Chaos, Solitons and Fractals 139 (2020) 110045

Contents lists available at ScienceDirect

Chaos, Solitons and Fractals

Nonlinear Science, and Nonequilibrium and Complex Phenomena

journal homepage: www.elsevier.com/locate/chaos

Frontiers

Robustness of LSTM neural networks for multi-step forecasting of chaotic time series

Matteo Sangiorgio*, Fabio Dercole

Department of Electronics, Information and Bioengineering, Politecnico di Milano, Via Ponzio 34/5, I-20133, Milano, Italy

ARTICLE INFO

Article history:
Received 19 December 2019
Revised 23 May 2020
Accepted 21 June 2020
Available online 30 June 2020

Keywords:
Deterministic chaos
Recurrent neural networks
Teacher forcing
Exposure bias
Multi-step prediction
Nonlinear time series

ABSTRACT

Recurrent neurons (and in particular LSTM cells) demonstrated to be efficient when used as basic blocks to build sequence to sequence architectures, which represent the state-of-the-art approach in many sequential tasks related to natural language processing. In this work, these architectures are proposed as general purposes, multi-step predictors for nonlinear time series. We analyze artificial, noise-free data generated by chaotic oscillators and compare LSTM nets with the benchmarks set by feed-forward, one-step-recursive and multi-output predictors. We focus on two different training methods for LSTM nets. The traditional one makes use of the so-called teacher forcing, i.e. the ground truth data are used as input for each time step ahead, rather than the outputs predicted for the previous steps. Conversely, the second feeds the previous predictions back into the recurrent neurons, as it happens when the network is used in forecasting. LSTM predictors robustly show the strengths of the two benchmark competitors, i.e., the good short-term performance of one-step-recursive predictors and greatly improved mid-long-term predictions with respect to feed-forward, multi-output predictors. Training LSTM predictors without teacher forcing is recommended to improve accuracy and robustness, and ensures a more uniform distribution of the predictive power within the chaotic attractor. We also show that LSTM architectures maintain good performances when the number of time lags included in the input differs from the actual embedding dimension of the dataset, a feature that is very important when working on real data.

1. Introduction

In the last few decades, many attempts to forecast the future evolution of chaotic systems, and to discover how far they can be predicted, have been carried out adopting a wide range of models. Some early attempts were performed in the 80s and 90s [1–5], but the topic became more and more debated in recent years, due to the development of lots of machine learning techniques in the field of time series analysis and prediction. Attempts include famous models such as the support vector machines [6], but the most widely used architectures are artificial neural networks (ANNs). The main distinction within ANNs is between feed-forward (FF) structures—static maps which approximate the relationship between inputs and outputs—and recurrent neural nets (RNNs)—dynamical models whose outputs also depend on a hidden state characterizing each neuron.

Examples of the first category are the traditional multi-layer perceptrons [7–14], radial basis function networks [15–20], fuzzy neural networks [21–25], deep belief neural nets [26], and convolutional neural networks [27]. RNNs are particularly suited to be used as predictors of nonlinear time series [7,28–35], though their training is computationally heavy because of the back propagation through time of the internal memory [36]. Recently, two particular RNN classes, reservoir computers and long short-term memory (LSTM) networks, have been demonstrated to be extremely efficient in the prediction of chaotic dynamics [37–56]. Reservoir computers make use of many recurrent neurons with random connections and training is only performed on the output layer. Conversely, recurrent neurons are trained in LSTM nets. This avoids the typical issue of vanishing or exploding optimization gradients by modulating the information flow through so-called gates, whose behaviors are learned as well.

The comparison between feed-forward and recurrent nets is not trivial. Despite the initial optimism, RNNs turned out to achieve similar performances as traditional FF architectures on many forecasting tasks [34,57]. Recurrent neurons have been demonstrated to be efficient when they are used as basic blocks to build up sequence to sequence architectures. This kind of structure represents the state of the art approach in many natural language processing tasks (e.g., machine translation, question answering,

* Corresponding author.
E-mail address: matteo.sangiorgio@polimi.it (M. Sangiorgio).

https://doi.org/10.1016/j.chaos.2020.110045
0960-0779/

automatic summarization, text-to-speech,...), though they are recently losing ground to advanced FF architectures. Other works in the field of chaotic dynamics forecasting focused on wavelet networks [58], swarm optimized neural nets [59], Chebyshev [60], and residual [34] nets. Some authors developed hybrid forecasting schemes that integrate machine learning tools and knowledge-based models [44,61–63] or combine multiple predictive models [64,65].

Forecasting chaotic dynamics one or few time steps ahead is usually an easy task as demonstrated by the high performances obtained on many systems, in both continuous and discrete time [10,12,21,28,46,66,67]. The situation dramatically changes when considering a mid-long horizon, because of the expansion of small errors intrinsically due to the chaotic nature of the ground truth data. Even when the data are generated by a known chaotic system, small initialization or numerical errors are amplified up to predict a random point in the system's attractor. The forecasting of a chaotic time series over a multi-step horizon is commonly done by recursively performing one-step ahead predictions [60,68–70]. An alternative consists of training the model to directly compute multiple outputs [19], each of which represents the prediction at a certain time step, or even identifying a specific model for each future step [59].

In this work, we compare three state-of-the-art architectures for the multi-step prediction of noise-free, chaotic time series: FF-recursive, FF-multi-output, and LSTM nets, the latter trained according to two different methods: the so-called teacher forcing (TF), so far traditionally used, and the variant without teacher forcing (no-TF). We analyze strengths an weaknesses of these four predictors, from both the qualitative and quantitative perspectives.

Because optimized to predict only the next time step, the FF-recursive net is trained to replicate the dynamical systems generating the data, i.e., the map from previous data to the next one. Its performance however degrades on mid-long horizons because of the system's chaoticity. The FF-multi-output net is trained to predict the whole forecasting horizon, but it acts as if the outputs were independent variables, rather than the same variable sampled at subsequent steps. For this reason, it struggles to replicate the dynamical behavior of the chaotic maps.

To overcome the critical aspects of FF nets, we adopt models which are able to deal with temporal dynamics, such as the LSTM nets. These architectures efficiently couple the advantages of the FF-recursive and multi-output nets. However, they are usually trained with the so-called TF algorithm, in which the ground truth data are used as input for each time step ahead, rather than the outputs predicted for the previous steps. Essentially, the net learns to predict only one step, because even when predicting further, it receives the actual data up to the step before the one to be predicted. In other words, the TF prevents small errors in the approximated map being corrected, since they are not propagated in time. In inference mode, multi-step predictions necessarily require previous prediction to be fed back, so that the network behaviors in training and prediction do not coincide (this issue is known as exposure bias in the machine learning literature [71]). TF is critical especially when dealing with chaotic dynamics for their well known sensitivity to initialization and prediction errors.

To solve this issue, we propose to abandon the TF method, feeding back the network predictions even during training. We therefore compare four neural predictors: FF-recursive, FF-multi-output, LSTM-TF, and LSTM-no-TF. The obtained results show that LSTM nets trained without TF are the most performing in predicting chaotic dynamics. They are also more robust, with respect to the three competitors, when a redundant (or lacking) number of time lags are included in the input—a feature that makes them suitable to work with real data.

2. Methods

The problem of forecasting h steps ahead (leads) of a time series $y(t)$ consists of identifying a predictor that takes as input the d last samples (lags), $y(t-d+1), \ldots, y(t-1), y(t)$, and returns as output the predictions $\hat{y}(t+1), \hat{y}(t+2), \ldots, \hat{y}(t+h)$ of the corresponding values $y(t+1), y(t+2), \ldots, y(t+h)$.

The number d of lags defines how many autoregressive terms the model takes as input. In order to make the learning problem feasible, this number must be large enough to establish a one-to-one relation between the input and the one-step ahead prediction $\hat{y}(t+1)$. In nonlinear time series analysis a feasible d is said to allow the embedding of the dataset and the minimum d is called the dataset embedding dimension. When the time series is generated by a known (autonomous, time-invariant) chaotic system, the embedding theory says that the smallest integer larger than twice the fractal dimension of the system's attractor generically allows the embedding [72]. Moreover, if the system is formulated as a d-dimensional nonlinear regression of the output variable $y(t)$, i.e., $y(t+1)$ is expressed as a nonlinear function of the lags $y(t-d+1), \ldots, y(t-1), y(t)$, then d is the embedding dimension. This is the case for the chaotic oscillators on which we test our predictors. For real-world datasets, the embedding dimension can be numerically estimated [73]. In any case, too large values of d can lead to a worse learning and performance of the predictor, as further discussed in Section 3.2.

It is important to underline that, in principle, both FF nets and RNNs are able to solve the forecasting problem defined above. The difference is that FF predictors see the problem as a static one (they try to reproduce the relationship between the input and output sets), while RNNs look at the same problem under a dynamical perspective. In both cases, it is necessary to reorganize the time series, $y(1), y(2), \ldots, y(t), \ldots,$ in N pairs of input $\left[y(t-d+1), \ldots, y(t-1), y(t)\right]$, and output $\left[y(t+1), y(t+2), \ldots, y(t+h)\right]$ vectors, called input-output samples in the machine learning terminology. The predictor is then a function from the d-dimensional input space to the h-dimensional output space.

The three neural architectures we focus on, FF-recursive and FF-multi-output used as benchmark and LSTM, are reported in Fig. 1 and will be described in details in the following subsections. The rest of this section concerns the chaotic oscillators on which we test the predictors, the metrics adopted for their performance evaluation, and the details of the training procedure.

2.1. FF-recursive predictor

The most common and natural approach consists of identifying the best single-step ahead predictor and then use it in a recursive way, feeding the previous step prediction back into the input vector of the following step (see Fig. 1a). Note that, despite the dynamic nature of the time series, the identification of a FF-recursive predictor is a static task. It basically requires to mimic the mapping from d inputs to a single output.

The main advantage of this approach is that once the one-step predictor has been trained, it can be used recursively to forecast an arbitrarily long sequence. This is, however, its main drawback: it is not optimized for the task we are dealing with, that is a multi-step prediction.

2.2. FF-multi-output predictor

In order to optimize the prediction on the entire sequence, it is necessary to define a model with multiple outputs. A first attempt can be done adopting a feed-forward and fully connected architecture (Fig. 1b). This structure is the same one already used for the single-step predictor. The only difference is in the output layer,

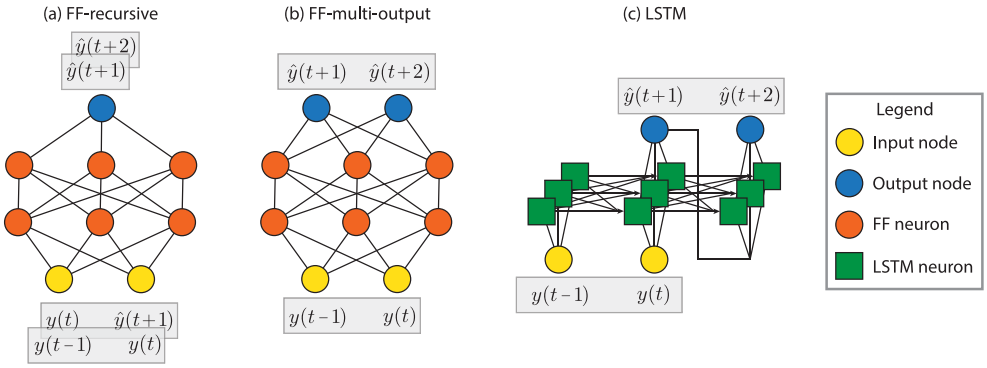


Fig. 1. Neural architectures (shown for $d = 2$ and $h = 2$, for simplicity). For illustrative purposes, we represent 2 hidden layers of 3 neurons each for the FF architectures and a single layer with 3 nodes for the LSTM.

whose number of nodes increases from 1 to h. Each neuron in the output layer focuses on the prediction of the considered variable at a different time step. The main issue with this architecture is that it does not take into account that the outputs are sequential (i.e., the same variable at different time steps). In fact, the model would act in the same way if the outputs were to predict different system's variables at the same time step.

2.3. LSTM predictor

The third ANN we consider is rather similar to the sequence-to-sequence architecture that obtained in the last decade outstanding performances in natural language processing [74]. The idea, schematically represented in Fig. 1c, is to set up a structure capable of explicitly take into account the sequentiality of the time series. To do that, the nodes in the hidden layers are not traditional feed-forward neurons, but recurrent ones, in particular LSTM cells. Each LSTM cell has two internal states (named hidden and cell state, respectively) and three gates (input, output, and forget gates) [36,75]. Intuitively, the cell state is responsible for keeping track of the relevant information provided by past inputs, the hidden state synthesizes the information provided by the current input, the cell state, and the previous hidden state. The gates regulate the flow of information in and out of the neuron. Specifically, the input and forget gates control the extent to which a new input value and the current cell state respectively affect the new cell state; the output gate controls the extent to which the internal states affect the output.

The distinctive feature of RNNs is that the weights of the neurons that process different time steps are shared. For this reason, the number of parameters does not change if one considers more (or less) autoregressive terms in the past (lags d), or a different number of predicted steps ahead (leads h). In principle, this should be an advantage when d and/or h are large. At the same time, the weights sharing across different layers complexifies the optimization problem and hence requires higher computational effort. This architecture potentially overcomes the limitations of the FF-recursive and multi-output predictors, because it is trained to reproduce the entire sequence of output variables and it explicitly takes into account that these h outputs are sequential values of the same variable.

The RNNs are almost always trained using a technique known as teacher forcing (TF) [76]. It consists of using the ground truth data as input for each time step ahead, rather than the output predicted by the network at the previous step, as shown in Fig. 2a. This technique proved to be efficient in natural language processing related tasks, granting faster convergence and avoiding the accumulation of errors in the initial phase of training [77]. For these reasons, all the high-level API for deep learning adopt the TF by default when training RNNs. To implement a training without TF (see Fig. 2b), it is necessary to code a non-standard implementation directly in a low-level API, such as TensorFlow or PyTorch.

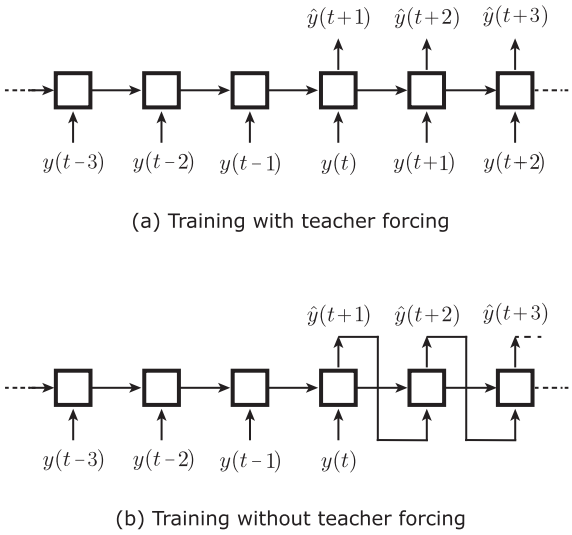


Fig. 2. LSTM nets training procedure with (a) and without (b) teacher forcing.

In inference mode, when forecasting multiple time steps, the unknown previous value is replaced by the model prediction. TF therefore introduces a discrepancy between training and inference (exposure bias) that yields errors that accumulate along the sequence of predictions [77]. Basically, training with TF does not allow the network to correct its own mistakes because, during the training phase, the prediction at a certain time step does not affect future predictions, as it does during inference. In forecasting tasks, this leads to a situation that is somewhat similar to use a one-step, recursive predictor.

We thus propose to train the LSTM architecture without TF (LSTM-no-TF). Coupling these two elements, recurrent neurons and no-TF, solves at the same time the drawbacks of the FF-recursive and multi-output predictors and of LSTM-TF. Indeed, this architecture is trained to predict the entire sequence of output variables and it explicitly takes into account the temporal connection between these outputs. Thanks to the no-TF method, its behaviors in training and inference modes coincide, so that the network can correct the small prediction errors that do propagate during training.

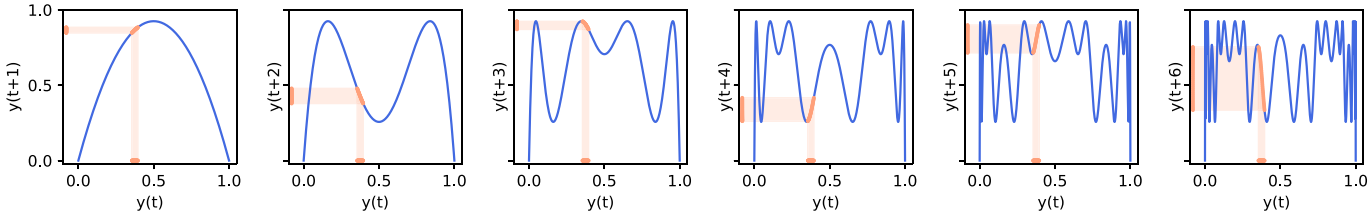


Fig. 3. Graph of the logistic map ($r = 3.7$, first panel on the left) and of the iterated maps 2-to-6. Orange areas show how a small error at time t propagates when the map is iterated.

2.4. Chaotic systems

The predictive power of the architectures presented in Section 2.1, 2.2 and 2.3 are tested on four discrete-time dynamical systems known to be chaotic: the logistic map, the Hènon map, and two generalized Hènon maps. The logistic and the Hènon maps are the classical prototypes of chaos in non-reversible and reversible discrete-time systems, respectively, whereas the generalized Hènon map is considered to include low and high-dimensional hyperchaos. We limit the analysis to a single output variable for each system so that the corresponding time series is univariate. For the ease of the reader, the systems are here presented.

The logistic map is a one-dimensional quadratic map used to describe biomass or economic growth. It has only one state variable, the density of the biological or economic resource that we take equal to the output, so that the systems can be defined by a single difference equation that directly describes the dynamics of the output, i.e.,

$$y(t+1) = r \cdot y(t) \cdot (1 - y(t)), \tag{1}$$

where $r > 1$ is the growth rate at low density. As it is well known, the logistic map exhibits a chaotic behavior for most of the values of r in the range (3.6, 4). The equation describing the map is a simple quadratic polynomial, which assumes the shape represented in the first panel of Fig. 3 for $r = 3.7$. Reproducing this function with a neural net is an easy task. Things start being more and more complex when this simple map is iterated (the other panels in Fig. 3). This is a general property of chaotic systems, that we illustrate for the logistic map. The effect of the increasing complexity that occurs when the map is iterated is that small errors in the input (on the horizontal axis in Fig. 3) give larger and larger errors in the output (vertical axis). Mid-long term prediction of chaotic systems is therefore intrinsically problematic, even if one disposes the truth model generating the data.

The Hènon map, a toy model of celestial mechanics, is a two-dimensional system traditionally defined in state space:

$$\begin{cases} x_1(t+1) = 1 - a \cdot x_1(t)^2 + x_2(t) \\ x_2(t+1) = b \cdot x_1(t), \end{cases} \tag{2}$$

where a and b are model parameters, usually taking values 1.4 and 0.3, respectively. Considering the first state variable as output, $y(t) = x_1(t)$, the system is equivalent to the following two-dimensional nonlinear regression:

$$y(t+1) = 1 - a \cdot y(t)^2 + b \cdot y(t-1). \tag{3}$$

The generalized Hènon map is an extension of the Hènon map introduced with the purpose of generating hyperchaos (a chaotic attractor characterized by at least two positive Lyapunov exponents, i.e., at least two directions of divergence within the attractor) [78,79]. The state equations of the d-dimensional case are:

$$\begin{cases} x_1(t+1) = a - x_{d-1}(t)^2 - b \cdot x_d(t) \\ x_j(t+1) = x_{j-1}(t), \quad j = 2, \ldots, d, \end{cases} \tag{4}$$

(for $d = 2$, using equation $x_1(t)/a$ and $(-b/a)x_2(t)$ as new coordinates and $-b$ as new parameter gives the traditional formulation (2)). An hyperchaotic behavior is observed for $a = 1.9$ and $b = 0.03$. The system can be rewritten as a nonlinear regression on the output, $y(t) = x_1(t)$, obtaining:

$$y(t+1) = a - y(t-d+2)^2 - b \cdot y(t-d+1). \tag{5}$$

We consider the 3D generalized Hènon ($d = 3$, the corresponding Lyapunov exponents are $L_1 = 0.276$, $L_2 = 0.257$, and $L_3 = -4.040$ and the attractor's fractal dimension, computed with the Kaplan-Yorke formula, is 2.13) and the 10D maps, with 9 positive Lyapunov exponents and attractor's fractal dimension equal to 9.13.

2.5. Performance metrics

The predictions obtained with the proposed ANNs have to be compared with the ground truth values that are computed by simulating the chaotic system. The comparison is done using appropriate metrics. A traditional metric for regression tasks is the mean squared error (MSE, hereafter). Focusing on the forecasting of the i^{th} step ahead, we can consider N ground truth samples $\mathbf{y} = \left[y_1, y_2, \ldots, y_N\right]$, not necessarily in temporal order, and the corresponding i-step ahead predictions $\hat{\mathbf{y}}^{(i)} = \left[\hat{y}_1^{(i)}, \hat{y}_2^{(i)}, \ldots, \hat{y}_N^{(i)}\right]$, i.e., $y_k = y(t_k + i)$ for some t_k in the dataset and $\hat{y}_k^{(i)}$ is the predictor output $\hat{y}(t_k + i)$ computed on the input $[y(t_k), y(t_k - 1), \ldots, y(t_k - d + 1)]$.

The MSE is then defined as:

$$\mathrm{MSE}\left(\mathbf{y}, \hat{\mathbf{y}}^{(i)}\right) = \frac{1}{N} \sum_{k=1}^{N} \left(y_k - \hat{y}_k^{(i)}\right)^2. \tag{6}$$

This is the metric typically used as loss function to be minimized during the network training. The main disadvantage of the MSE is that the assumed numerical values do not provide a good insight on the quality of the prediction, because they are not normalized with respect to the variability of the data. For this reason, to access the prediction quality, it is preferable to use a relative metrics, such as the R^2-score, sometimes known as Nash Sutcliffe model efficiency coefficient (not to be confused with the square of Pearson correlation coefficient). It is defined by the following expression:

$$R^2\left(\mathbf{y}, \hat{\mathbf{y}}^{(i)}\right) = 1 - \frac{\mathrm{MSE}\left(\mathbf{y}, \hat{\mathbf{y}}^{(i)}\right)}{\mathrm{MSE}(\mathbf{y}, \bar{y})} = 1 - \frac{\sum_{k=1}^{N} \left(y_k - \hat{y}_k^{(i)}\right)^2}{\sum_{k=1}^{N} (y_k - \bar{y})^2}, \tag{7}$$

where $\bar{y}$ is the mean of the ground truth data. The R^2-score measures the predictive power of a given model with respect to the predictive power of the trivial model which always forecasts the mean value of the observed data (in this case, $R^2 = 0$). This metric varies in the range $(-\infty, 1]$, the upper bound obtained in case of a perfect forecasting. The R^2-score is widely used because it can be seen as a normalized version of the MSE, being $\mathrm{MSE}(\mathbf{y}, \bar{y})$ the data variance.

The MSE and R^2-score, here defined for the prediction of the i^{th} step, can be averaged over the whole forecasting horizon of h

Table 1
Description of the neural network's hyper-parameter (so-called to avoid confusion with the network parameters, weights and biases, that define the input-output mapping.

Hyper-parameter	Description
Batch size	Number of training input-output samples used to compute the loss function gradient in one iteration of the optimization.
Learning rate	Step along the gradient by which the network's parameters are updated at each iteration of the optimization (also called step size).
L-rate decay	Exponential rate of decay of the learning rate during training.
Num. of epochs	Number of times the learning algorithm iterates through the entire training dataset, with sample reshuffling at each iteration.

steps, i.e.,

$$\langle \text{MSE} \rangle = \frac{1}{h}\sum_{i=1}^{h} \text{MSE}\left(\mathbf{y}, \hat{\mathbf{y}}^{(i)}\right)$$
$$\langle R^2 \rangle = \frac{1}{h}\sum_{i=1}^{h} R^2\left(\mathbf{y}, \hat{\mathbf{y}}^{(i)}\right). \quad (8)$$

Specifically for the prediction of chaotic systems, we introduce a third metric based on the system's largest Lyapunov exponent (LLE). It is defined as the number of Lyapunov times ahead for which the prediction error does not exceed a given threshold, where the Lyapunov time (LT) is the inverse of the LLE. This metric normalizes the prediction power of the neural model to the average chaoticity of the data generator. It allows to fairly compare the predictor's performance across different chaotic systems, though this is outside the scope of the present work.

2.6. Training procedure

Each system is used to generate 50,000 data points which has been split into training (70% of the samples), validation (15%), and test set (15%). The training set, as the name suggests, is used to train the parameters (i.e. weights and bias) of the neural network. The samples of the validation set are not directly processed by the optimization algorithm, but are used to tune the hyper-parameters of the neural network, which define the network structure and the learning algorithm (see Table 1). The test set does not have a role in the model identification, it is just used to evaluate the model's performance in inference mode.

In order to limit the number of hyper-parameters to be tuned on the one hand, and to provide a fair comparison between different architectures, we fixed the number of hidden layers (3 layers), and the number of neurons per layer (10 neurons). Other hyper-parameters (see Table 1) have been tuned through a grid search optimization. We tested a batch size of 256, 512, and 1024; learning rate equals to 0.1, 0.01, and 0.001; decay factor of 0.001, and 0. The number of epochs have been fixed to 5000, a value which ensures the convergence of each training. For each combination of the hyper-parameters, the optimization is repeated 2 times varying the random weight initialization in order to avoid dependence on particularly unlucky initial values. For the training procedure, we used Adam optimizer [80], an improved version of the classic stochastic gradient descent optimization algorithm which is extensively used in deep learning applications. The feed-forward models have been implemented using Keras [81] with Tensorflow as backend, the LSTM network (both the training modes) using PyTorch [82].

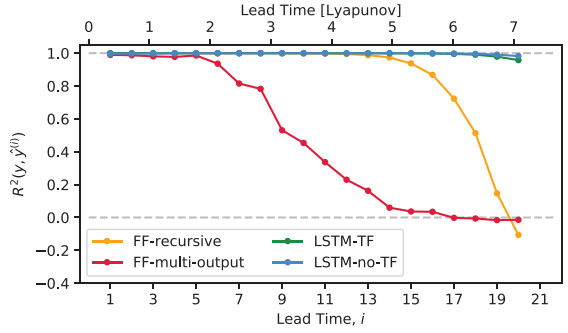
3. Results

We analyze the predictive power of the four considered neural predictors, FF-recursive, FF-multi-output, and LSTM with and without teacher forcing (LSTM-TF and LSTM-no-TF), on the chaotic systems of Section 2.4. We first train the FF-recursive predictor. As said in Section 2.1, it is the most diffused way for predicting nonlinear time series and thus represents a benchmark. In addition, it allows us defining the length of the predictive horizon (i.e. the number h of predicted time steps) for each chaotic system. Once the single-step predictor has been identified, we apply it recursively and compute the R^2-score for each time step ahead on the test dataset. We set the length h of the predicting horizon as the first step for which R^2-score becomes negative. Once h has been defined, it is possible to train the multi-output and the LSTM predictors.

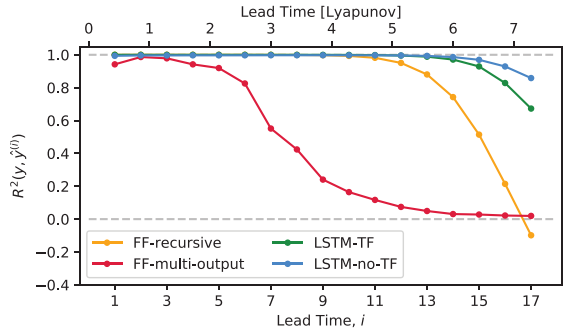
The R^2-score for each time step ahead during testing is reported in Fig. 4 for each predictor (lines on the same panel) and data generator (same color line on different panels). The main result is that our four predictors rank in the same order in all cases. Leveraging on the wide range of complexity of the considered data generators and the corresponding forecasting tasks, we can consider this result to be a general property of the considered neural architectures. A second remarkable result is the rather uniform performance of each predictor across different systems, read in terms of the number of Lyapunov times (LTs) predicted with sufficient R^2-score (quality threshold 0.7 in Table 2). Note that we intentionally designed our setup to challenge the predictors on tasks with differ-

Table 2
Overall performance of the predictors over the four considered chaotic systems.

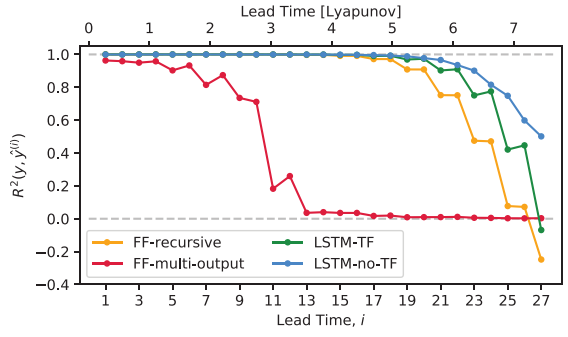
System	Lags (d)	Leads (h)	LT	Predictor	# LT $R^2 > 0.7$	$\langle R^2 \rangle$
Logistic	1	20	2.83	FF-recursive	6.01	0.85
				FF-multi-output	2.83	0.46
				LSTM-TF	> 7.08	> 0.99
				LSTM-no-TF	> 7.08	> 0.99
Hènon	2	17	2.33	FF-recursive	6.00	0.83
				FF-multi-output	2.57	0.43
				LSTM-TF	6.86	0.96
				LSTM-no-TF	> 7.28	0.98
3D generalizedHènon	3	27	3.62	FF-recursive	6.07	0.82
				FF-multi-output	2.76	0.35
				LSTM-TF	6.62	0.89
				LSTM-no-TF	6.90	0.94
10D generalizedHènon	10	109	15.87	FF-recursive	5.67	0.84
				FF-multi-output	0.00	0.13
				LSTM-TF	5.67	0.90
				LSTM-no-TF	6.23	0.94



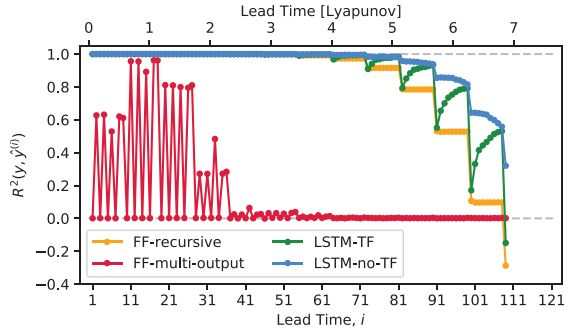
(a) Logistic



(b) Hénon



(c) 3D generalized Hénon



(d) 10D generalized Hénon

Fig. 4. $R^2(\mathbf{y}, \hat{\mathbf{y}}^{(i)})$-score obtained with the four predictors (see colored lines) on the logistic (a), Hènon (b), 3D generalized Hènon (c), and 10D generalized Hènon (d) maps. Performances computed on the test dataset.

ent complexity. For this reason, we have used same-size networks and datasets to predict dynamics with different complexity. This is however an unfair setup for the comparison of the predictor performances on different tasks and, indeed, the performance of all predictors decays with the dimension and complexity of the forecasted system. Nonetheless, all neural architectures seem to show a universal predictive power, in line with the preliminary results in [14], where a fair comparison has been carried out for the FF-recursive predictor.

The FF-recursive predictor can mimic almost perfectly the behavior of the real system for 5 LTs. After that, the performances briefly degrade to the point that chaos takes over between 7 and 8 LTs. The FF-multi-output predictor definitely provides the poorest performances: the R^2-score drops after 2 LTs and then vanishes, showing that the predictions become close to the mean value of the training dataset. For the three low-dimensional systems, the maps to be approximated (the iterations of the system's map, see Fig. 3) become so complex, for increasing lead time, that the network is not able to reproduce them (Fig. 4a-c). For the 10D generalized Hènon, the large horizon h (each LT corresponds to 15 time steps) is an additional issue. To perform similarly to the low-dimensional cases, the FF-multi-output predictor should reproduce more than 30 iterated maps independently, since the sequential nature of the outputs is not considered. This task is so complex that the optimization always gets stuck in local minima, where only few non-necessarily consecutive steps are accurately predicted. FF-multi-output performances are thus acceptable only on short forecasting horizon (small h). The two LSTM predictors exhibit a trend similar to the FF-recursive one, always providing a higher precision especially after 4–5 LTs. Training the LSTM without TF gives the best performance in all cases.

Note the behavior of the R^2-score in the two generalized Hènon maps (Fig. 4c-d), which is system-specific: the performance deteriorates step-wise while increasing the number of steps ahead. In fact, this step-wise behavior is peculiar of the generalized Hènon map; more precisely, of recurrences in which the nonlinear map giving $y(t+1)$ is applied only to a few last samples of the lag-window of d previous samples ($y(t-8)$ and $y(t-9)$ in the 10D generalized Hènon map). The first $d-1$ steps ahead are forecasted using actual data (null error), required to initialize the predictor. After that, these $d-1$ predictions (with a certain error), are used to forecast the values from d to $2\cdot(d-1)$ steps ahead, and so on.

The 10D map also highlights the relevance of the exposure bias problem (i.e., the discrepancy between training and inference modes) for LSTM networks with TF. The R2-score of the LSTM-TF also exhibits a structure organized in blocks of length $d-1$, but the trend within each block is increasing. This counterintuitive behavior shows that the predictor particularly suffers at the beginning of each block, because predictions of lower quality start to be used as input, while this has not been done in training. The propagation of the information through the hidden states then interestingly allows the predictor to recover, until the next performance drop at the beginning of the next block.

Looking across the panels of Fig. 4 and at the results summarized in Table 2, it can be seen that the FF-recursive predictor is able to maintain rather similar performances on systems of different order. Conversely, the LSTM predictors seem to suffer more in the two generalized Hènon maps. Indeed, the forecasting task of a generalized Hènon map is more critical for a recurrent network than it is for a FF architecture. Because the prediction $\hat{y}(t+1)$ does not depend on the input from $y(t), \ldots, y(t-d+3)$, a FF net just has to nullify the corresponding weights. Conversely, an RNN has to perform a two-fold task: propagate the information through time over the d lags of the network (for the systems here considered, d equals the number of state variables) to remember past inputs and reproduce the nonlinear function. The results show that training the LSTM predictors without TF strongly mitigates this issue, supporting our hypothesis that this training method allows the proper information flow over subsequent time steps.

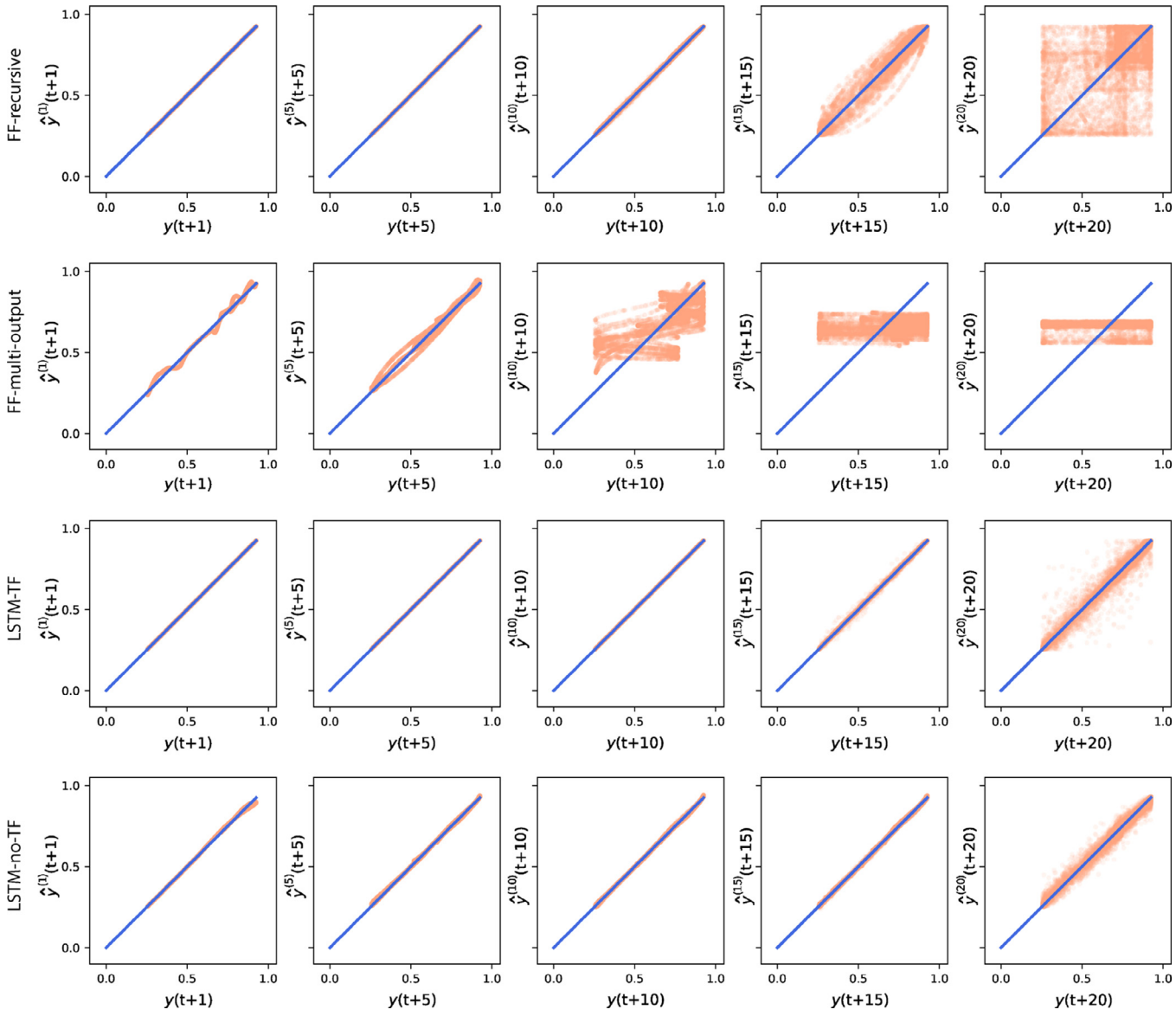


Fig. 5. Scatter plots of the points $(y(t), \hat{y}^{(i)}(t))$ for the logistic map, $i = 1, 5, 10, 15, 20$ (columns 1 to 5; different predictors in rows 1 to 4). Perfect predictions aligns on the diagonal.

A different perspective to look at the results is to represent each pair of ground truth and prediction in a scatter plot (see Fig. 5 for the case of the logistic map). When forecasting is perfect, the points lay on the diagonal. It is interesting to note that the LSTM-no-TF predictor makes small mistakes in the first time step, compared to the TF case (compare rows 3 and 4, column 1, in Fig. 5). This is a direct consequence of the approach used to train the net, which takes into account the whole horizon, so that a small error in the initial steps can be, in a sense, useful to obtain a better overall performance. In other words, the LSTM-no-TF predictor is specifically trained to provide the best performance on the entire sequence. Conversely, the FF-recursive and LSTM-TF predictors follow a different strategy: they are essentially both optimized to predict one step ahead. For this reason the scatter plots of the first time step (rows 1 and 3, column 1, Fig. 5) are almost perfectly aligned on the diagonal. The same trend can be seen even for $(y(t+5), \hat{y}(t+5))$ and $(y(t+10), \hat{y}(t+10))$, while the situation change for $(y(t+15), \hat{y}(t+15))$ and $(y(t+20), \hat{y}(t+20))$, where the points generated with the LSTM-no-TF are much more aligned on the diagonal than those obtained with the FF-recursive and LSTM-TF predictors.

3.1. Performance distribution over the system's attractor

LLE and LT are average quantities which characterize the entire chaotic attractor. However, the same predictor could provide different performances in different regions of the attractor. For instance, starting a prediction near a saddle point typically affects the score of the prediction negatively. Fig. 6 shows the chaotic attractor of the 3D generalized Hènon system reconstructed via delayed coordinate embedding (which in this case is equivalent to the state space). The samples of the test set have been organized in triplets $(y(t), y(t-1), y(t-2))$ which identify a point in the reconstructed state space. We then perform a forecasting of the following $h = 27$ steps ahead. The color assigned to each point depends on the performance obtained when the forecasting starts from the point itself, computed as:

$$R^2(y(t+1:t+h), \hat{y}^{(1:h)}(t+1:t+h)) = R^2\left([y(t+1), \ldots, y(t+h)], [\hat{y}^{(1)}(t+1), \ldots, \hat{y}^{(h)}(t+h)]\right). \quad (9)$$

The attractor relative to the FF-recursive predictor (Fig. 6a) is manly represented in warm colors (R^2-score close to 1). Points

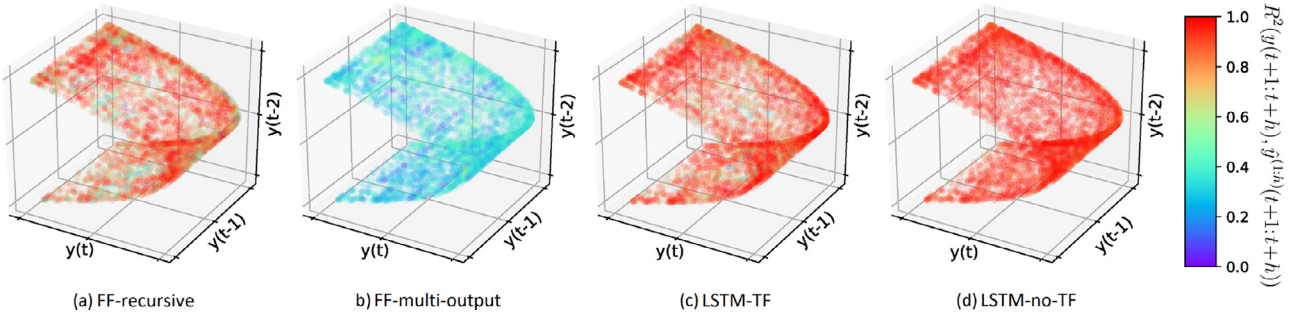


Fig. 6. Performance distribution of the four predictors over the chaotic attractor of the 3D generalized Hènon map. The color of each point indicates the 27-step forecasting performance when starting from that point.

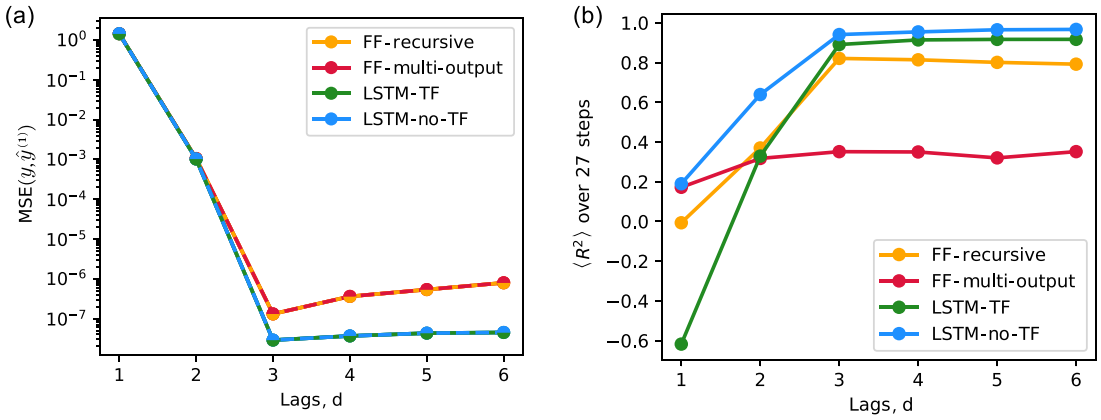


Fig. 7. One-step ahead MSE (a) and R^2-score computed on a 27-step horizon (b) for the 3D generalized Hénon map, with d ∈ [1, 6].

depicted with cold hues do not seem to have any particular distribution. The FF-multi-output network (Fig. 6b) confirms the low precision already observed in Fig. 4. The LSTM-TF (Fig. 6c) achieves better performances than the FF-recursive predictor, but still shows a similar random pattern. The best performance and uniformity is obtained with the LSTM-no-TF predictor (Fig. 6d).

3.2. Sensitivity to the embedding dimension

Till now, we always assumed to have a one-to-one mapping between d consecutive samples of the dataset and the next (i.e., the dataset embedding dimension). In all cases here considered, it is actually possible to compute d analytically from the systems' equations. However, this is not possible in general. Moreover, the system's equations are typically unknown in real-world applications, so that d has to be identified by means of numerical techniques. One way, especially suitable for ANNs, consists of optimizing d as an additional hyper-parameter [83–86]. Since hyper-parameters tuning is a time-consuming task, it is worthwhile to find a suitable set of values of d, in order limit the computational effort. Some insight can be obtained from the data auto-correlation function or by means of nonlinear time-series tools (such as the false-neighbors method or computing the dataset fractal dimension for increasing d [73]).

To investigate which predictor is more robust with respect to an uncertainty about the value of d, we reconsider the 3D generalized Hènon map. We first focus on the single-step prediction, comparing the performance of the FF and LSTM predictors (note that, when $h = 1$, the recursive and multi-output FF predictors are equivalent, as well as the LSTM with or without TF). Fig. 7a reports the one-step ahead MSE and shows that $d < 3$ is not sufficient. As expected, $d = 3$ guarantees the best performance, in accordance with the conclusion analytically obtained from the equations of the model. When $d > 3$, the behavior of the LSTM predictors is nearly the same as the one obtained with only the necessary inputs. The extra available inputs have essentially no impact on the outputs, since they are filtered by the sequential LSTM architecture. This allows the predictor to correctly learn them as not useful to improve the prediction. The FF structure appears to be less robust, because there is a static link between input and output. In principle, they should have the ability to avoid dependence on useless inputs, by setting to zero the weights of the connection between the redundant inputs and the first hidden layer. However, after the training, the value of these weights is close, but not exactly equal, to zero, so that the effect of useless inputs remains visible (note, however, the logarithmic scale in Fig. 7a).

Fig. 7b shows the average $\langle R^2 \rangle$ computed over 27 steps for different lags d (the same considered in Fig. 7a). The LSTM-no-TF predictor again exhibits the best performance both when the embedding dimension is underestimated ($d < 3$) and overestimated ($d > 3$). Comparing Fig. 7a and Fig. 7b, we can see that while the numerical imprecision caused by useless inputs is almost negligible in one-step (note again the logarithmic scale in Fig. 7a), it becomes more significant when considering the whole forecasting horizon (27 steps in Fig. 7b), also showing considerable differences between predictors. The FF-multi-output is definitively the less robust predictor. The FF-recursive is fairly robust, though it suffers the chaotic nature of the data. LSTM structures are the most robust to the overestimation of the number of necessary lags. Note that the average R^2-score even increases by exploiting the redundant inputs. Without TF, the predictor is also robust to the underestimation of d, confirming that TF should not be used when dealing with forecasting tasks.

4. Discussion

4.1. Simplicity and robustness of the experimental setting

An exhaustive analysis of all the aspects of the problem (i.e., the class of chaotic systems, the number of samples, and the features of the neural architectures) would require an outrageous computational effort. We thus fix these elements in order to avoid choices which could potentially affect the results.

First, we avoid continuous-time systems because numerical integration unavoidably introduces errors, especially in chaotic systems, so that the neural network ends up learning the discrete-time representation of the real system defined by the integrator. Besides numerical integration, which can be accurately performed, the choice of the sampling time strongly affects the results. A short sampling time makes the single-step predictor almost equal to the identity map. This dramatically hurts the performance of FF-recursive and LSTM-TF predictors. Moreover, many steps are required for each Lyapunov time, preventing FF-multi-output predictors to reach acceptable performances (an analogous situation is represented in Fig. 4d). On the other hand, a long sampling time makes one-step mapping too complex (see Fig. 3). In addition, we use systems that can be explicitly formulated as a multi-step recurrence on a single output variable $y(t)$. For these systems, the dimension of the recurrence gives the number of inputs d (lags) that make the forecasting problem feasible (the embedding dimension of the dataset), hence avoiding the problems related to its numerical determination.

Second, we fix the number of hidden layers (3), each one with a fixed number of neurons (10). Note that, for FF architectures, the number of the networks parameters still depends on the number of inputs (the d lags of the system to be learned) and outputs (the predictive horizon length h), which are system's specific. After considering several combinations, we selected the one which ensures a reasonable compromise between performances and training convergence.

We also maintain a constant number of samples (50,000). As we have underlined while presenting our results, we do it intentionally, in connection with the choice of the chaotic data generators, to challenge our predictors on tasks with different complexity. On the other hand, scaling the datasets' and the network's dimensions with the complexity of the task (in our case a good proxy would be the fractal dimension of the chaotic attractor) would be necessary to compare the performance of each predictor across different systems [14].

To further support the robustness of our main result—performance ranking among the four predictors on each of the considered forecasting tasks—we confirm that we always observed the ranking of Fig. 4 in all unreported tests that we have run before deciding to fix the experimental setup as here described.

4.2. Training with and without teacher forcing

We usually link the concept of training with or without teacher forcing to the specific case of recurrent neural networks. However, the same topic have been widely discussed, under different names, also with reference to FF architectures and, more in general, to empirical identification and prediction of dynamical systems.

In the case of FF neural networks, the topic have been discussed since the early nineties by referring to series-parallel (analogue to TF) and parallel (no-TF) training modes [87]. In contrast to what happen in NLP, where TF is considered as the standard approach, both TF and no-TF are widely adopted in system identification and time series prediction, with results that are application specific [88,89]. The real advantage of TF in FF nets is that, as we have explained in the description of the FF-recursive predictor, it breaks the output-to-input feedback during training and hence transforms a dynamic task into a static one. For this reason, training with TF is amenable to parallelization and thus more efficient from a computational perspective [89]. Conversely, in recurrent nets there are two feedbacks: one from output at the time t, to input at time $t+1$ as in FF nets, and another in the hidden states. The first can be removed by adopting TF, while the flow of information through the hidden states cannot be teacher forced. As consequence, even adopting TF, we cannot process each time step as an independent sample as it happens with FF architectures. The two approaches thus require a similar computational effort when training recurrent nets.

Similar results appeared in the field of empirical modeling with polynomial functions. In this context, training with TF shares the main concept with the minimization of the one-step prediction error, while no-TF looks at the performance over a multi-step horizon (simulation error) and is argued as a promising technique for the identification of polynomial models [90]. The computational effort required by the two approaches is however significantly different, because the minimization of the prediction error is simply obtained with the least squares formula.

The use of TF has been widely discussed also in the context of reservoir computing. In fact, reservoir computers are RNNs trained in a specific way. Instead of adopting the usual backpropagation almost always used to train neural nets (specifically the backpropagation through time), the recurrent weights are randomly set and only the parameters of the hidden-to-output liner layer are identified. Under TF, their optimal value is computed with the least squares formula. Without TF, the peculiar trick (and advantage) of reservoir computing gets lost, as feeding the prediction at time $t+k$ as input for the prediction at time $t+k+1$ makes the loss a nonlinear function of the output weights.

To the best of our knowledge, our paper is the first contributing to the debate in the field of chaotic dynamics prediction.

4.3. Advanced feed-forward architectures

In recent years, recurrent networks have been losing ground to high-performing FF architectures (abandoning the traditional fully-connected structure) in several machine learning tasks.

A special type of FF models which have been successfully adapted to sequence tasks are convolutional neural networks [91,92]. 1D convolution through time has the main role of filtering noise and/or of extracting complex features from the time series (see, for instance, the seminal paper by LeCun [93], and other recent practical applications [94,95]). Convolutional layers have also been used in hybrid architectures to preprocess the signal before feeding it into recurrent layers or, alternatively, on top of them. In chaotic dynamics prediction, there is a mapping from input to output and the input are noise-free. Convolution is thus not only unnecessary, but it could also have negative effective distorting the input and making the reproduction of the nonlinear map more complex.

Other FF architectures, named transformers [96], recently shown outstanding performances on NLP-related tasks, thanks to the so-called attention mechanism [97]. Transformers are specifically thought for NLP (e.g., text translation from a language to another, or question answering), inherently different from the prediction of a numerical time series. This kind of problems requires an encoder which compresses the information stored in the input sequence in a context vector, and a decoder that starting from the context vector, produces the output sequence. Encoder and decoder have different parameters because their roles are inherently different. In our neural predictors, on the other hand, inputs and outputs are samples of the same variable and thus the encoder-decoder architecture is unjustified. Another issue is that the last

value of the input values has to be processed twice: it is the last input to the encoder, but also the first of the decoder. In principle one can adopt such structures in numerical time series forecasting [98], but it seems to be a mind-bending solution. Transformers are always trained with TF and present a feed-forward structure. In a certain sense, when considering a simple case as the logistic map (the single step prediction is equivalent to reproducing a parabola), transformers can be seen as complex versions of the FF-recursive predictor.

4.4. Chaotic dynamics in recurrent networks

Recurrent neural networks are dynamical system which are know to frequently exhibit a chaotic behavior [99]. This is usually considered as a remarkable issue, especially in those applications (e.g., control) which require stability properties. For this reason, many authors proposed alternative formulations of the RNNs' equations and imposed apposite constraints to guarantee the model stability [100].

In the context considered in this work, we try to reproduce chaotic dynamics and thus imposing the stability of our neural model would be dangerous, since the real system is not stable. We think that the RNNs natural aptitude for being chaotic [101] would actually help them in reproducing the dynamics of the considered oscillators.

5. Conclusion

In this paper, we compared the forecasting accuracy and robustness of four different predictors based on feed-forward and recurrent neural nets on artificial, noise-free time series generated by chaotic oscillators. Specifically, we considered feed-forward (FF) recursive and multi-output nets, and a recurrent architecture with LSTM cells. In particular, regarding the LSTM nets, we questioned the so-called teacher forcing (TF) training method in the context of time series forecasting. This method, successfully put forward in natural language processing applications, consists in using the ground truth from the prior time steps instead of feeding back the network predictions as input for the following steps. Adopting TF, the network is not trained to compensate the effect of one-step prediction errors on multi-step predictions. In other words, the LSTM-TF focuses on the single-step forecasting, and thus has the same issues of the FF-recursive predictor.

We proposed a training method for RNNs that avoids TF (LSTM-no-TF in our figures) and we performed an in depth comparison with TF and with FF predictors. We showed that LSTM predictors outperform FF-recursive and multi-output ones, in four standard chaotic systems: logistic, Hènon and two generalized Hènon maps. We also showed that the predictive power of LSTM predictors can be further enhanced by training without TF. The LSTM-no-TF predictor shows all the strengths of the three competitors: it well reproduces the functions mapping previous into future samples (as FF-recursive), it is optimized over the entire forecasting horizon (as FF-multi-output), and its internal dynamics makes it suitable for sequential tasks (as LSTM-TF). In addition, the LSTM-no-TF turns out to have the best performance and robustness with respect to the network's hyper-parameters, including the number of autoregressive terms given as input. Analyzing the predictive power within the same chaotic attractor, the situation is quite similar: LSTM, especially with no-TF, ensures a more uniform predictive power with respect to the two FF nets.

In general, we can conclude that prediction and system identification are surely related but different tasks. A FF network trained on the one-step prediction task can mimic fairly well the behavior of the real chaotic system but other configurations exhibit greater predictive power and robustness over multiple-steps.

The results obtained seems to be robust, since they are in accordance with the one obtained by recent works in the field [38,42,56,61]. Machine learning tools predicts almost perfectly the future evolution of the system for 5/6 Lyapunov times; after that predictions start degrading.

Particularly interesting is the robustness of the LSTM-no-TF prediction when dealing with an uncertain value of the dataset embedding dimension. In this study we focused only on artificial data generated by well-known systems that can be formulated as nonlinear regression on the output variable. We thus a-priori knew the correct number of inputs (the dataset embedding dimension). Dealing with real-world time series, one has to estimate the number of autoregressive terms to include in the input, so that the robustness with respect to this source of uncertainty is undoubtedly a remarkable advantage.

This is not the only relevant aspect about the practical application of this paper. In fact, the presented results, showed for convenience in discrete-time maps, are expected to hold in a broader context, including continuous-time systems and real-world datasets (e.g., signal related to meteorology, fluid dynamics, biometric, econometrics) [102,103]. Further research in this direction is needed to confirm the effectiveness of FF and recurrent neural predictors also on real processes supposed to be chaotic.

The availability of accurate multi-step predictions is extremely important also for the performance of control systems, nowadays present in essentially any human activity. This is particularly true for control schemes adopting a receding horizon, such as model predictive control, and in dealing with complex systems for which a model-free or hybrid knowledge-learning-based approach is necessary. Traditionally adopted in process control in several fields, model predictive control is jointly used with learning in many modern applications, such as the control of smart grids and autonomous vehicles [104].

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Matteo Sangiorgio: Conceptualization, Formal analysis, Methodology, Software, Validation, Writing - original draft, Writing - review & editing. **Fabio Dercole:** Conceptualization, Formal analysis, Methodology, Supervision, Writing - original draft, Writing - review & editing.

Acknowledgments

The authors are grateful to Prof. Giorgio Guariso and to the two anonymous reviewers for their constructive comments and suggestions.

References

[1] Farmer JD, Sidorowich JJ. Predicting chaotic time series. Phys Rev Lett 1987;59(8):845.
[2] Casdagli M. Nonlinear prediction of chaotic time series. Physica D 1989;35(3):335–56.
[3] Jones RD, Lee Y-C, Barnes C, Flake GW, Lee K, Lewis P, et al. Function approximation and time series prediction with neural networks. In: Proceedings of the 1990 IJCNN international ioint conference on neural networks. IEEE; 1990. p. 649–65.
[4] Navone H, Ceccatto H. Learning chaotic dynamics by neural networks. Chaos, Solitons & Fractals 1995;6:383–7.
[5] Verdes P, Granitto P, Navone H, Ceccatto H. Forecasting chaotic time series: global vs. local methods. Novel Intelligent Automation and Control Systems 1998;1:129–45.

[6] Mukherjee S, Osuna E, Girosi F. Nonlinear prediction of chaotic time series using support vector machines. In: Neural networks for signal Processing VII. Proceedings of the 1997 IEEE Signal Processing Society Workshop. IEEE; 1997. p. 511–20.
[7] Lin T-N, Giles CL, Horne BG, Kung S-Y. A delay damage model selection algorithm for narx neural networks. IEEE Trans Signal Process 1997;45(11):2719–30.
[8] Bonnet D, Labouisse V, Grumbach A. /Spl delta/-narma neural networks: a new approach to signal prediction. IEEE Trans Signal Process 1997;45(11):2799–810.
[9] Yeh J-P. Identifying chaotic systems using a fuzzy model coupled with a linear plant. Chaos, Solitons & Fractals 2007;32(3):1178–87.
[10] Atsalakis G, Skiadas C, Nezis D. Forecasting chaotic time series by a neural network. In: Proceedings of the 8th international conference on applied stochastic models and data analysis, Vilnius, Lithuania, 30; 2008. p. 7782.
[11] Han M, Wang Y. Analysis and modeling of multivariate chaotic time series based on neural network. Expert Syst Appl 2009;36(2):1280–90.
[12] Woolley JW, Agarwal P, Baker J. Modeling and prediction of chaotic systems with artificial neural networks. Int J Numer Methods Fluids 2010;63(8):989–1004.
[13] Covas E, Benetos E. Optimal neural network feature selection for spatial-temporal forecasting. Chaos: An Interdisciplinary Journal of Nonlinear Science 2019;29(6):063111.
[14] Dercole F, Sangiorgio M, Schmirander Y. An empirical assessment of the universality of anns to predict oscillatory time series. Accepted for publication in IFAC-PapersOnLine 2020.
[15] Leung H, Lo T, Wang S. Prediction of noisy chaotic time series using an optimal radial basis function neural network. IEEE Trans Neural Networks 2001;12(5):1163–72.
[16] Guerra FA, Coelho LdS. Multi-step ahead nonlinear identification of lorenzs chaotic system using radial basis neural network with learning by clustering and particle swarm optimization. Chaos, Solitons & Fractals 2008;35(5):967–79.
[17] Ding H-L, Yeung DS, Ma Q-L, Ng WW, Wu D-L, Li J-C. Prediction of chaotic time series using l-gem based rbfnn. In: Proceedings of the 2009 international conference on machine learning and cybernetics, 2. IEEE; 2009. p. 1172–7.
[18] Todorov Y, Koprinkova-Hristova P, Terziyska M. Intuitionistic fuzzy radial basis functions network for modeling of nonlinear dynamics. In: Proceedings of the 2017 21st international conference on process control (PC). IEEE; 2017. p. 410–15.
[19] Van Truc N, Anh DT. Chaotic time series prediction using radial basis function networks. In: Proceedings of the 2018 4th international conference on green technology and sustainable development (GTSD). IEEE; 2018. p. 753–8.
[20] Masnadi-Shirazi M, Subramaniam S. Attractor ranked radial basis function network: anonparametric forecasting approach for chaotic dynamic systems. Sci Rep 2020;10(1):1–10.
[21] Maguire LP, Roche B, McGinnity TM, McDaid L. Predicting a chaotic time series using a fuzzy neural network. Inf Sci (Ny) 1998;112(1–4):125–36.
[22] Kuremoto T, Obayashi M, Yamamoto A, Kobayashi K. Predicting chaotic time series by reinforcement learning. In: Proc. of the 2nd iIntern. conf. on computational intelligence, robotics and autonomous sSystems (CIRAS 2003); 2003.
[23] Yang HY, Ye H, Wang G, Khan J, Hu T. Fuzzy neural very-short-term load forecasting based on chaotic dynamics reconstruction. Chaos, Solitons & Fractals 2006;29(2):462–9.
[24] Zhang J, Chung HS-H, Lo W-L. Chaotic time series prediction using a neuro-fuzzy system with time-delay coordinates. IEEE Trans Knowl Data Eng 2008;20(7):956–64.
[25] Chen Z, Lu C, Zhang W, Du X. A chaotic time series prediction method based on fuzzy neural network and its application. In: Proceedings of the 2010 international workshop on chaos-fractal theories and applications. IEEE; 2010. p. 355–9.
[26] Kuremoto T, Obayashi M, Kobayashi K, Hirata T, Mabu S. Forecast chaotic time series data by dbns. In: Proceedings of the 2014 7th international congress on image and signal processing. IEEE; 2014. p. 1130–5.
[27] Penkovsky B, Porte X, Jacquot M, Larger L, Brunner D. Coupled nonlinear delay systems as deep convolutional neural networks. Phys Rev Lett 2019;123(5):54101.
[28] Zhang J-S, Xiao X-C. Predicting chaotic time series using recurrent neural network. Chin Phys Lett 2000;17(2):88.
[29] Cannas B, Cincotti S, Marchesi M, Pilo F. Learning of chua's circuit attractors by locally recurrent neural networks. Chaos, Solitons & Fractals 2001;12(11):2109–15.
[30] Cannas B, Cincotti S. Neural reconstruction of lorenz attractors by an observable. Chaos, Solitons & Fractals 2002;14(1):81–6.
[31] Han M, Xi J, Xu S, Yin F-L. Prediction of chaotic time series based on the recurrent predictor neural network. IEEE Trans Signal Process 2004;52(12):3409–16.
[32] Ma Q-L, Zheng Q-L, Peng H, Zhong T-W, Xu L-Q. Chaotic time series prediction based on evolving recurrent neural networks. In: Proceedings of the 2007 international conference on machine learning and cybernetics, 6. IEEE; 2007. p. 3496–500.
[33] Cechin AL, Pechmann DR, de Oliveira LP. Optimizing markovian modeling of chaotic systems with recurrent neural networks. Chaos, Solitons & Fractals 2008;37(5):1317–27.
[34] Maathuis H, Boulogne L, Wiering M, Sterk A. Predicting chaotic time series using machine learning techniques. In: Preproceedings of the 29th Benelux conference on artificial intelligence (BNAIC 2017); 2017. p. 326–40.
[35] Yu R, Zheng S, Liu Y. Learning chaotic dynamics using tensor recurrent neural networks. In: Proceedings of the ICML; 2017.
[36] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT press; 2016.
[37] Butcher JB, Verstraeten D, Schrauwen B, Day C, Haycock P. Reservoir computing and extreme learning machines for non-linear time-series data analysis. Neural networks 2013;38:76–89.
[38] Pathak J, Lu Z, Hunt BR, Girvan M, Ott E. Using machine learning to replicate chaotic attractors and calculate lyapunov exponents from data. Chaos: An Interdisciplinary Journal of Nonlinear Science 2017;27(12):121102.
[39] Lu Z, Pathak J, Hunt B, Girvan M, Brockett R, Ott E. Reservoir observers: model-free inference of unmeasured variables in chaotic systems. Chaos: An Interdisciplinary Journal of Nonlinear Science 2017;27(4):41102.
[40] Canaday D, Griffith A, Gauthier DJ. Rapid time series prediction with a hardware-based reservoir computer. Chaos: An Interdisciplinary Journal of Nonlinear Science 2018;28(12):123119.
[41] Lu Z, Hunt BR, Ott E. Attractor reconstruction by machine learning. Chaos: An Interdisciplinary Journal of Nonlinear Science 2018;28(6):061104.
[42] Pathak J, Hunt B, Girvan M, Lu Z, Ott E. Model-free prediction of large spatiotemporally chaotic systems from data: a reservoir computing approach. Phys Rev Lett 2018;120(2):24102.
[43] Antonik P, Gulina M, Pauwels J, Massar S. Using a reservoir computer to learn chaotic attractors, with applications to chaos synchronization and cryptography. Physical Review E 2018;98(1):012215.
[44] Vlachas PR, Byeon W, Wan ZY, Sapsis TP, Koumoutsakos P. Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 2018;474(2213):20170844.
[45] Wan ZY, Vlachas P, Koumoutsakos P, Sapsis T. Data-assisted reduced-order modeling of extreme events in complex dynamical systems. PLoS ONE 2018;13(5):e0197704.
[46] Madondo M, Gibbons T. Learning and modeling chaos using lstm recurrent neural networks. In: MICS 2018 Proceedings; 2018.
[47] Yeo K. Data-driven reconstruction of nonlinear dynamics from sparse observation. J Comput Phys 2019.
[48] Weng T, Yang H, Gu C, Zhang J, Small M. Synchronization of chaotic systems and their machine-learning models. Physical Review E 2019;99(4):042203.
[49] Hassanzadeh P, Chattopadhyay A, Palem K, Subramanian D. Data-driven prediction of a multi-scale lorenz 96 chaotic system using a hierarchy of deep learning methods: reservoir computing, ann, and rnn-lstm.. Bulletin of the American Physical Society 2019.
[50] Jüngling T, Lymburn T, Stemler T, Corrêa D, Walker D, Small M. Reconstruction of complex dynamical systems from time series using reservoir computing. In: Proceedings of the 2019 IEEE international symposium on circuits and systems (ISCAS). IEEE; 2019. p. 1–5.
[51] Jiang J, Lai Y-C. Model-free prediction of spatiotemporal dynamical systems with recurrent neural networks: role of network spectral radius. Physical Review Research 2019;1(3):33056.
[52] Zhu Q, Ma H, Lin W. Detecting unstable periodic orbits based only on time series: when adaptive delayed feedback control meets reservoir computing. Chaos: An Interdisciplinary Journal of Nonlinear Science 2019;29(9):93125.
[53] Haluszczynski A, Räth C. Good and bad predictions: assessing and improving the replication of chaotic attractors by means of reservoir computing. Chaos: An Interdisciplinary Journal of Nonlinear Science 2019;29(10):103143.
[54] Nakai K, Saiki Y. Machine-learning construction of a model for a macroscopic fluid variable using the delay-coordinate of a scalar observable. arXiv preprint arXiv:190305770 2019.
[55] Fan H, Jiang J, Zhang C, Wang X, Lai Y-C. Long-term prediction of chaotic systems with machine learning. Physical Review Research 2020;2(1):012080.
[56] Vlachas P, Pathak J, Hunt B, Sapsis T, Girvan M, Ott E, et al. Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics. Neural Networks 2020.
[57] Gers FA, Eck D, Schmidhuber J. Applying lstm to time series predictable through time-window approaches. In: Proceedings of the Neural Nets WIRN Vietri-01. Springer; 2002. p. 193–200.
[58] Cao L, Hong Y, Fang H, He G. Predicting chaotic time series with wavelet networks. Physica D 1995;85(1–2):225–38.
[59] López-Caraballo C, Salfate I, Lazzús J, Rojas P, Rivera M, Palma-Chilla L. Mackey-glass noisy chaotic time series prediction by a swarm-optimized neural network. In: Journal of Physics: Conference Series, 720. IOP Publishing; 2016.
[60] Akritas P, Antoniou I, Ivanov V. Identification and prediction of discrete chaotic maps applying a chebyshev neural network. Chaos, Solitons & Fractals 2000;11(1–3):337–44.
[61] Pathak J, Wikner A, Fussell R, Chandra S, Hunt BR, Girvan M, et al. Hybrid forecasting of chaotic processes: using machine learning in conjunction with a knowledge-based model. Chaos: An Interdisciplinary Journal of Nonlinear Science 2018;28(4):41101.
[62] Lei Y, Hu J, Ding J. A hybrid model based on deep lstm for predicting high-dimensional chaotic systems. arXiv preprint arXiv:200200799 2020.
[63] Doan NAK, Polifke W, Magri L. Physics-informed echo state networks for chaotic systems forecasting. In: Proceedings of the International Conference on Computational Science. Springer; 2019. p. 192–8.
[64] Inoue H, Fukunaga Y, Narihisa H. Efficient hybrid neural network for chaotic time series prediction. In: Proceedings of the international conference on artificial neural nNetworks. Springer; 2001. p. 712–18.

[65] Okuno S, Aihara K, Hirata Y. Combining multiple forecasts for multivariate time series via state-dependent weighting. Chaos: An Interdisciplinary Journal of Nonlinear Science 2019;29(3):033128.
[66] Yang F-P, Lee S-J. Applying soft computing for forecasting chaotic time series. In: Proceedings of the 2008 IEEE international conference on granular computing. IEEE; 2008. p. 718–23.
[67] Atsalakis G, Tsakalaki K. Simulating annealing and neural networks for chaotic time series forecasting. Chaotic Modeling and Simulation 2012;1:81–90.
[68] Bakker R, Schouten JC, Giles CL, Takens F, Bleek CMvd. Learning chaotic attractors by neural networks. Neural Comput 2000;12(10):2355–83.
[69] Lim TP, Puthusserypady S. Error criteria for cross validation in the context of chaotic time series prediction. Chaos: An Interdisciplinary Journal of Nonlinear Science 2006;16(1):013106.
[70] Shi X, Feng Y, Zeng J, Chen K. Chaos time-series prediction based on an improved recursive levenberg–marquardt algorithm. Chaos, Solitons & Fractals 2017;100:57–61.
[71] He T, Zhang J, Zhou Z, Glass J. Quantifying exposure bias for neural language generation. arXiv preprint arXiv:190510617 2019.
[72] Takens F. Detecting strange attractors in turbulence. In: Proceedings of the Dynamical systems and turbulence, Warwick 1980. Springer; 1981. p. 366–81.
[73] Kennel MB, Brown R, Abarbanel HD. Determining embedding dimension for phase-space reconstruction using a geometrical construction. Physical review A 1992;45(6):3403.
[74] Sutskever I, Vinyals O, Le QV. Sequence to sequence learning with neural networks. In: Advances in neural information processing systems; 2014. p. 3104–12.
[75] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput 1997;9(8):1735–80.
[76] Williams RJ, Zipser D. A learning algorithm for continually running fully recurrent neural networks. Neural Comput 1989;1(2):270–80.
[77] Bengio S, Vinyals O, Jaitly N, Shazeer N. Scheduled sampling for sequence prediction with recurrent neural networks. In: Advances in neural information processing systems; 2015. p. 1171–9.
[78] Baier G, Klein M. Maximum hyperchaos in generalized hénon maps. Phys Lett A 1990;151(6–7):281–4.
[79] Richter H. The generalized henon maps: examples for higher-dimensional chaos. Int J Bifurcation Chaos 2002;12(6):1371–84.
[80] Kingma DP, Ba J. Adam: a method for stochastic optimization. arXiv preprint arXiv:14126980 2014.
[81] Chollet F, et al. Keras: the python deep learning library. Astrophysics Source Code Library 2018.
[82] Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, et al. Automatic differentiation in pytorch NIPS-W; 2017.
[83] Povinelli RJ, Johnson MT, Lindgren AC, Roberts FM, Ye J. Statistical models of reconstructed phase spaces for signal classification. IEEE Trans Signal Process 2006;54(6):2178–86.
[84] Manabe Y, Chakraborty B. A novel approach for estimation of optimal embedding parameters of nonlinear time series by structural learning of neural network. Neurocomputing 2007;70(7–9):1360–71.
[85] Yijie W, Min H. Prediction of multivariate chaotic time series based on optimized phase space reconstruction. In: Proceedings of the 2007 Chinese Control Conference. IEEE; 2007. p. 169–73.
[86] Maus A, Sprott J. Neural network method for determining embedding dimension of a time series. Commun Nonlinear Sci Numer Simul 2011;16(8):3294–302.
[87] Kumpati SN, Kannan P, et al. Identification and control of dynamical systems using neural networks. IEEE Trans Neural Networks 1990;1(1):4–27.
[88] Menezes Jr JMP, Barreto GA. Long-term time series prediction with the narx network: an empirical evaluation. Neurocomputing 2008;71(16–18):3335–43.
[89] Ribeiro AH, Aguirre LA. Parallel training considered harmful: comparing series-parallel and parallel feedforward network training. Neurocomputing 2018;316:222–31.
[90] Piroddi L, Spinelli W. An identification algorithm for polynomial narx models based on simulation error minimization. Int J Control 2003;76(17):1767–81.
[91] Oord Avd, Dieleman S, Zen H, Simonyan K, Vinyals O, Graves A, et al. Wavenet: a generative model for raw audio. arXiv preprint arXiv:160903499 2016.
[92] Bai S, Kolter JZ, Koltun V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:180301271 2018.
[93] LeCun Y, Bengio Y, et al. Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks 1995;3361(10).
[94] Pancerasa M, Sangiorgio M, Winkler DW, Ambrosini R, Saino N, Casagrandi R. Can advanced machine learning techniques help to reconstruct barn swallows long-distance migratory paths?. In: Artificial intelligence international conference. PremC; 2018. 89–89
[95] Pancerasa M, Sangiorgio M, Ambrosini R, Saino N, Winkler DW, Casagrandi R. Reconstruction of long-distance bird migration routes using advanced machine learning techniques on geolocator data. Journal of the Royal Society Interface 2019;16(155):20190031.
[96] Devlin J, Chang M-W, Lee K, Toutanova K. Bert: pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:181004805 2018.
[97] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. In: Advances in neural information processing systems; 2017. p. 5998–6008.
[98] Wu N, Green B, Ben X, O'Banion S. Deep transformer models for time series forecasting: the influenza prevalence case. arXiv preprint arXiv:200108317 2020.
[99] Laurent T, von Brecht J. A recurrent neural network without chaos. arXiv preprint arXiv:161206212 2016.
[100] Miller J, Hardt M. Stable recurrent models. arXiv preprint arXiv:180510369 2018.
[101] Li Z, Ravela S. On neural learnability of chaotic dynamics. arXiv preprint arXiv:191205081 2019.
[102] Matsumoto T, Nakajima Y, Saito M, Sugi J, Hamagishi H. Reconstructions and predictions of nonlinear dynamical systems: a hierarchical bayesian approach. IEEE Trans Signal Process 2001;49(9):2138–55.
[103] Siek M. Predicting storm surges: chaos, computational intelligence, data assimilation and ensembles: UNESCO-IHE Ph.D. thesis. CRC Press; 2011.
[104] Rosolia U, Zhang X, Borrelli F. Data-driven predictive control for autonomous systems. Annual Review of Control, Robotics, and Autonomous Systems 2018;1(2):259–86.