E. and Gan, J. (2020) 'Determinants of risk sharing via exports: trade openness and specialisation', *International Journal of Computational Economics and Econometrics*, Vol. 10, No. 4, pp.380–397.

Black, F. (1986) 'Noise', *The Journal of Finance*, Vol. 41, No. 3, pp.528–543.

Bouali, S. (1999) 'Feedback loop in extended Van der Pol's equation applied to an economic model of cycles', *International Journal of Bifurcation and Chaos*, Vol. 9, No. 4, pp.745–756.

Buali, S. (2002) 'The hunt hypothesis and the dividend policy of the firm. The chaotic motion of the profits', in *8th International Conference of the Society for Computational Economics*, June, pp.1–12.

Chan, K.H., Hayya, J.C. and Ord, J.K. (1977) 'A note on trend removal methods: the case of polynomial regression versus variate differencing', *Econometrica*, Vol. 45, No. 3, ppp.737–744 [online] http://www.jstor.org/stable/1911686.

Chen, K., Zhou, Y. and 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)*, IEEE, pp.2823–2824.

Cuchiero, C., Gonon, L., Grigoryeva, L., Ortega, J-P. and Teichmann, J. (2020) 'Discrete-time signatures and randomness in reservoir computing', *IEEE Transactions on Neural Networks and Learning Systems*, DOI: 10.1109/TNNLS.2021.3076777.

Dunis, C.L. and Huang, X. (2002) 'Forecasting and trading currency volatility: an application of recurrent neural regression and model combination', *Journal of Forecasting*, Vol. 21, No. 5, pp.317–354.

Fu, K., Cheng, D., Tu, Y. and Zhang, L. (2016) 'Credit card fraud detection using convolutional neural networks', in Hirose, A., Ozawa, S., Doya, K., Ikeda, K., Lee, M. and Liu, D. (Eds.): *Neural Information Processing*, pp.483–490, Springer International Publishing, Cham.

Han, D., Liu, Q. and Fan, W. (2018) 'A new image classification method using CNN transfer learning and web data augmentation', *Expert Systems with Applications*, Vol. 95, pp.43–56 [online] http://www.sciencedirect.com/science/article/pii/S0957417417307844.

Hanias, M., Magafas, L. and Stavrinides, S.G. (2019) ''Reverse engineering' in econophysics', *International Journal of Productivity Management and Assessment Technologies*, Vol. 7, No. 1, pp.36–49.

Heaton, J.B., Polson, N.G. and Witte, J.H. (2016) *Deep Learning in Finance*, arXiv preprint arXiv:1602.06561.

Heryadi, Y. and Warnars, H.L.H.S. (2017) 'Learning temporal representation of transaction amount for fraudulent transaction recognition using CNN, stacked LSTM, and CNN-LSTM', in *2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom)*, pp.84–89.

Hochreiter, S. and Schmidhuber, J. (1997) 'Long short-term memory', *Neural Comput.*, Vol. 9, No. 8, pp.1735–1780 [online] https://doi.org/10.1162/neco.1997.9.8.1735.

Hori, T., Watanabe, S., Zhang, Y. and Chan, W. (2017) *Advances in Joint CTC-Attention based End-to-End Speech Recognition with a Deep CNN Encoder and RNN-LM*, arXiv preprint arXiv:1706.02737.

Jaeger, H. (2001) *The 'Echo State' Approach to Analysing and Training Recurrent Neural Networks – With An Erratum Note*, German National Research Center for Information Technology GMD Technical Report, No. 148, Bonn, Germany.

Jaeger, H. (2002) *Tutorial on Training Recurrent Neural Networks, Covering BPPT, RTRL, EKF and the 'Echo State Network' Approach*, Vol. 5, GMD-Forschungszentrum Informationstechnik Bonn.

Jaeger, H. (2012) *Long Short-Term Memory in Echo State Networks: Details of a Simulation Study*, Jacobs University Bremen.

Jaeger, H. and Haas, H. (2004) 'Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication', *Science*, Vol. 304, No. 5667, pp.78–80.

Karim, F., Majumdar, S., Darabi, H. and Chen, S. (2018) 'LSTM fully convolutional networks for time series classification', *IEEE Access*, Vol. 6, pp.1662–1669.

Köhler, T. and Lorenz, D. (2005) *A Comparison of Denoising Methods for One Dimensional Time Series*, Technical Report DFG SPP 1114, University of Bremen, Bremen, Germany.

Livieris, I., Pintelas, E. and Pintelas, P. (2020) 'A CNN-LSTM model for gold price time series forecasting', *Neural Computing and Applications*, Vol. 32, No. 23, pp.17351–17360.

Lukoševičius, M. and Jaeger, H. (2009) 'Reservoir computing approaches to recurrent neural network training', *Computer Science Review*, Vol. 3, No. 3, pp.127–149.

Lukoševičius, M. (2012) *A Practical Guide to Applying Echo State Networks*, pp.659–686, Springer, Berlin, Heidelberg.

Maknickienė, N., Rutkauskas, A.V. and Maknickas, A. (2011) 'Investigation of financial market prediction by recurrent neural network', *Innovative Technologies for Science, Business and Education*, Vol. 2, No. 11, pp.3–8.

Mao, J., Xu, W., Yang, Y., Wang, J., Huang, Z. and Yuille, A. (2014) *Deep Captioning with Multimodal Recurrent Neural Networks (M-RNN)*, arXiv preprint arXiv:1412.6632.

Mejdoub, H. and Arab, M.B. (2019) 'Insurance risk capital and risk aggregation: bivariate copula approach', *International Journal of Computational Economics and Econometrics*, Vol. 9, No. 3, pp.202–218.

Nelson, D.M., Pereira, A.C. and de Oliveira, R.A. (2017) 'Stock market's price movement prediction with LSTM neural networks', in *2017 International Joint Conference on Neural Networks*, IEEE, pp.1419–1426.

Olah, C. (2015) *Understanding LSTM Networks* [online] https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed 10 March 2021).

Pribylova, L. (2009) 'Bifurcation routes to chaos in an extended Van der Pol's equation applied to economic models', *Electronic Journal of Differential Equations [Electronic Only]*, Vol. 2009, No. 53, pp.1–21.

Qi, M. and Zhang, G.P. (2008) 'Trend time-series modeling and forecasting with neural networks', *IEEE Transactions on Neural Networks*, Vol. 19, No. 5, pp.808–816.

Qian, F. and Chen, X. (2019) 'Stock prediction based on LSTM under different stability', in *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, IEEE, pp.483–486.

Rao, K., Sak, H. and Prabhavalkar, R. (2017) 'Exploring architectures, data and units for streaming end-to-end speech recognition with rnn-transducer', in *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp.193–199.

Roy, A., Sun, J., Mahoney, R., Alonzi, L., Adams, S. and Beling, P. (2018) 'Deep learning detecting fraud in credit card transactions', in *2018 Systems and Information Engineering Design Symposium (SIEDS)*, IEEE, pp.129–134.

Sherstinsky, A. (2020) 'Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network', *Physica D: Nonlinear Phenomena*, Vol. 404, p.132306.

Tsiptsis, K.K. and Chorianopoulos, A. (2011) *Data Mining Techniques in CRM: Inside Customer Segmentation*, John Wiley & Sons, West Sussex, UK.

Vaisla, K.S. and Bhatt, A.K. (2010) 'An analysis of the performance of artificial neural network technique for stock market forecasting', *International Journal on Computer Science and Engineering*, Vol. 2, No. 6, pp.2104–2109.

Verstraeten, D., Schrauwen, B., d'Haene, M. and Stroobandt, D. (2007) 'An experimental unification of reservoir computing methods', *Neural Networks*, Vol. 20, No. 3, pp.391–403.

Vosvrda, M. (2001) 'Bifurcation routes and economic stability', *Bulletin of the Czech Econometric Society*, Vol. 14, pp.43–60.

Vosvrda, M. et al. (2001) 'Bifurcation routes and economic stability', in *7th International Conference of the Society for Computational Economics*, Yale University, June, pp.28–30.

Wang, C. (2015) *Convolutional Neural Network for Image Classification*, Johns Hopkins University Baltimore, MD, 21218, USA.

Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C. and Xu, W. (2016) 'CNN-RNN: a unified framework for multi-label image classification', in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhang, M., Li, W. and Du, Q. (2018) 'Diverse region-based CNN for hyperspectral image classification', *IEEE Transactions on Image Processing*, Vol. 27, No. 6, pp.2623–2634.

Zheng, L., Yang, Y. and Tian, Q. (2018) 'Sift meets CNN: a decade survey of instance retrieval', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 40, No. 5, pp.1224–1244.