Physica D 482 (2025) 134880

Contents lists available at ScienceDirect

Physica D

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

End-to-end Artificial Intelligence to analyze dynamical processes: A linear benchmark test

Emanuele Salgarollo, João Valle, Matteo Sangiorgio *, Fabio Dercole

Department of Electronics, Information and Bioengineering, Politecnico di Milano, Via Giuseppe Ponzio, 34, Milan, 20133, Italy

ARTICLE INFO

Communicated by Dmitry Pelinovsky

Keywords:
Machine learning
Recurrent neural networks
Linear systems
Eigenvalues

ABSTRACT

We envisage AI architectures to analyze complex time series in an end-to-end fashion, meaning that the quantitative metrics of the time series are learned directly from data, without the use of specific human-thought algorithms. That is, we challenge AI to learn those specific algorithms. We present a first step in this direction, a benchmark test on linear dynamical processes. We tackle the archetypical task of learning the eigenvalues of the state-transition matrix of a linear (discrete-time, stable) dynamical system, from output data. We train a scalable LSTM neural network with artificially generated data from random matrices of dimension 2-to-5. With noise-free data, the performance of the trained network is very good (average $R^2 = 0.955$), especially in estimating the dominant eigenvalues, whereas there is space for improvements on non-dominant real eigenvalues and on the dimension of the generating matrix. Remarkably, the performance is robust to measurement noise and the network outperforms the mean-square identification of the corresponding AR process (the latter giving exact eigenvalues on noise-free data) at noise standard deviation starting from 10^{-6}.

1. Introduction

Recent advancements in Artificial Intelligence (AI) have shifted focus towards end-to-end learning approaches, where Neural Network (NN) architectures learn complex tasks directly from raw data, with no need to learn sub-tasks or make use of traditional algorithms in between sub-tasks. Splitting complex tasks into sub-tasks or adapting traditional algorithms to take advantage of AI are very challenging activities, with often unclear impacts on the final performance of the AI system, making end-to-end approaches very appealing to minimize human intervention.

This paradigm has been successfully applied for the first time in the field of computer vision, especially in classification tasks. In the early stage of computer vision, prefilters [1,2] were usually applied before input into a NN, requiring an intermediate algorithm. However, AlexNet [3] was able to classify images from the ImageNet dataset [4] with high accuracy directly from raw data, avoiding prefiltering and achieving a state-of-the-art performance for the time. Image classification paved the way for the end-to-end learning era. Other fields were impacted, such as natural language processing with advanced generative models [5–8], autonomous driving [9–12], weather forecasting [13], disease diagnosis [14,15], fault detection [16,17], and several other applications [18,19].

Several NN architectures have been designed to follow the end-to-end paradigm, dealing directly with raw data. Examples include VGG [20] and ResNet [21] for computer vision, WaveNet [22] and DeepSpeech [23] for speech recognition, and Transformers [24] for natural language processing. However, many traditional engineering domains still rely on modular frameworks, where Machine Learning (ML) techniques are used for some sub-tasks and traditional algorithms, e.g., those based on linear algebra, for others [10,25–30].

In time series analysis and, more broadly, in the context of system dynamics, ML impacted two related tasks: the forecasting of future samples of the series, a very practical task, and the identification of a model able to generate artificial series with similar dynamical feature, a more conceptual task very important for the understanding and interpretation of complex phenomena. They are strictly related, as forecasting is the mapping between past and future samples, while model identification only requires the prediction one step ahead. Moreover, long-term forecasting and, as well, long-term simulations of the identified models do have theoretical limits for in most complex systems, such as chaotic ones [31–34].

ML is currently state-of-the-art for data forecasting and model identification [35–40]. Strictly speaking, ML is applied end-to-end to both tasks. However, they are not significant end-to-end ML applications, because input and output of the NN are data of the same nature, samples of the system's observable, so that the NN does not need to learn any specific algorithm to elaborate samples into quantities of different nature.

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

https://doi.org/10.1016/j.physd.2025.134880
Received 1 February 2025; Received in revised form 23 July 2025; Accepted 4 August 2025
Available online 12 August 2025

The lack of long-term prediction power in most complex systems, makes it important to quantify complexity through specific dynamical features of the data series, such as the embedding dimension (fundamental to interpret the data series as an observable of a dynamical system [41–45]), the possibly fractal dimension of the system's attractor in the embedding space of delayed observations, and the so-called Lyapunov exponents that quantify chaoticity [46,47]. The use of ML to extract these relevant dynamical features still relies on traditional algorithms, which are all based on the knowledge of a differentiable model of the system dynamics, to which linearization or higher-order techniques are applied (see, e.g., ML applications to compute Lyapunov exponents [48–50]).

Experimenting end-to-end learning approaches to these tasks is a relatively new topic in systems dynamics and is the focus of our study (the only attempts concern embedding identification and dimensionality reduction [51–55]). As it is natural in the traditional model-based techniques in nonlinear analysis, we start from linear problems, because they provide first-order approximations of more complex nonlinear ones.

ML approaches have already shown promises in solving eigenvalue and eigenfunction related problems, such as inverse eigenvalue problems [56], where a matrix is reconstructed from prescribed spectral data. The approach was however not fully end-to-end, as supervised learning was used jointly with specific pre- and post-processing methods. Another work in this direction employs a deep learning model to discover representations of Koopman eigenfunctions from time series data [55]. Using the eigenfunctions of the Koopman operator as a base of the functional space of the system's observables, it is possible to obtain a linear global approximation of a nonlinear dynamics, in the space of the leading eigen-coordinates.

To generate complex dynamics that are highly faithful to the original dynamics, creating models that imitate or extract features from dynamics considered simpler, such as linear systems, is essential. Thus, identifying eigenvalues is essential to reproduce their dynamics. There are analytical solutions to estimate eigenvalues, using linear algebra techniques, and numerical solutions, such as the autoregressive method and the Jacobi method, but they have limitations due to human dependence or high computational cost. Therefore, the use of ML models is proven to be a powerful tool for this purpose.

In this work we test a fully end-to-end ML approach on a simple, yet fundamental eigenvalue problem: the estimation of the eigenvalues of the state-transition matrix of a discrete-time linear time-invariant dynamical system from the time series of a linear observable. There are standard and efficient techniques, based on linear algebra, to compute the eigenvalues from sufficient samples (at least as many noise-free samples as the dimension of the state-transition matrix, under standard controllability and observability conditions), the most standard consisting of a linear (auto-)regression (AR) model identification, followed by root-solving for the model's characteristic polynomial's [57]. The point of trying a ML end-to-end approach is therefore not about accuracy or performance, but rather about the conceptual challenge for a NN architecture to learn a mapping from input, the observable time series, to output, the eigenvalues, which traditionally requires a human thought algorithm. This work has hence to be considered as a benchmark test, to open the way for end-to-end AI approaches to work on more complex time series, like those produced by nonlinear and time-variant processes and, in general, to work with real datasets.

We train our NN model on noise-free artificial data, generated from random systems of dimension 2-to −5. This allows us to know the exact values for eigenvalues of the state-transition matrices, that we use as target values for the training. We also use artificial data for testing the NN model, where the known eigenvalues are used as the actual values to be compared with the estimates provided by the NN and AR models. The estimates are obtained from both noise-free and noisy data, to assess the effect of the measurement noise of real datasets.

Our NN model demonstrates the ability to estimate the eigenvalues with high accuracy and robustness to measurement noise for systems with dimensions between 2 and 5. Our analysis shows that the dominant eigenvalue of the state-transition matrix, which defines the main system's dynamics, is accurately estimated, whereas the accuracy on non-dominant eigenvalues is generally higher for complex conjugates pairs. We interestingly find that the eigenvalue estimates provided by our end-to-end NN are more robust to measurement noise than the traditional AR-identification algorithm, thus elevating our NN model as a valid alternative to traditional algorithms in noisy environments.

To approach our task, we use Recurrent NNs (RNNs), a neural architecture specifically designed to process sequential data [58]. Specifically, we adopt Long Short-Term Memory (LSTM) cells [59], due to their ability to selectively retain important information on the dynamic process for a long period and avoid the vanishing gradient problem [60–64].

The reminder of the paper is organized into three sections. In the Methods Section 2, we present the generation of the dataset (noise-free data and noisy data), the traditional AR-identification algorithm, and our AI approach, including the NN architecture that we propose. The assessment of the AI approach (and the traditional AR algorithm) for noise-free and noisy data are presented and discussed in the Result Section 3. Section 4 further discusses our findings and concludes on their impacts on the future development of end-to-end AI approaches to nonlinear time series analysis.

2. Methods

2.1. Dataset generation

The dataset used for the learning task is generated by simulating N discrete-time linear time-invariant (LTI) autonomous asymptotically stable systems of the form:

$$\begin{aligned} x(t+1) &= A \cdot x(t) \\ y(t) &= C \cdot x(t), \end{aligned} \tag{1}$$

where $x(t)$ is the n-dimensional system's state at time t ($n \times 1$ vector), $y(t)$ is the system's output variable (the scalar state observable), A is the state-transition matrix ($n \times n$), and C is the row vector ($1 \times n$) defining the output transformation. The systems' dimension varies from 2 to $n_{\max} = 5$, with a higher frequency for larger dimensions, as larger systems require more training.

The simulations are the input data I for the NN, whereas the eigenvalues of the corresponding state-transition matrices are the target (true) output data O. More precisely, the dataset is composed of N pairs (I, O), where I is the sequence $y(t)$, $t = 1, \dots, T$, of the system's observable ($T = 100$ is used for all systems in the current implementation), whereas O ($2n_{\max}$ elements) is the list of real and imaginary parts of the n eigenvalues of A, followed by zero elements. All eigenvalues have modulus smaller than one (because of the asymptotic stability of A) and are sorted in decreasing order of magnitude, starting from the real and imaginary parts of the one with the largest modulus (the dominant one), followed by the real and imaginary parts or the others (sub-dominant).

We generate each pair (I, O) using Algorithm 1 (see Fig. 1 for a graphical representation). We randomly extract the eigenvalues, rather than directly extracting the elements of matrix A. This allows us to uniformly distribute the complex eigenvalues in the unit disk of the complex plane, which is a desirable property for the training dataset. We set a probability δ_c to extract a pair of complex conjugate eigenvalues, otherwise a real eigenvalue is extracted. We use $\delta_c = 0.9$, which gives a frequency of real eigenvalues in line with the element-wise extraction of the elements of matrix A. Note that to achieve a uniform distribution of the complex pairs in the unit disk, we extract the modulus ρ according to a square-root distribution (we extract ρ^2

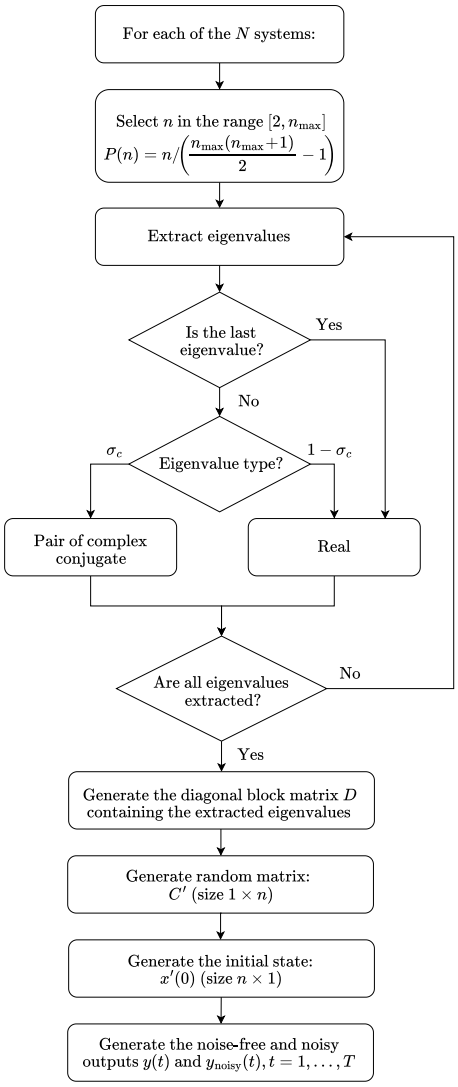

Fig. 1. Data generation flowchart: N is the dataset size; N systems of the form (2) are randomly extracted, each with dimension n from 2 to $n_{\max} = 5$. For each system, a time series of $T = 100$ time steps is generated from a random initial condition.

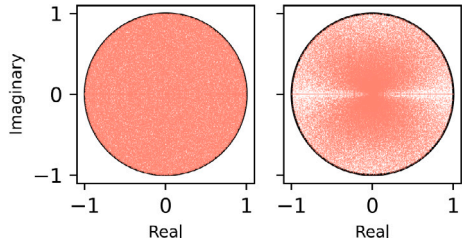

Fig. 2. Eigenvalues distribution for data generation. Left: extraction following Alg. 1 (square-root extraction of the modulus ρ for complex pairs); $N = 50000$, $n_{\max} = 5$, fraction of real eig's $= 0, 19$. Right: element-wise extraction of the elements of matrix A from $\mathcal{U}(-1, 1)$; same number N and matrix dimensions used in the left panel, fraction of real eig's $= 0, 20$ (unstable matrices are made stable by scaling the elements by $\rho/\rho_{\max}$, where rho is extracted as in Algorithm 1 and $\rho_{\max}$ is the dominant eigenvalue before scaling).

uniformly in $(0, 1)$), see Fig. 2 for a comparison with the element-wise extraction of the elements of matrix A.

To generate a time series of system (1) for the component I of the dataset corresponding to the eigenvalues extracted in O, we do not need the matrix A. Assuming that all extracted eigenvalues are simple (i.e., all different), we can imagine using the eigencoordinates $x'(t)$, for which the system's dynamics is written as

$$\begin{aligned} x'(t+1) &= D \cdot x'(t) \\ y(t) &= C' \cdot x'(t), \end{aligned} \tag{2}$$

where the state-transition matrix D has the following block diagonal structure: a 1×1 block e_i for each real eigenvalue e_i; a 2×2 block $\rho \cdot \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$ for each complex pair $e_{i,i+1} = \rho \cdot \exp(\pm\theta \cdot j)$. We therefore only need to randomly extract vectors C' and $x'(0)$ and simulate system (2).

As already noted, the elements of the target output O from $2n + 1$ to $2n_{\max}$ are set to zero to force the network learning $n_{\max} - n$ spurious null eigenvalues. This means that our NN cannot distinguish true zero eigenvalues from spurious ones. However, zero eigenvalues characterize non-reversible systems and, specifically, "finite memory" system's modes that disappear in one time step (or a few, if multiple), so modes that are in any case poorly visible on output data. We therefore neglect non-reversible systems.

Algorithm 1 Dataset generation

$O \leftarrow 0$ # *Initialize the* $2n_{\max}$ *elements of* O *to 0*
randomly select n between 2 and $n_{\max}$ with $P(n) = n/\left(\frac{n_{\max}(n_{\max}+1)}{2} - 1\right)$
$i \leftarrow 1$
while $i \leq n - 1$ **do**
  randomly extract $p \sim \mathcal{U}(0, 1)$
  if $p < \delta_c$ **then** # *Pair of complex conjugates*
    randomly extract $\rho^2 \sim \mathcal{U}(0, 1)$ # *This gives complex pairs uniformly distributed in the unit disk*
    randomly extract $\theta \sim \mathcal{U}(0, \pi)$
    define two complex eigenvalues: $e_i = \rho \cdot \exp(\theta \cdot j)$ and $e_{i+1} = \rho \cdot \exp(-\theta \cdot j)$
    $O_{2(i-1)+1} = \text{Re}(e_i)$; $O_{2(i-1)+2} = \text{Im}(e_i)$
    $O_{2(i-1)+3} = \text{Re}(e_{i+1})$; $O_{2(i-1)+4} = \text{Im}(e_{i+1})$
    $i \leftarrow i + 2$ # *Skip both eigenvalues*
  else # *Real eigenvalue*
    randomly extract $e_i \sim \mathcal{U}(-1, 1)$
    $O_{2(i-1)+1} = e_i$; $O_{2(i-1)+2} = 0$
    $i \leftarrow i + 1$
  end if
end while
if $i = n$ **then** # *Last unpaired eigenvalue is real*
  randomly extract $e_i \sim \mathcal{U}(-1, 1)$
  $O_{2(i-1)+1} = e_i$; $O_{2(i-1)+2} = 0$
end if
define the eigenvalues block diagonal matrix D (1×1 block e_i for each real eigenvalue; a 2×2 block $\rho \cdot \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$ for each complex pair)
randomly extract vector C' ($1 \times n$) element-wise from $\mathcal{U}(-1, 1)$
randomly extract $x'(0)$ element-wise from $\mathcal{U}(-1, 1)$
Simulate the system (2) and produce the time series $y(t)$, $t = 1, \ldots, T$

To consider the effect of measurement noise on the input I, we perturb each sample $y(t)$ by a factor that is different from one by a random value sampled from a normal distribution with zero mean and standard deviation equal to a control parameter σ:

$$y_{\text{noisy}}(t) = y(t)(1 + \mathcal{N}(0, \sigma^2)) \tag{3}$$

The target outputs O are not perturbed because we want to test the robustness of the eigenvalue estimator algorithm to measurement noise.

Finally, note that, the eigenvalues estimation task from the system's output $y(t)$ is feasible under standard controllability and observability conditions, that are fulfilled with probability one by our algorithm ($(D, x'(0))$ must be a controllable pair and (D, C') an observable one in system (2), assuring that all system's modes are visible in the state observable defined by C').

2.2. AR approach

The standard approach to address eigenvalues estimation from the time series of a system's observable involves identifying an auto-regressive (AR) model, followed by root-solving for the characteristic

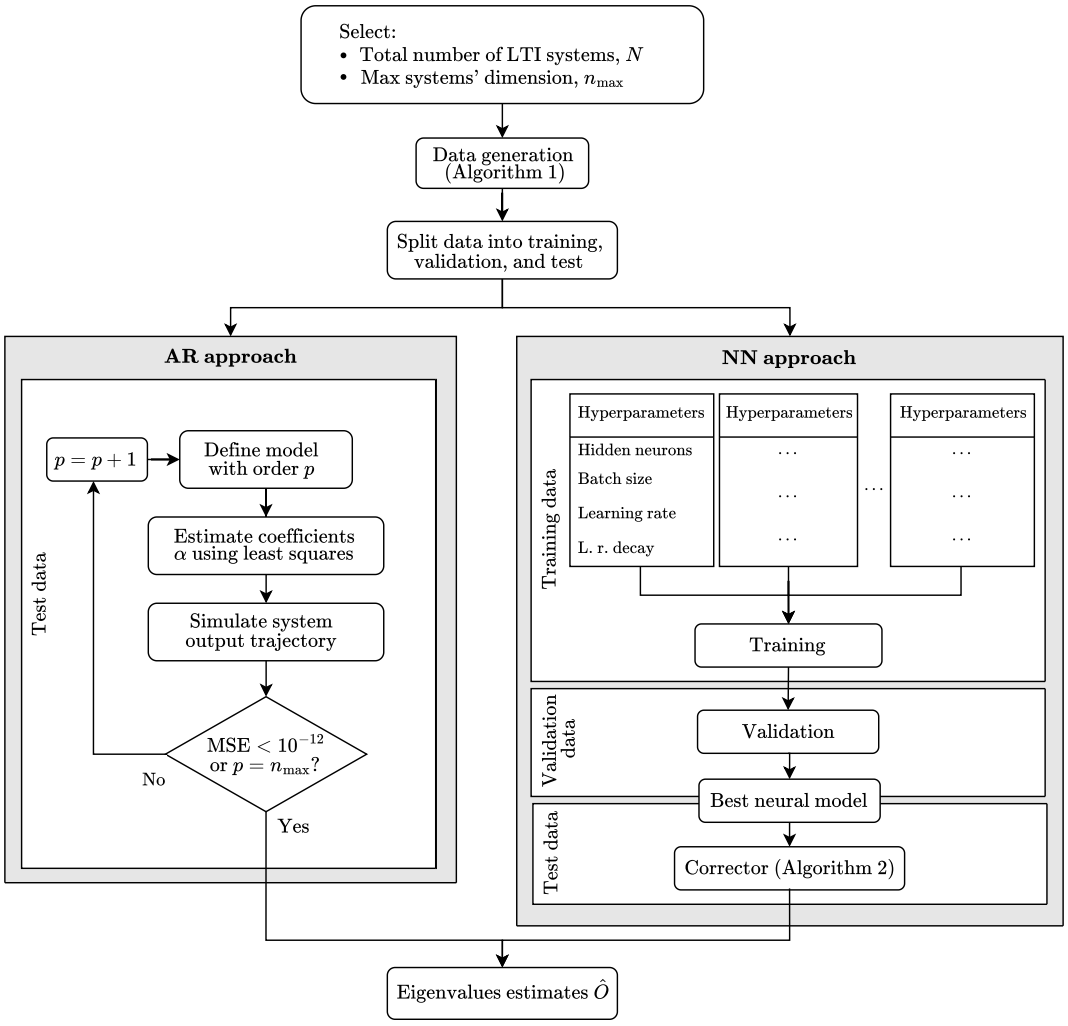

Fig. 3. Flowchart for eigenvalue estimation using the NN and AR approaches.

polynomial of the identified model. This approach represents the state-of-the-art for solving this problem [65,66], with perfect results (apart from numerical precision) on noise-free data. The results of the AR-identification algorithm on noisy data will therefore serve as a benchmark to assess the effectiveness of the AI approach and its robustness to measurement noise.

An AR model describes a process by considering the linear dependencies between the present observation and a set of previous observations, enabling the prediction of the future behavior of a time series based on historical patterns. The mathematical formulation of an AR model of order p is as follows:

$$y(t) = \alpha_1 y(t-1) + \alpha_2 y(t-2) + \cdots + \alpha_p y(t-p) + \epsilon(t), \tag{4}$$

where α_i, $i = 1' \ldots, p$ are the model coefficients, and $\epsilon(t)$ is an error term due to model mismatch and/or measurement noise.

Parameter identification for the α_i's is done using the least squares method, which minimizes the sum of squared errors between observations and corresponding model's predictions, resulting in the estimate:

$$\hat{\alpha} = (M^T M)^{-1} M^T Y, \tag{5}$$

where

$$\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{bmatrix}, \quad M = \begin{bmatrix} \bar{y}_p & \bar{y}_{p-1} & \cdots & \bar{y}_1 \\ \bar{y}_{p+1} & \bar{y}_p & \cdots & \bar{y}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \bar{y}_{T-1} & \bar{y}_{T-2} & \cdots & \bar{y}_{T-p} \end{bmatrix}, \quad Y = \begin{bmatrix} \bar{y}_{p+1} \\ \bar{y}_{p+2} \\ \vdots \\ \bar{y}_T \end{bmatrix}.$$

Using the obtained coefficients $\hat{\alpha}$, we can write the system's characteristic polynomial:

$$\lambda^p - \hat{\alpha}_1 \lambda^{p-1} - \hat{\alpha}_2 \lambda^{p-2} - \cdots - \hat{\alpha}_p, \tag{6}$$

which corresponds to the denominator of the system's transfer function, the roots of which are the system's eigenvalues (under the discussed controllability and observability conditions).

This procedure gives the exact eigenvalues from noise-free data only if the dimension n of the system is known and $p = n$ is used (and the number T of data is equal to or larger than n). If $p > n$ is used, multiple solutions exist, so that there is no guarantee of obtaining the desired one with $p - n$ zero eigenvalues (the one with $\alpha_i = 0$ for $i > n$). To estimate the system's dimension n, which is a priori unknown, we set a threshold on the mean squared error (MSE) and we increase the order p until the MSE goes below the threshold. This gives exact results on noise-free data, but can lead to an overestimation of the model's order in the presence of noise. In the latter case, the workflow for comparing the eigenvalues estimates by the NN and AR approaches is summarized in Fig. 3.

2.3. NN approach

We address the eigenvalues estimation from the time series of a system's observable using RNNs, neural models specifically designed to learn sequential tasks, such as time series analysis. Thanks to the fact that their weights and biases are shared across the time steps

(parameter sharing), RNNs can process sequences of arbitrary length with a fixed number of parameters [33], a remarkable advantage compared to fully connected architectures, where the number of parameters increases linearly with the length of the sequence [67,68].

We use LSTM cells, specific recurrent neurons with two internal states, called hidden and cell state. The first is the same as the state of a standard recurrent neuron and is passed to the following layers and time steps, the second (peculiar to the LSTM cell) is responsible for tracking the long-term dynamics of the input and contributes to the update of the hidden state. LSTM cells have three gates, namely the forget, input, and output gates, that modulate the information flows through a sigmoid function that returns values between 0 (closed gate) and 1 (open gate). The forget gate resets the cell state if the combination of input and hidden states into the sigmoid is low; the input gate defines the extent to which the new input affects the update of the cell state, protecting it from irrelevant contributions; the output gate modulates the update of the hidden state. This gated structure can retain relevant information on the process dynamics for a long duration and avoids vanishing (or exploding) gradients that generally affect the convergence of the training in standard RNNs. The internal states are initialized at zero at the beginning of each sequence allowing the parallelization of the training procedure [69].

A first LSTM layer is composed of several cells (200, see below for the definition of the hyperparameters of the NN architecture) that work in parallel, taking the time series $y(t)$ in input and updating their internal states. In an optional second layer, the LSTM cells take as input a combination of the hidden states of the first-layer cells at time t and correspondingly update their internal states. The hidden states of the last-layer cells at the last time step $t = T$ contain an encoding of the time series. This information is passed to two dense layers, and finally to the output layer that produces the eigenvalues' estimates $\tilde{O}$.

These estimates are then post-processed by a corrector algorithm (Algorithm 2) to identify the dimension n of the system that generated the time series $y(t)$ and to rectify unavoidable numerical inconsistencies due to the fact that there is no constraint in the NN that forces the estimated eigenvalues to be either real or in complex pairs. Since the network has been trained with zero target values for the eigenvalues in position from $n+1$ to $n_{\max}$, the dimension n is identified by setting at zero the eigenvalues that result too small in magnitude (see threshold δ_1 in Algorithm 2), that are most likely spurious. Non-spurious eigenvalues that are close to being complex conjugate (see threshold δ_2) are set so, otherwise they are set real by disregarding their estimated imaginary part. This procedure gives the final estimates $\hat{O}$. We use the thresholds values $\delta_1 = 0.05$ and $\delta_2 = 0.1$.

Following the standard ML practice, we split the data into training data (70%), used for the calibration of the NN's parameters (weights and biases), validation data (15%), used to tune the hyperparameters that define the neural architecture as well as the configuration of the learning algorithm, and test data (15%), which are reserved for the assessment of the performance of the trained NN.

We train the neural model to minimize the MSE between actual (O) and estimated ($\tilde{O}$) eigenvalues (note that the same metric was used for estimating the parameters of the AR model, but in that case, the error was calculated between the actual and estimated samples of the time series). The gradients of the MSE with respect to the NN's parameters are computed through the back-propagation through time algorithm [47,70], a version of the standard back-propagation modified to work for recurrent architectures. The gradients are then used by an optimizer to update the NN's parameters. We used Adam (adaptive moment estimation) optimizer [71] to this aim. Adam is a state-of-the-art algorithm for deep learning derived from the standard stochastic gradient descent algorithm. It adaptively estimates the first- and second-order moments of the gradients using running averages with standard values of the decay rates ($\beta_1 = 0.9$ and $\beta_2 = 0.999$ for the first and second moments, respectively). The estimates of the two moments are then used to evolve the learning rate during training, by means of a tunable decay factor (see below for the definition of the hyperparameters of the NN architecture).

Algorithm 2 Eigenvalues corrector

$n \leftarrow n_{\max}$ *# Dimension estimate*
$\mathrm{Re}(e_i) \leftarrow \tilde{O}_{2(i-1)+1}$, $\mathrm{Im}(e_i) \leftarrow \tilde{O}_{2(i-1)+2}$, $i = 1, \dots, n_{\max}$
$i \leftarrow 1$
while $i \leq n_{\max} - 1$ **do**
  if $|\tilde{e}_i| < \delta_1$ **then** *# Spurious eigenvalue*
    $\hat{e}_i \leftarrow 0$
    $n \leftarrow n - 1$
    $i \leftarrow i + 1$
  else if $|\mathrm{Im}(\tilde{e}_i)| > \delta_1$ **and** $\left|\mathrm{Im}(\tilde{e}_i) + \mathrm{Im}(\tilde{e}_{i+1})\right| < \delta_2$ **and** $\left|\mathrm{Re}(\tilde{e}_i) - \mathrm{Re}(\tilde{e}_{i+1})\right| < \delta_2$ **then** *# Complex conjugate pair*
    $\hat{O}_{2(i-1)+1} \leftarrow \left(\mathrm{Re}(\tilde{e}_i) + \mathrm{Re}(\tilde{e}_{i+1})\right)/2$
    $\hat{O}_{2(i-1)+3} \leftarrow \hat{O}_{2(i-1)+1}$
    $\hat{O}_{2(i-1)+2} \leftarrow \left(|\mathrm{Im}(\tilde{e}_i)| + |\mathrm{Im}(\tilde{e}_{i+1})|\right)/2$
    $\hat{O}_{2(i-1)+4} \leftarrow -\hat{O}_{2(i-1)+2}$
    $i \leftarrow i + 2$ *# Skip both eigenvalues*
  else *# Real eigenvalue*
    $\hat{O}_{2(i-1)+1} \leftarrow \mathrm{Re}(\tilde{e}_i)$
    $\hat{O}_{2(i-1)+2} \leftarrow 0$
    $i \leftarrow i + 1$
  end if
end while
if $i = n_{\max}$ **then** *# Last unpaired eigenvalue*
  if $|\tilde{e}_i| < \delta_1$ **then** *# Spurious eigenvalue*
    $\hat{e}_i \leftarrow 0$
    $n \leftarrow n - 1$
  else *# Real eigenvalue*
    $\hat{O}_{2(i-1)+1} \leftarrow \mathrm{Re}(\tilde{e}_i)$
    $\hat{O}_{2(i-1)+2} \leftarrow 0$
  end if
end if

Hyperparameter tuning is performed following the grid search approach, i.e., considering all the possible combinations of the following values: neurons per hidden layer (50, 200), batch size (500, 1000, 4000), learning rate (0.01, 0.001), and learning rate decay factor (0.001, 0.0001). For each combination, We perform the training for a maximum of 1000 epochs. The MSE is monitored during the learning process so that we can stop it if there is no improvement for 200 consecutive epochs (early stopping [72]). During each epoch, the whole training dataset is processed by splitting it into subsets named mini-batches (the parameters are thus updated based on the gradient computed for each specific mini-batch). The negative impact of unfortunate initializations of the NN parameters is limited by repeating the training twice with different random seeds for each combination of the hyperparameters. The combination ensuring the lowest MSE value computed on the validation dataset is then selected (200 neurons per layer, batch size equal to 1000, learning rate to 0.001, and decay factor to 0.0001).

The performance of the selected trained NN is finally measured on the test dataset. We measure performance through the R^2-score, the standard way to normalize the MSE between the eigenvalues estimates $\tilde{O}$ and the true values O ($R^2 = 1 - \mathrm{MSE}/\mathrm{var}(O)$; $R^2 = 1$ corresponds to MSE = 0, i.e., perfect estimations; $R^2 = 0$ means the same performance of estimating all the components of O with the average value, that is a quite unsatisfactory performance; $R^2 < 1$ corresponds to MSE larger than the variance of O).

3. Results

3.1. Noise-free data

We first present the results of the noise-free case (the AR approach is not considered in this case, since it provides exact results, as explained

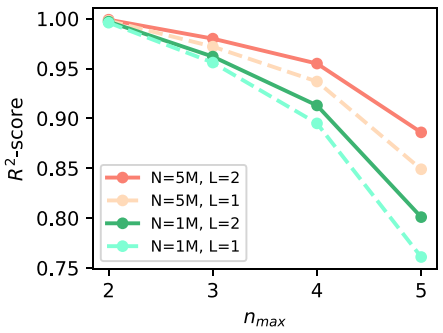

Fig. 4. Accuracy (R^2-score) evaluated on the test fraction of two different datasets of size $N = 1$M and $N = 5$M, using $L = 1$ or $L = 2$ LSTM layers, for increasing maximum dimension for the LTI systems in the datasets ($n_{\max}$ from 2 to 5).

in Section 2.2). Fig. 4 reports the sensitivity of the NN performance with respect to the dataset dimensionality (N equal to 1 or 5 million) and the number of LSTM layers (L equal to 1 or 2), separately for $n_{\max} \in \{2, 3, 4, 5\}$.

The R^2-score decreases significantly as $n_{\max}$ increases. In the best case ($N = 5$M and $L = 2$), the R^2-score decreases from around 1 for $n_{\max} = 2$ (almost perfect estimates) to around 0.87 for $n_{\max} = 5$, because the number of possible eigenvalues configurations increases with $n_{\max}$ and make the task more complex. When $n_{\max} = 2$, only two configurations are possible (eigenvalues can be both real or in a pair of complex conjugates); $n_{\max} = 5$ is associated with a wide range of possible cases ($n = 2$ is just one of them) and configurations. For instance, if $n = 5$, we can have 5 real eigenvalues, 3 real and a pair of complex conjugates, or only one real and two pairs of complex conjugate eigenvalues, with subcases depending on whether the dominant eigenvalue (the one with the largest modulus) is real or complex.

Fig. 4 also shows that more data (N) and more hidden layers (L) in the NN architecture both improve the performance, except for $n_{\max} = 2$, where the performance is already top with $N = 1$M and $L = 1$. However, as the $n_{\max}$ increase, we see that the R^2-score decreases less in the ($N = 5$M, $L = 2$)-case, than in the other curves. This behavior was expected since a larger dataset offers more information to the learning process, and more LSTM layers empower the NN with more learning capabilities.

We also estimate the dimension n of the LTI system that generated the input series (see the eigenvalues corrector threshold Algorithm 2), even though our neural network is not specifically trained to learn this feature, which is essentially categorical. Nonetheless, the accuracy in the estimation of n, which is based on a threshold on the eigenvalues magnitude ($\delta_1 = 0.05$ in Algorithm 2), under which eigenvalues are considered spurious, follows a trend similar to the R^2-score shown in Fig. 4: 97.1% for $n_{\max} = 2$; 85.6% for $n_{\max} = 3$; 77.5% for $n_{\max} = 4$; 61.9% for $n_{\max} = 5$ (accuracies for the case $N = 5$M and $L = 2$).

In order to provide a feeling of the goodness of the NN estimates, we graphically compare in Fig. 5 the positions in the complex plane of the actual and estimated eigenvalues and the corresponding time evolution of the system's observable $y(t)$. These results are relative to the NN with $L = 2$ LSTM layers trained on the dataset of size $N = 5$M for systems with $n_{\max} = 4$, a case for which we obtained a R^2-score equal to 0.96. We selected the eigenvalues configurations to cover all the possible combinations for a state-transition matrix of order 4, 3, and 2. For the matrices of order 2: 2 real eigenvalues (case a); a pair of complex conjugates (case b). For matrices of order 3: 3 real eigenvalues (cases c, with positive dominant, and d, with negative dominant); one real and a pair of complex conjugates (cases e, with real dominant, and f, with complex dominants). For matrices of order 4: 4 real eigenvalues (cases g, with positive dominant, and h, with negative dominant); 2 reals and a pair of complex conjugates (cases i, with complex dominants and positive sub-dominant, j, with complex dominants and negative sub-dominant, and k, with real dominant); 2 pairs of complex conjugates (case l).

The estimated eigenvalues (orange diamonds) nearly overlap with the actual eigenvalues (green squares) in the majority of the cases, indicating the high accuracy of the NN's estimates (see all the complex planes in Fig. 5). The dominant eigenvalues (the real of complex pair with largest modulus) are always estimated with very high accuracy (the R^2-score restricted to the dominant eigenvalues is 0.99). Non-dominant complex eigenvalues are also estimated accurately (see, e.g., cases e, k, and l; the R^2-score restricted to non-dominant complex pairs is 0.94), while non-dominant real eigenvalues are sometimes miscalculated (see cases a, d, f-h; see also cases c, i, and j where the sub-dominant real eigenvalue is well estimated; the R^2-score restricted to the real sub-dominant eigenvalues is 0.59). A possible explanation for this behavior is that the contribution to the output time series generated by complex pair of eigenvalues is more easily identifiable, compared to the contribution of a real eigenvalue, because of the oscillations of complex modes. Conversely, the modes of real eigenvalues vanish exponentially, so that, if non-dominant, their contribution to the output signal might get masked by other modes.

The two time series shown in each panel next to a complex plane (panels in the second and fourth columns of Fig. 5) are one representative of the actual eigenvalues (green continuous line) and the other representative of the estimated ones (orange dashed line). More specifically, the first is obtained by simulating system (2) with the actual eigenvalue matrix D and random vectors C' and $x'(0)$, whereas the second is obtained with the matrix $\hat{D}$ of the estimated eigenvalues and using the same C' and $x'(0)$. The general comment is that the estimation errors on the system's eigenvalues have moderate impacts on the output time series of the system. This highlights how difficult it is to discern small dissimilarities in terms of different eigenvalues. Even in the critical case of an inaccurate estimation of a non-dominant real eigenvalue, like, e.g., case h, the differences in the two representative time series are quite small. This is a key point of our results. Since the neural model only takes as input the time series of $y(t)$, we have to accept unavoidable estimation errors on the eigenvalues that produce negligible differences in the representative time series of the system's output.

3.2. Noisy data

We explore the robustness against measurement noise of our NN and we compare it against the benchmark results obtained with the AR algorithm described in Section 2.2. We use the NN model with $L = 2$ LSTM layers trained on the noise-free dataset of size $N = 5$M and we test it on the noisy time series produced with Algorithm 1 and Eq. (3), varying the noise standard deviation σ from $\sigma = 10^{-7}$ to $\sigma = 10^{-2}$.

As expected, the AR approach ensures a very high accuracy when the noise level is low (it is an exact algorithm on noise-free data). However, for increasing noise intensity, the NN approach turns out to be more robust. The R^2-score curves in Fig. 6 show that the NN model has a consistent pattern for all the considered dimensions, from 2 to $n_{\max}$, of the LTI systems in the test dataset. The R^2 remains very close to the noise-free value up to a critical noise intensity (that decreases with $n_{\max}$), above which the NN performance drops quite sharply. In contrast, the R^2-score obtained with the AR algorithm decreases more linearly with the intensity of noise, eventually offering a lower performance with respect to the NN. For systems of sizes up to $n_{\max} = 4$ and 5, this happens starting from very low noise intensity, with standard deviation σ around 10^{-6}. For higher intensities, when the NN performance drops down, the AR approach is better, but at levels of accuracy non very interesting ($R^2 = 0.6$ for $n_{\max} = 2$ and R^2 below 0.3 for higher dimensional systems).

As we did for the noise-free case, we compare the noisy estimates with the actual eigenvalues in the same 12 cases selected in Fig. 5

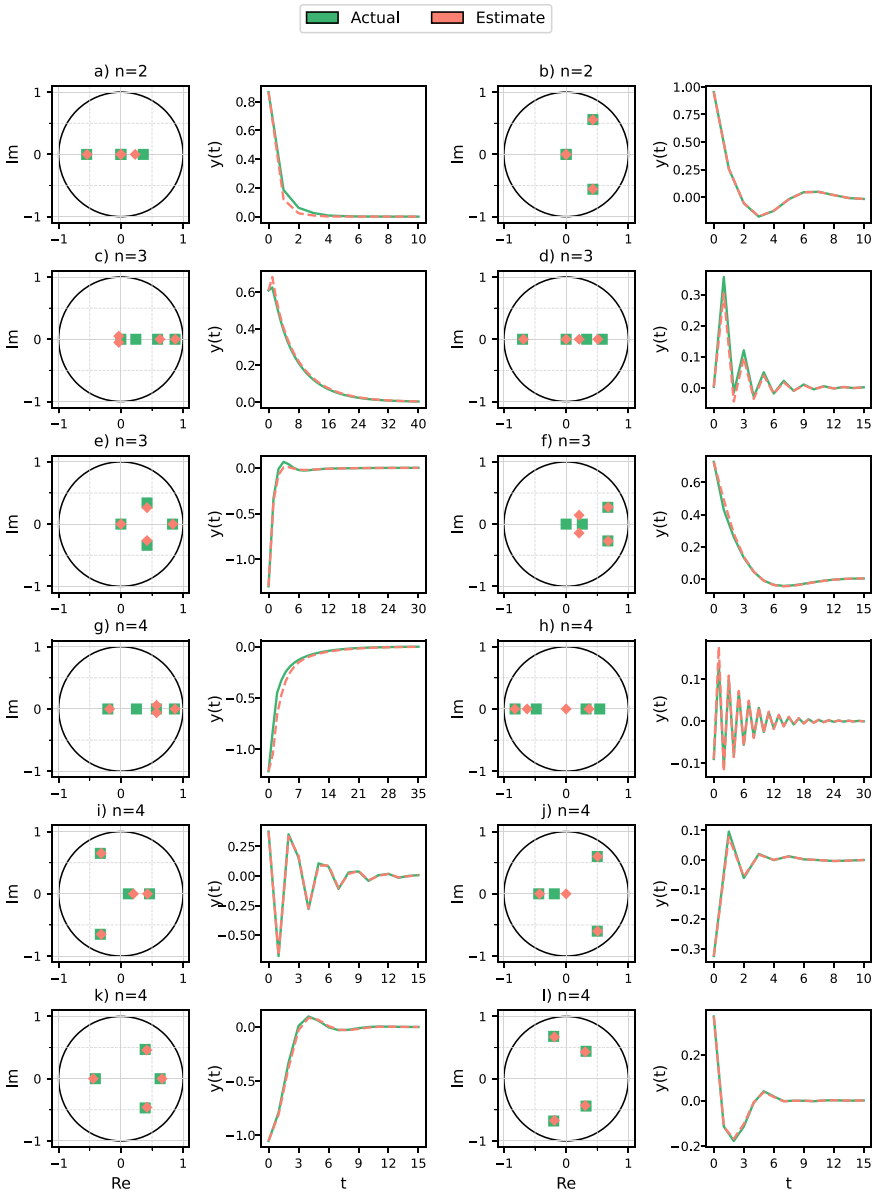

Fig. 5. Actual (green squares) and estimated (orange diamonds) eigenvalues in the complex plane for $n_{\max} = 4$ and $N = 5$M. Time series of the systems' observable $y(t)$ generated by the actual (green continuous line) and estimated (orange dashed line) eigenvalues (the time span is adjusted to the system's time response).

(which are taken from the dataset with $n_{\max} = 4$ and $N = 5$M). The comparison is shown in Fig. 7, which has the same structure of Fig. 5 (green squares for the actual eigenvalues, orange diamonds for the noise-free estimates, shown for comparison, and red diamonds for the noisy estimates), and is obtained for noise standard deviation $\sigma = 10^{-3}$.

Despite the obvious degradation of performance, as measurement noise reduces the correlation patterns present in the noise-free time series, making the eigenvalues estimation task more difficult, the accuracy on the dominant eigenvalues remains very good in all cases. The worsening is especially visible on real non-dominant eigenvalues, and even, in some cases like e, on sub-dominant complex pairs.

Despite the non-negligible estimation errors on the systems' eigenvalues in noisy data, the dynamics of the representative time series generated with the noisy estimates of the eigenvalues are consistent with the one generated with the actual values (see the second and fourth columns of Fig. 7).

4. Conclusion

We explored the possibility of using ML tools to extract quantitative features representative of a time series and of the underlying dynamical system that generated the series. The peculiarity of the proposed approach is that we set up an end-to-end AI that directly computes the features of interest from the time series of a system's observable, without relying on any classical algorithm from dynamical systems theory or a priori knowledge of the system. This represents a considerable paradigm shift in the context of ML tools applied to dynamical systems, which are usually employed to solve time series prediction or system identification tasks, on top of which classical algorithms then compute the features of interest. Contrary to this classical approach, in which the ML tool is essentially required to reproduce a mapping between past and future values of the system's variables, our end-to-end approach requires the AI to learn the entire algorithm, from the system's observable to the features of interest.

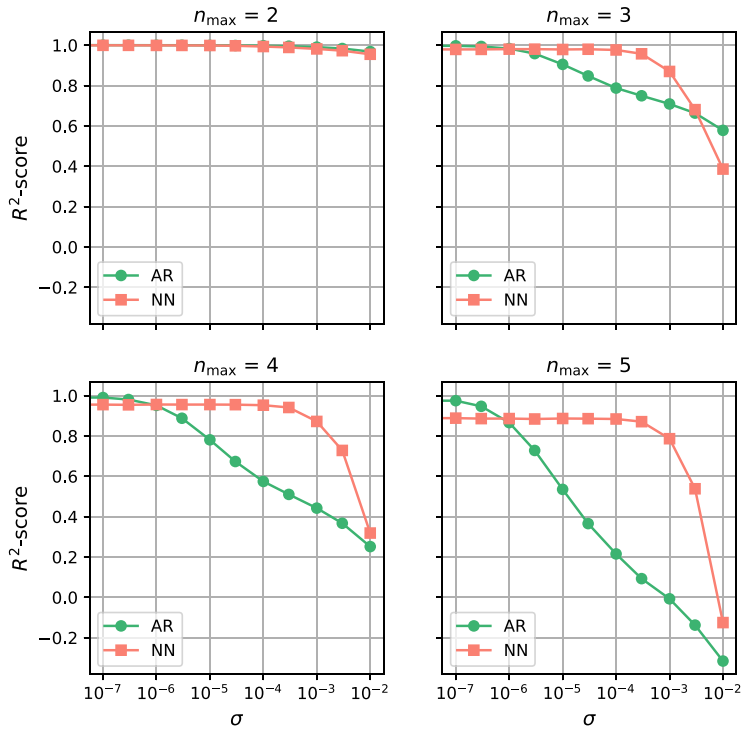

Fig. 6. Accuracy (R^2-score) evaluated on the test fraction of the dataset of size $N = 5$M, using $L = 2$ LSTM layers, for increasing maximum dimension for the LTI systems in the dataset (n_{max} from 2 to 5) and increasing intensity of measurement noise (noise standard deviation σ).

As a first step towards the development of end-to-end AI tools for time series and dynamical systems analysis, we considered a basic eigenvalue problem, i.e., the problem of estimating the eigenvalues of the state-transition matrix of asymptotically stable LTI systems (in discrete-time). We adopted recurrent NN with LSTM cells, which are specifically designed for sequential data, such as time series. In a first experiment, we used the NN to estimate the eigenvalues of noise-free time series generated by LTI systems with dimension from 2 to 5. In addition to the system dimensionality, we analyzed the sensitivity of the estimation against the size of the dataset and the number of recurrent layers in the NN architecture. In a second experiment, we tested the robustness of the model against different levels of measurement noise, comparing it with the standard AR algorithm for linear model identification, which is the benchmark for this task. We analyzed the performance of the eigenvalue estimates in noisy scenarios, comparing them both with the noise-free estimate and with the actual eigenvalues.

Our results for the noise-free data show that the proposed approach can accurately estimate the eigenvalues of the state-transition matrix, provided that enough data and neural elements (LSTM layers in the NN) are provided for the learning task. Specifically, high precision is reached on the dominant eigenvalues (largest magnitude) and on complex conjugate sub-dominant ones, whereas less accuracy is obtained on sub-dominant real eigenvalues, the trace of which on the system's observable is often masked by the modes of the dominant and complex conjugate sub-dominant eigenvalues. We also provide an indication for the dimension n of the LTI system that generated the input series, even though our neural network is not specifically trained to learn this feature, which is of categorical nature (see a specific comment below).

The result of the eigenvalue estimation from noisy data showed that our NN presents a more stable accuracy for lower noise levels, compared to the traditional AR algorithm. For high noise intensity, the NN presents a threshold effect, with a sudden collapse in accuracy at a critical noise intensity (with a lower threshold, the higher the dimension of the LTI systems). In contrast, the AR algorithm presents a linear-like decrease in accuracy, which falls off the performance of the NN at noise standard deviation around 10^{-6} for LTI systems of order 4 and 5.

There are several ways to improve the preliminary results here presented, essentially grouped into two directions. One is to investigate which is the best neural architecture to solve the eigenvalue task. The other is about the impact of noisy data.

Along the first one, several architectures, in terms of the type of recurrent cells and mix of recurrent and fully connected layers, could be tested and scaled with the maximum allowed dimension for the LTI system in the dataset. Also, a more thorough analysis of the network hyperparameters, including the number of layers and number of cells per layer, and all the parameters controlling the learning procedure, could be carried out.

One more conceptual aspect to which we need to pay attention is the nature, numerical or categorical, of the features of interest. The system's dimension and the type, real or complex conjugate, of the eigenvalues are indeed categorical information, that could be inferred with neural architectures and loss functions typical of classification problems. Mixed numerical-categorical neural architectures are more complex to train and manage, so we here opted for numerical surrogates of the categorical information. For example, we estimate the imaginary part for all n eigenvalues, using a zero actual value for real eigenvalues and we do not force the network output to provide complex conjugate pairs. The results are then made consistent by the corrector Algorithm 2 reported in the Methods Section 2.3. The NN estimates the spurious eigenvalues (from $n+1$ to n_{max}) in the same way (in this case the target value for both real and imaginary parts are set to zero), allowing us to infer the system's dimension. All these quantities trained to be zero could be better inferred by a suitable classifier.

Along the second direction, the impact of noisy data on the training process should be investigated. In the present work we did not do it, because we imagine that the data for training are not measured on the field, but artificially generated by our Algorithm 1. However, this is not always the case for real applications. This is especially true for extending this work to nonlinear time series (observables of nonlinear dynamical systems), because for those systems, even if the

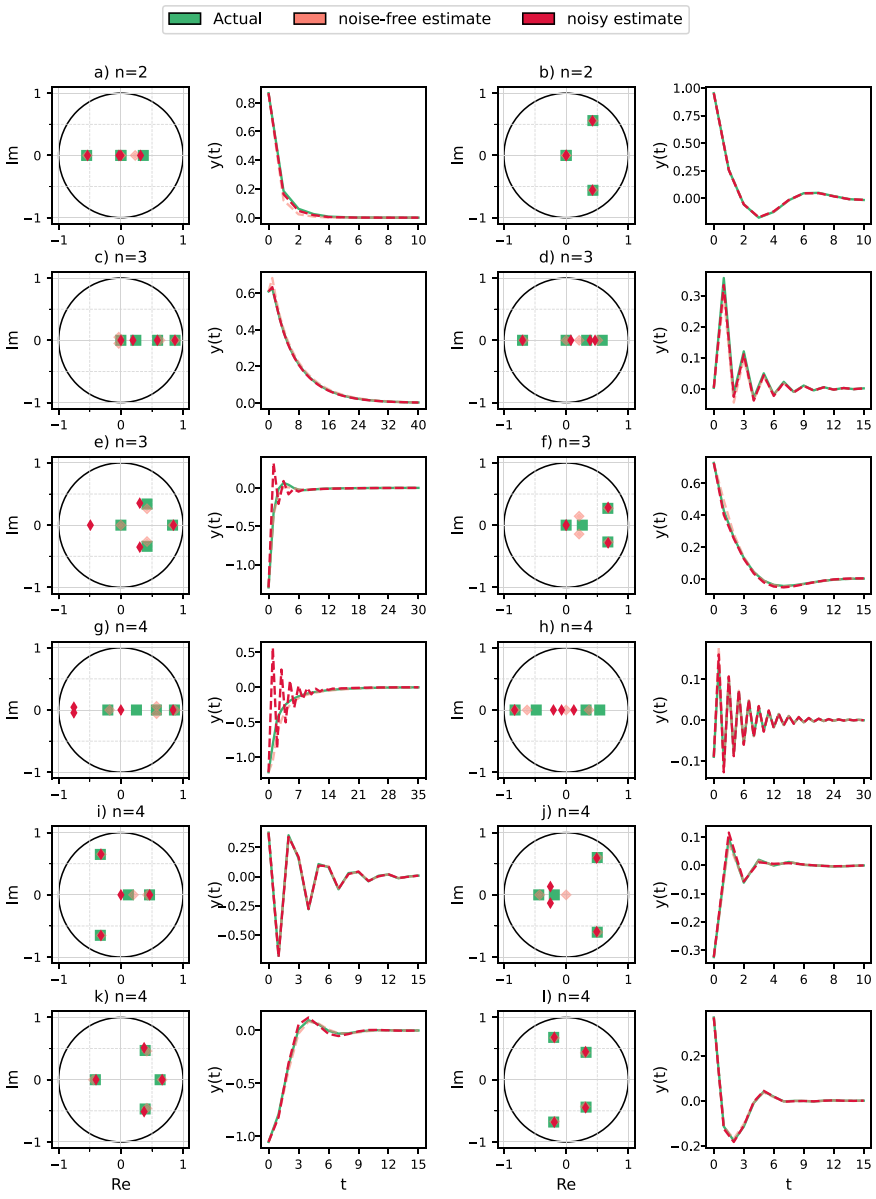

Fig. 7. Actual (green squares) and estimates from noisy data (red diamonds) eigenvalues in the complex plane for $n_{\max} = 4$ and $N = 5$M (noise standard deviation $\sigma = 10^{-3}$, NN $R^2 =$, AR $R^2 =$, see Fig. 6; noise-free estimates, the same orange diamonds from Fig. 5, are visible in background for comparison). Time series of the systems' observable $y(t)$ generated by the actual (green continuous line) and estimated (red and orange dashed lines) eigenvalues (the time span is adjusted to the system's time response).

system's equations are known, so state trajectories and the observable can be produced error-free, the target values for the feature of interest, like, e.g., the Lyapunov exponents, cannot be computed with extreme accuracy, being the results of approximated numerical algorithms.

Beyond the specific learning task we addressed here, the eigenvalue problem, our interest is in acquiring most of the experience from this basic task to then exploit it on end-to-end tasks for complex nonlinear systems. Actually, the state-of-the-art of system identification is very efficient and accurate for linear systems, so we could hardly hope to do better with an end-to-end AI approach. Nevertheless, we found interesting the robustness to measurement noise shown by our NN and, in any case, understanding the eigenvalue problem is a preparatory step for tackling more complex tasks in an end-to-end fashion.

One future research direction will hence be in the context of nonlinear dynamics and time series long-term features, like the embedding dimension, fundamental to reconstruct the space of the system's variable from a single observable, the fractal dimension of the system's attractor in the reconstructed space, and the Lyapunov exponents quantifying complexity (the stretching and folding forces within the attractor). An even more challenging task is the estimation of the full spectrum of Lyapunov exponents, whose partial sums correspond to the average exponential rates of expansion (or contraction) of k-dimensional volumes ($k \geq 2$). These quantities are relevant in various contexts, especially in mechanics and fluid dynamics, where they provide insight into stability, predictability, and transport properties of complex dynamics [73–76].

In the nonlinear context, there are several fields in which our LSTM framework could be successfully applied, to analyze and control dynamical processes. For example, the Lyapunov exponent could be used as a target function in a model predictive control scheme, in order to tame chaos and drive the process to a more regular periodic or stationary motion. One typical limitation is that the available data series are too short to compute the Lyaounov exponent with traditional algorithms (based on attractor reconstruction), and even generating synthetic data with the predictive model of the control scheme (that could be NN-based as well [49]) would not allow the computation in real-time in-the-loop. Interestingly, our NN model can be trained off-line on many long time series artificially generated from known

chaotic models and seems to provide a significant accurately to estimate the Lyapunov exponent from rather short time series (preliminary unpublished results).

Finally, beyond the theoretical relevance of the insights derived from this study, it is worth mentioning that the problem here considered could be used as a benchmark to evaluate the performance of novel ML tools (neural architectures, learning procedures, etc.). Indeed, the proposed task is scalable in terms of complexity and dimensionality (the dataset can be extended to higher orders of the LTI system). In the context of nonlinear systems, extensions of the problem to the above mentioned quantitative features would result in even more challenging benchmarks.

CRediT authorship contribution statement

Emanuele Salgarollo: Writing – original draft, Visualization, Software, Methodology, Formal analysis, Data curation. **João Valle:** Writing – review & editing, Writing – original draft, Visualization, Formal analysis. **Matteo Sangiorgio:** Writing – review & editing, Visualization, Supervision, Software, Methodology, Conceptualization. **Fabio Dercole:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Conceptualization.

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.

Data availability

Data will be made available on request.

References

[1] D. Gabor, Theory of communication. Part 1: The analysis of information, J. Inst. Electr. Eng.- Part III: Radio Commun. Eng. 93 (26) (1946) 429–441.

[2] I. Sobel, G. Feldman, et al., A 3x3 isotropic gradient operator for image processing, 1968, pp. 271–272, A talk at the Stanford Artificial Project in 1968.

[3] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. 25 (2012).

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, Ieee, 2009, pp. 248–255.

[5] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, et al., Deep speech 2: End-to-end speech recognition in english and mandarin, in: International Conference on Machine Learning, PMLR, 2016, pp. 173–182.

[6] I. Serban, A. Sordoni, Y. Bengio, A. Courville, J. Pineau, Building end-to-end dialogue systems using generative hierarchical neural network models, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30, (1) 2016.

[7] Z. Lin, P. Xu, G.I. Winata, F.B. Siddique, Z. Liu, J. Shin, P. Fung, Caire: An end-to-end empathetic chatbot, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, (09) 2020, pp. 13622–13623.

[8] X. Tan, J. Chen, H. Liu, J. Cong, C. Zhang, Y. Liu, X. Wang, Y. Leng, Y. Yi, L. He, et al., Naturalspeech: End-to-end text-to-speech synthesis with human-level quality, IEEE Trans. Pattern Anal. Mach. Intell. (2024).

[9] Y. Xiao, F. Codevilla, A. Gurram, O. Urfalioglu, A.M. López, Multimodal end-to-end autonomous driving, IEEE Trans. Intell. Transp. Syst. 23 (1) (2020) 537–547.

[10] A. Tampuu, T. Matiisen, M. Semikin, D. Fishman, N. Muhammad, A survey of end-to-end driving: Architectures and training methods, IEEE Trans. Neural Netw. Learn. Syst. 33 (4) (2020) 1364–1384.

[11] M.-j. Lee, Y.-g. Ha, Autonomous driving control using end-to-end deep learning, in: 2020 IEEE International Conference on Big Data and Smart Computing, BigComp, IEEE, 2020, pp. 470–473.

[12] P.S. Chib, P. Singh, Recent advancements in end-to-end autonomous driving using deep learning: A survey, IEEE Trans. Intell. Veh. (2023).

[13] K. Chen, L. Bai, F. Ling, P. Ye, T. Chen, K. Chen, T. Han, W. Ouyang, Towards an end-to-end artificial intelligence driven global weather forecasting system, 2023, arXiv preprint arXiv:2312.12462.

[14] K. Si, Y. Xue, X. Yu, X. Zhu, Q. Li, W. Gong, T. Liang, S. Duan, Fully end-to-end deep-learning-based diagnosis of pancreatic tumors, Theranostics 11 (4) (2021) 1982.

[15] Y. Song, A. Elibol, N.Y. Chong, Abdominal multi-organ segmentation using multi-scale and context-aware neural networks, IFAC J. Syst. Control. 27 (2024) 100249.

[16] F. Rafique, L. Fu, R. Mai, End to end machine learning for fault detection and classification in power transmission lines, Electr. Power Syst. Res. 199 (2021) 107430.

[17] L. Zhang, Q. Fan, J. Lin, Z. Zhang, X. Yan, C. Li, A nearly end-to-end deep learning approach to fault diagnosis of wind turbine gearboxes under nonstationary conditions, Eng. Appl. Artif. Intell. 119 (2023) 105735.

[18] Y. Luo, Y. Zhang, X. Cai, X. Yuan, E2gan: End-to-end generative adversarial network for multivariate time series imputation, in: Proceedings of the 28th International Joint Conference on Artificial Intelligence, AAAI Press Palo Alto, CA, USA, 2019, pp. 3094–3100.

[19] A.D. Pazho, C. Neff, G.A. Noghre, B.R. Ardabili, S. Yao, M. Baharani, H. Tabkhi, Ancilia: Scalable intelligent video surveillance for the artificial intelligence of things, IEEE Internet Things J. (2023).

[20] K. Simonyan, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.

[21] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[22] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, et al., Wavenet: A generative model for raw audio, 12, 2016, arXiv preprint arXiv:1609.03499.

[23] A. Hannun, Deep speech: Scaling up end-to-end speech recognition, 2014, arXiv preprint arXiv:1412.5567.

[24] A. Vaswani, Attention is all you need, Adv. Neural Inf. Process. Syst. (2017).

[25] J. Krejsa, S. Věchet, J. Hrbáček, P. Schreiber, High level software architecture for autonomous mobile robot, Recent. Adv. Mechatronics (2010) 185–190.

[26] M. Pancerasa, M. Sangiorgio, R. Ambrosini, N. Saino, D.W. Winkler, R. Casagrandi, Reconstruction of long-distance bird migration routes using advanced machine learning techniques on geolocator data, J. R. Soc. Interface 16 (155) (2019) 20190031.

[27] C. Mühlroth, M. Grottke, Artificial intelligence in innovation: how to spot emerging trends and technologies, IEEE Trans. Eng. Manage. 69 (2) (2020) 493–510.

[28] C. Jin, W. Chen, Y. Cao, Z. Xu, Z. Tan, X. Zhang, L. Deng, C. Zheng, J. Zhou, H. Shi, et al., Development and evaluation of an artificial intelligence system for COVID-19 diagnosis, Nat. Commun. 11 (1) (2020) 5088.

[29] A. Hussein, S. Watanabe, A. Ali, Arabic speech recognition by end-to-end, modular systems and human, Comput. Speech Lang. 71 (2022) 101272.

[30] N. Borghi, G. Guariso, M. Sangiorgio, Forecasting convective storms trajectory and intensity by neural networks, Forecasting 6 (2) (2024) 326–342.

[31] J. Pathak, B. Hunt, M. Girvan, Z. Lu, E. Ott, Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach, Phys. Rev. Lett. 120 (2) (2018) 024102.

[32] J. Pathak, A. Wikner, R. Fussell, S. Chandra, B.R. Hunt, M. Girvan, E. Ott, Hybrid forecasting of chaotic processes: Using machine learning in conjunction with a knowledge-based model, Chaos: Interdiscip. J. Nonlinear Sci. 28 (4) (2018).

[33] M. Sangiorgio, F. Dercole, Robustness of LSTM neural networks for multi-step forecasting of chaotic time series, Chaos Solitons Fractals 139 (2020) 110045.

[34] F. Dercole, M. Sangiorgio, Y. Schmirander, An empirical assessment of the universality of ANNs to predict oscillatory time series, IFAC-PapersOnLine 53 (2) (2020) 1255–1260.

[35] H. Jaeger, H. Haas, Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication, Science 304 (5667) (2004) 78–80.

[36] G. Bontempi, S. Ben Taieb, Y.-A. Le Borgne, Machine learning strategies for time series forecasting, in: Business Intelligence: Second European Summer School, eBISS 2012, Brussels, Belgium, July 15-21, 2012, Tutorial Lectures 2, Springer, 2013, pp. 62–77.

[37] J. Bouvrie, B. Hamzi, Kernel methods for the approximation of nonlinear systems, SIAM J. Control Optim. 55 (4) (2017) 2460–2492.

[38] B. Lim, S. Zohren, Time-series forecasting with deep learning: a survey, Phil. Trans. R. Soc. A 379 (2194) (2021) 20200209.

[39] B. Hamzi, H. Owhadi, Learning dynamical systems from data: a simple cross-validation perspective, part I: parametric kernel flows, Phys. D: Nonlinear Phenom. 421 (2021) 132817.

[40] L. Yang, B. Hamzi, Y. Kevrekidis, H. Owhadi, X. Sun, N. Xie, Hausdorff metric based training of kernels to learn attractors with application to 133 chaotic dynamical systems, Phys. D: Nonlinear Phenom. (2024) 134192.

[41] F. Takens, Detecting strange attractors in turbulence, in: Dynamical Systems and Turbulence, Warwick 1980, Springer, 1981, pp. 366–381.

[42] T. Sauer, J.A. Yorke, How many delay coordinates do you need? Int. J. Bifurc. Chaos 3 (03) (1993) 737–744.

[43] J. Zhang, H.S.-H. Chung, W.-L. Lo, Chaotic time series prediction using a neuro-fuzzy system with time-delay coordinates, IEEE Trans. Knowl. Data Eng. 20 (7) (2008) 956–964.

[44] M. Sangiorgio, F. Dercole, G. Guariso, Basic concepts of chaos theory and nonlinear time-series analysis, in: Deep Learning in Multi-Step Prediction of Chaotic Dynamics: From Deterministic Models to Real-World Systems, Springer, 2022, pp. 11–29.
[45] E. Tan, S. Algar, D. Corrêa, M. Small, T. Stemler, D. Walker, Selecting embedding delays: An overview of embedding techniques and a new method using persistent homology, Chaos: Interdiscip. J. Nonlinear Sci. 33 (3) (2023).
[46] K.T. Alligood, T.D. Sauer, J.A. Yorke, Chaos: An Introduction to Dynamical Systems, Springer New York, NY, 1996.
[47] M. Sangiorgio, F. Dercole, G. Guariso, Deep Learning in Multi-Step Prediction of Chaotic Dynamics: from Deterministic Models to Real-World Systems, Springer, 2022.
[48] D. Ayers, J. Lau, J. Amezcua, A. Carrassi, V. Ojha, Supervised machine learning to estimate instabilities in chaotic systems: Estimation of local Lyapunov exponents, Q. J. R. Meteorol. Soc. 149 (753) (2023) 1236–1262.
[49] J. Pathak, Z. Lu, B.R. Hunt, M. Girvan, E. Ott, Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data, Chaos: Interdiscip. J. Nonlinear Sci. 27 (12) (2017).
[50] S. Zhou, X. Wang, Identifying the linear region based on machine learning to calculate the largest Lyapunov exponent from chaotic time series, Chaos: Interdiscip. J. Nonlinear Sci. 28 (12) (2018).
[51] R. Pulch, A machine learning approach for identifying an effective dimension in parametric dynamical systems, PAMM 21 (1) (2021) e202100252.
[52] M. Ricci, N. Moriel, Z. Piran, M. Nitzan, Phase2vec: Dynamical systems embedding with a physics-informed convolutional network, 2022, arXiv preprint arXiv:2212.03857.
[53] T. Nogawa, Dimensional reduction of dynamical systems by machine learning: automatic generation of the optimum extensive variables and their time-evolution map, J. Stat. Mech. Theory Exp. 2023 (12) (2023) 123404.
[54] R. Yoon, B. Osting, A dynamical system-based framework for dimension reduction, Commun. Appl. Math. Comput. 6 (2) (2024) 757–789.
[55] B. Lusch, J.N. Kutz, S.L. Brunton, Deep learning for universal linear embeddings of nonlinear dynamics, Nat. Commun. 9 (1) (2018) 4950.
[56] N. Pallikarakis, A. Ntargaras, Application of machine learning regression models to inverse eigenvalue problems, Comput. Math. Appl. 154 (2024) 162–174.
[57] S. Bittanti, Model Identification and Data Analysis, John Wiley & Sons, 2019.
[58] P. Dubois, T. Gomez, L. Planckaert, L. Perret, Data-driven predictions of the Lorenz system, Phys. D: Nonlinear Phenom. 408 (2020) 132495.
[59] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[60] H. Xiao, M.A. Sotelo, Y. Ma, B. Cao, Y. Zhou, Y. Xu, R. Wang, Z. Li, An improved LSTM model for behavior recognition of intelligent vehicles, IEEE Access 8 (2020) 101514–101527.
[61] A. Sherstinsky, Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network, Phys. D: Nonlinear Phenom. 404 (2020) 132306.
[62] L. Qian, Z. Yuelei, Z. Haiqing, S. Xiaoyun, H. Guang, Z. Jun, L. Baoan, G. Kang, Improved attention-based AE-LSTM for short-term power load forecasting, in: 2022 37th Youth Academic Annual Conference of Chinese Association of Automation, YAC, IEEE, 2022, pp. 1292–1296.
[63] M. Liu, Y. Li, J. Hu, X. Wu, S. Deng, H. Li, A new hybrid model based on SCINet and LSTM for short-term power load forecasting, Energies 17 (1) (2023) 95.
[64] P. Voditel, A. Gurjar, A. Pandey, A. Jain, N. Sharma, N. Dubey, Image captioning-A deep learning approach using CNN and LSTM network, in: 2023 3rd International Conference on Pervasive Computing and Social Networking, ICPCSN, IEEE, 2023, pp. 343–348.
[65] K.J. Åström, Lectures on the Identification Problem: The Least Squares Method, Tech. Rep., Department of Automatic Control, Lund Institute of Technology (LTH), 1968.
[66] A. Amaranto, M. Mazzoleni, B-AMA: A python-coded protocol to enhance the application of data-driven models in hydrology, Environ. Model. Softw. 160 (2023) 105609.
[67] R. Gencay, T. Liu, Nonlinear modelling and prediction with feedforward and recurrent networks, Phys. D: Nonlinear Phenom. 108 (1–2) (1997) 119–134.
[68] G. Guariso, G. Nunnari, M. Sangiorgio, Multi-step solar irradiance forecasting and domain adaptation of deep neural networks, Energies 13 (15) (2020) 3987.
[69] M. Sangiorgio, F. Dercole, G. Guariso, Forecasting of noisy chaotic systems with deep neural networks, Chaos Solitons Fractals 153 (2021) 111570.
[70] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[71] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[72] L. Prechelt, Early stopping-but when? in: Neural Networks: Tricks of the Trade, Springer, 2002, pp. 55–69.
[73] C. Garth, X. Tricoche, et al., Interactive computation and rendering of finite-time Lyapunov exponent fields, IEEE Trans. Vis. Comput. Graphics 18 (8) (2012) 1368–1380.
[74] A. Pikovsky, A. Politi, Lyapunov Exponents: A Tool to Explore Complex Dynamics, Cambridge University Press, 2016.
[75] G. Nastac, J.W. Labahn, L. Magri, M. Ihme, Lyapunov exponent as a metric for assessing the dynamic content and predictability of large-eddy simulations, Phys. Rev. Fluids 2 (9) (2017) 094606.
[76] L. De Cruz, S. Schubert, J. Demaeyer, V. Lucarini, S. Vannitsem, Exploring the Lyapunov instability properties of high-dimensional atmospheric and climate models, Nonlinear Process. Geophys. 25 (2) (2018) 387–412.