

A first step toward using Artificial Intelligence to learn algorithms for time-series analysis

Emanuele Salgarollo¹, Matteo Sangiorgio¹, and Fabio Dercole¹

Department of Electronics, Information and Bioengineering, Politecnico di Milano,
Via Giuseppe Ponzio 34, I-20133 Milan, Italy
`matteo.sangiorgio@polimi.it`

Abstract. We challenge Machine Learning (ML) to learn an end-to-end computational task in the context of time-series and dynamical systems analysis. We go beyond the typical two-step procedure consisting of ML taking care of model identification, followed by traditional algorithms performing the computation on the identified model. We tackle the archetypical task of computing the eigenvalues of a linear system's state-transition matrix from an output time series. We train an LSTM network with data generated from random matrices of dimension 2-to-5. The performance is remarkable in estimating the dominant and complex conjugate sub-dominant eigenvalues, whereas sub-dominant real eigenvalues are estimated less precisely since the associated dynamics remain hidden in the time series. This preliminary test opens the door for the end-to-end learning of many complex algorithms in nonlinear analysis, such as the estimation of the embedding dimension of time series and of the dynamic and geometric features of the corresponding attractor.

Keywords: Machine learning, Neural Networks, End-to-end learning, LSTM cell, Pattern recognition, Linear systems

1 Introduction

In recent years, Artificial Intelligence (AI) applications have been moving towards end-to-end learning approaches, where the whole task is carried out entirely by the neural model, independently of traditional algorithms, and without the need for human guidance. This is the case of generative models employed in natural language processing, autonomous driving, weather forecasting, and several other applications [1–3]. However, many traditional engineering domains still rely on modular frameworks where Machine Learning (ML) techniques are used for some sub-tasks and traditional algorithms for others [4–7].

An archetypal example in the field of time-series analysis and dynamical systems theory is that of estimating the embedding dimension required to look at the data series as the observable of a dynamical system [8–10]. Through vectors of consecutive data (vectors of delayed outputs), the time series gets linked to the quantitative features representative of the state space dynamics of the underlying system, such as the type of attractor (periodic, quasi-periodic, or

chaotic), its Lyapunov exponents measuring attractiveness and chaoticity, the fractal dimension for strange attractors, etc.

The current state of the art requires solving a system identification task to obtain a high-fidelity model of the original system in the space of delayed outputs, and then using its trajectories to derive the desired feature employing specific algorithms developed in the context of nonlinear system theory [11].

In this work, we demonstrate the possibility of solving this type of problem through end-to-end learning techniques, characterized by the ability of the ML tool to directly learn the whole procedure without the need for any intermediate (human-driven) step. Specifically, as a first attempt, we focus on a very basic problem for linear dynamical systems, that is the estimation of the eigenvalues of the state-transition matrix of a linear time-invariant dynamical system from a time series of its output variable.

We tackle the problem using a Recurrent Neural Network (RNN), a neural architecture specifically designed to process sequential data. Long Short-Term Memory (LSTM) cells [12] are used as neurons because of their ability to selectively retain important information on the process dynamic for a long duration. Such a neural model demonstrates the ability to estimate the eigenvalues with high accuracy (R^2 -score > 0.9 for systems of order equal to 4 or lower). Our analysis shows that the dominant eigenvalues, which define the main system’s dynamics, are always accurately estimated. The other eigenvalues can be approximated with high precision only if they are a pair of complex conjugates, while non-dominant real eigenvalues are generally estimated with lower accuracy. Our work paves the way for the development of an innovative end-to-end ML tool for the analysis of dynamical systems and time series.

2 Methods

2.1 Learning setup

The dataset used for the learning task is generated by simulating N linear time-invariant systems in 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 state ($n \times 1$ vector), $y(t)$ is the output variable (scalar), A is the state-transition matrix ($n \times n$), and C is used to compute the output transformation ($1 \times n$ vector). The dataset is thus composed of N pairs of the neural network input I and corresponding target output O . I is a sequence of H values of $y(t)$ ($H = 200$ in our implementation). O is a list of $2n$ elements, representing the real and imaginary parts of the n eigenvalues of the state-transition matrix A . The eigenvalues are sorted starting with the dominant one, i.e., the one with the largest magnitude, followed by the real and imaginary pairs or the others in decreasing order of magnitude.

We generate each pair I, O using Algorithm 1. δ_p is the probability to extract a pair of complex conjugate eigenvalues and a probability $1 - \delta_p$ to extract a real eigenvalue. We use $\delta_p = 0.8$, which gives a frequency of real eigenvalues in line with a random extraction of the elements of matrix A . However, extracting the eigenvalues and the transformation matrix T , as in Algorithm 1, 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. The elements of the target output O from $2n + 1$ to $2n_{\max}$ are set to zero, to force the network to learn $n_{\max} - n$ spurious null eigenvalues. This will be used in post-processing to infer the dimension n of the system that generated the input sequence.

Algorithm 1 Dataset generation

```

1:  $O \leftarrow \vec{0}$  # Initialize the  $2n_{\max}$  elements of  $O$  to 0
2: randomly select  $n$  between 2 and  $n_{\max}$ 
3:  $i \leftarrow 1$ 
4: while  $i \leq n - 1$  do
5:   randomly extract  $p \sim \mathcal{U}(0, 1)$ 
6:   if  $p < \delta_p$  then # Pair of complex conjugates
7:     randomly extract  $\rho^2 \sim \mathcal{U}(0, 1)$  # This gives complex eigenvalues uniformly distributed
       in the unit disk
8:     randomly extract  $\theta \sim \mathcal{U}(0, \pi)$ 
9:     define two complex eigenvalues:  $e_i = \rho \cdot \exp(\theta \cdot j)$  and  $e_{i+1} = \rho \cdot \exp(-\theta \cdot j)$ 
10:     $O_{2(i-1)+1} = \text{Re}(e_i)$ ;  $O_{2(i-1)+2} = \text{Im}(e_i)$ 
11:     $O_{2(i-1)+3} = \text{Re}(e_{i+1})$ ;  $O_{2(i-1)+4} = \text{Im}(e_{i+1})$ 
12:     $i \leftarrow i + 2$  # Skip both eigenvalues
13:   else # Real eigenvalue
14:     randomly extract  $\text{Re}(e_i) \sim \mathcal{U}(-1, 1)$ 
15:     define a real eigenvalue  $e_i = \text{Re}(e_i)$ 
16:      $O_{2(i-1)+1} = e_i$ ;  $O_{2(i-1)+2} = 0$ 
17:      $i \leftarrow i + 1$ 
18:   end if
19: end while
20: if  $i = n$  then # Last unpaired eigenvalue is real
21:   randomly extract  $\text{Re}(e_i) \sim \mathcal{U}(-1, 1)$ 
22:   define a real eigenvalue  $e_i = \text{Re}(e_i)$ 
23:    $O_{2(i-1)+1} = e_i$ ;  $O_{2(i-1)+2} = 0$ 
24: end if
25: organize the eigenvalues in a  $n \times n$  block diagonal matrix  $D$  (a  $1 \times 1$  block  $e_i$  for each real
    eigenvalue; a  $2 \times 2$  block  $\rho \cdot \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$  for each pair of complex conjugate eigenvalues)
26: randomly extract the elements of  $n \times n$  matrix  $T$  from a  $\mathcal{U}(-1, 1)$ 
27: compute  $A = T \cdot D \cdot T^{-1}$ 
28: randomly extract the elements of  $1 \times n$  vector  $C$  from a  $\mathcal{U}(-1, 1)$ 
29: randomly extract  $x(0) \sim \mathcal{U}(-1, 1)$ 
30: produce the output  $y(t)$  for  $H$  time steps simulating the dynamical system of eq. (1)

```

It is worth noting that, to solve the eigenvalues estimation task the knowledge of the full state is not necessary. If the system is observable (a condition that is verified with probability one in our experimental setting), the time series of the system's (scalar) output $y(t)$ is sufficient to fully reconstruct its dynamics.

2.2 Neural approach

We tackle the eigenvalues' estimation problem by adopting RNNs, which are specifically designed to deal with sequential tasks and are thus well-suited to process the time series of $y(t)$. 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 [13], a remarkable advantage compared to fully connected architectures whose number of parameters linearly increases with the sequence length. The considered RNN is formed by LSTM cells, recurrent neurons with two internal states (hidden state 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 of the LSTM cell) is responsible for tracking long-term dynamics and is propagated only along the temporal dimension. LSTM cells have three gates (input, output, and forget gate) that modulate the information flows through a sigmoid function returning values between 0 (closed gate) and 1 (open gate). The forget gate resets the cell state if the combination of input and hidden state into the sigmoid is low; the input gate defines how much the candidate cell state update actually affects the cell state, protecting it from irrelevant contributions; the output gate modulates the candidate hidden state (that also represents the cell's output). This gated structure can retain relevant information 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 to zero at the beginning of each sequence allowing the parallelization of the training procedure [14].

Once the LSTM layer has sequentially processed all the time steps of I , the neurons' hidden states 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' estimate ($\hat{O}$). This latter is then post-processed by a corrector algorithm (Algorithm 2) to identify the dimension n of the system that generated the input series and to rectify unavoidable numerical inconsistencies due to the fact that there are no constraints in the network that force the estimated eigenvalues to be either real or (pairs of) complex conjugate values. 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), that 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. We use the thresholds values $\delta_1 = 0.05$ and $\delta_2 = 0.1$.

Following the standard ML practice, we split the available data into training data (70%) used for the calibration of the network's parameters, 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 a final assessment of the performance.

We train the neural model to minimize the Mean Squared Error (MSE) between actual and estimated eigenvalues. The gradients of the network's parameters with respect to the MSE are computed through the back-propagation

Algorithm 2 Eigenvalues corrector

```

1:  $n \leftarrow n_{\max}$  # Dimension estimate
2:  $i \leftarrow 1$ 
3: while  $i \leq n_{\max} - 1$  do
4:   if  $|\tilde{O}_i| < \delta_1$  then # Spurious eigenvalue
5:      $\tilde{O}_i \leftarrow 0$ 
6:      $n \leftarrow n - 1$ 
7:      $i \leftarrow i + 1$ 
8:   else if  $|\text{Im}(\tilde{O}_i)| > \delta_1$  and  $|\text{Im}(\tilde{O}_i) + \text{Im}(\tilde{O}_{i+1})| < \delta_2$  and  $|\text{Re}(\tilde{O}_i) - \text{Re}(\tilde{O}_{i+1})| < \delta_2$  then #
      Complex conjugate pair
9:      $\text{Re}(\tilde{O}_i) \leftarrow (\text{Re}(\tilde{O}_i) + \text{Re}(\tilde{O}_{i+1}))/2$ 
10:     $\text{Re}(\tilde{O}_{i+1}) \leftarrow \text{Re}(\tilde{O}_i)$ 
11:     $\text{Im}(\tilde{O}_i) \leftarrow (|\text{Im}(\tilde{O}_i)| + |\text{Im}(\tilde{O}_{i+1})|)/2$ 
12:     $\text{Im}(\tilde{O}_{i+1}) \leftarrow -\text{Im}(\tilde{O}_i)$ 
13:     $i \leftarrow i + 2$  # Skip both eigenvalues
14:   else # Real eigenvalue
15:      $\text{Re}(\tilde{O}_i) \leftarrow \text{Re}(\tilde{O}_i)$ 
16:      $\text{Im}(\tilde{O}_i) \leftarrow 0$ 
17:      $i \leftarrow i + 1$ 
18:   end if
19: end while
20: if  $i = n_{\max}$  then # Last unpaired eigenvalue
21:   if  $|\tilde{O}_i| < \delta_1$  then # Spurious eigenvalue
22:      $\tilde{O}_i \leftarrow 0$ 
23:      $n \leftarrow n - 1$ 
24:   else # Real eigenvalue
25:      $\text{Re}(\tilde{O}_i) \leftarrow \text{Re}(\tilde{O}_i)$ 
26:      $\text{Im}(\tilde{O}_i) \leftarrow 0$ 
27:   end if
28: end if

```

through time algorithm [15], a version of the standard back-propagation modified to deal with recurrent architectures. The gradients are used by Adam (adaptive moment estimation) optimizer to iteratively update the network’s parameters. 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. We perform the training for a maximum of 1000 epochs. The MSE in validation is monitored during the learning process so that we can stop it if there is no improvement for 200 consecutive epochs (early stopping). 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 gradients computed for each specific batch).

Hyperparameter tuning is performed following the grid search approach, i.e., considering all the combinations of 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). 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 negative im-

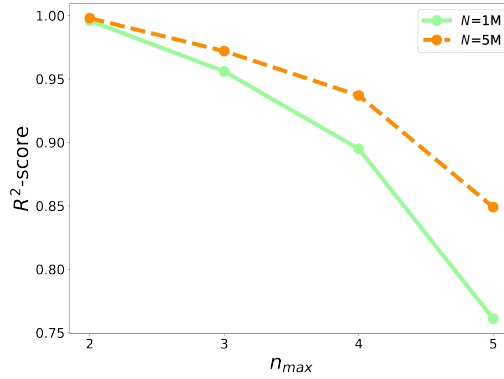


Fig. 1. Accuracy (R^2 -score) computed on the test set for different maximum system order ($n_{\max}$) and dataset dimensionality (N).

part of unfortunate initializations of the parameters is limited by repeating the optimization twice with different random seeds for each node of the grid.

3 Results

We first explore how the system order ($n_{\max}$, varied between 2 and 5) and dataset dimensionality (N , set to 1 million or 5 million) influence the accuracy of the proposed neural approach (Fig. 1).

The R^2 -score degrades for increasing values of $n_{\max}$ (from 2 to 5) because this parameter sets the complexity of the task. When $n_{\max} = 2$, only two configurations are possible (eigenvalues can be both real or a pair of complex conjugates). $n_{\max} = 5$ is associated with a wide range of possible cases ($n = 2$ in just one of them) and with different configurations. For instance, if $n = 5$, we can have 5 real eigenvalues, 3 real and a pair of complex conjugate eigenvalues, or only one real and two pairs of complex conjugate eigenvalues, with subcases depending on whether the dominant eigenvalue is real or complex.

As expected, a greater amount of data boosts the performance of the neural network, as is always the case when an ML tool is used to tackle a complex task. The harder the task becomes (larger $n_{\max}$ values), the more relevant the gap in terms of R^2 -score. For $n_{\max} = 2$, the task is precisely solved (R^2 -score > 0.99) even with $N = 1$ million. For $n_{\max} = 5$, the gap in terms of R^2 -score between $N = 1$ and 5 million is relevant (0.09, see Fig. 1).

To provide a wider picture of the analysis, we report in Fig. 2 a graphical assessment for $n_{\max} = 4$ and $N = 5$ million, for which the R^2 -score is equal to 0.937. We specifically selected the systems to cover all the possible combinations that can be generated by a state-transition matrix of order four (4 real eigenvalues; 2 real and a pair of complex conjugate eigenvalues; 2 pairs of complex conjugates), three (3 real eigenvalues; one real eigenvalue and a pair of complex

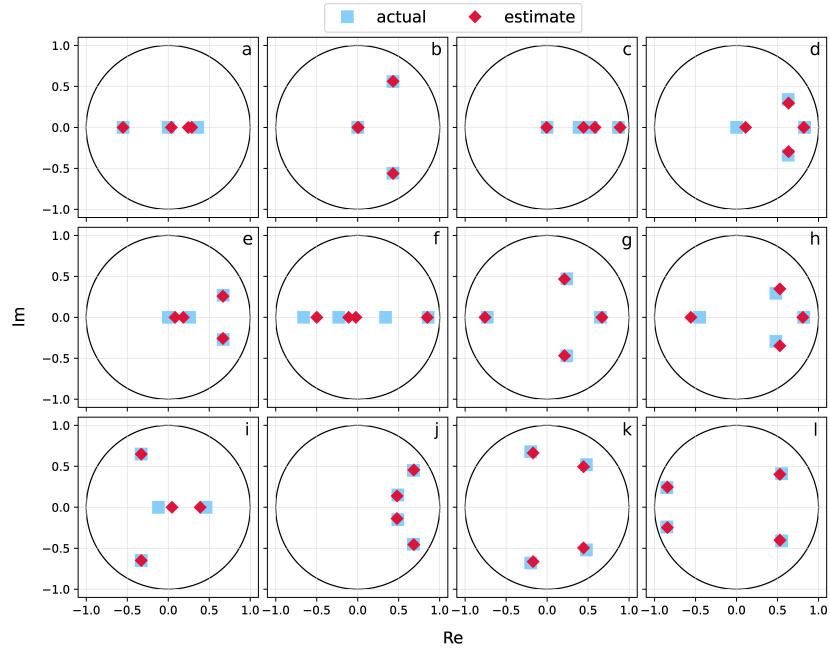


Fig. 2. Positions of the actual (light blue squares) and estimated (red diamonds) eigenvalues in the complex plane for $n_{\max} = 4$ and $N = 5M$.

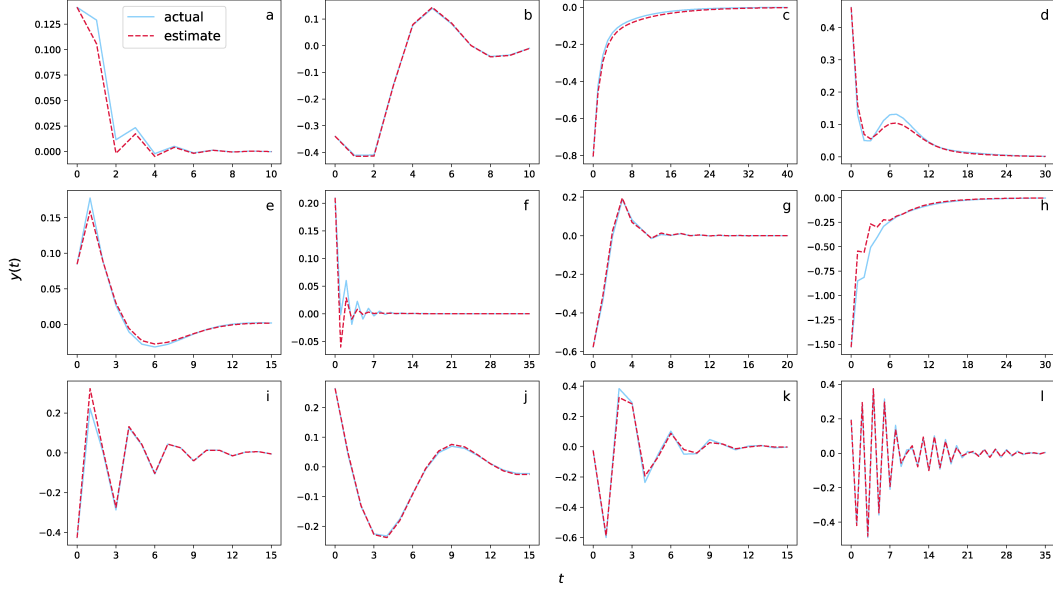


Fig. 3. Time series of the dynamical systems' output $y(t)$ generated by the actual (light blue continuous line) and estimated (red dashed line) eigenvalues. In each panel, time series are relative to the corresponding case of Fig. 2. The number of simulated time steps is equal to the time response of the dynamical system.

conjugates) and two (2 real eigenvalues; a pair of complex conjugates). The neural network generally estimates the eigenvalues with high accuracy (actual eigenvalues are represented by light blue circles, estimated ones by red diamonds). In particular, complex conjugate eigenvalues are always estimated more accurately than real ones, especially if non-dominant. The reason may be related to the fact that oscillations in the time series generated by complex conjugate eigenvalues can be relatively easily identified. Conversely, real eigenvalues produce monotonic dynamics, that quickly expire especially when the eigenvalue is close to 0, resulting in patterns that are much harder to spot.

We further investigate the impacts of the differences between actual and estimated eigenvalues shown in Fig. 3. The dynamic representative of each set of eigenvalues is computed by considering the block-diagonal matrix D used in Algorithm 1 to generate the state-transition matrix A and the block-diagonal matrix $\hat{D}$ obtained with the estimated eigenvalues. The two matrices D and $\hat{D}$ are used as state-transition matrices (with random output vector C and initial condition $x(0)$), to generate two output signals, the first representative of the actual eigenvalues and the second of the estimated ones. The comparison shows that the actual and estimated time series are very similar in all the panels of Fig. 3 and highlights how difficult it is to discern small dissimilarities in terms of different eigenvalues. Even in the critical case of poorly estimated non-dominant real eigenvalues (Fig. 3f), the differences in the time series are quite small.

4 Conclusion

We explored the possibility of using ML tools to derive the quantitative features representative of a time series and of the underlying dynamical system that generated the time series. The peculiarity of our end-to-end approach is that we train the AI to directly compute the features of interest from a sequence of a system’s observable without relying on any classical algorithm from dynamical systems theory. This represents a paradigm shift in the context of ML tools for dynamical systems, which are usually employed to solve system identification tasks, on top of which classical algorithms compute the features of interest. Contrary to this mainstream approach, in which ML essentially reproduces a mapping between input and output variables of the dynamical system, our direct, end-to-end approach requires the AI to learn the entire algorithm from the system’s observable to the features of interest; a much more complex task.

As a first step towards the development of end-to-end AI tools for time-series and dynamical systems analysis, we considered the problem of estimating the eigenvalues associated with the state-transition matrix of a linear (discrete-time, time-invariant) dynamical system. We adopted recurrent neural networks with LSTM cells, which are specifically designed for sequential data.

Our results show that the proposed approach can accurately estimate the eigenvalues of the state-transition matrix, provided that enough data are used for the learning of the neural network. 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 system that generated the input series, even though our neural network is not specifically trained to learn this feature, which is essentially categorical (see a specific comment below). Nonetheless, the accuracy in the estimation of n follows a trend similar to the R^2 -score shown in Fig. 1 (for the case $N = 5M$: 97.1% for $n_{\max} = 2$; 85.6% for $n_{\max} = 3$; 77.5% for $n_{\max} = 4$; 61.9% for $n_{\max} = 5$).

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 to evaluate the robustness of the tested architectures to noisy data, in order to make the AI tool ready to be used on real datasets. Both directions are interesting to be pursued. However, in our opinion, the interest is in acquiring most of the experience from this basic task, the eigenvalue estimation problem, to be then exploited on end-to-end tasks on complex nonlinear systems. On the one hand, 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. On the other hand, many real datasets are poorly described by linear models. Nonetheless, the understanding of the eigenvalue problem is preparatory for our aim of tackling more complex tasks in an end-to-end fashion.

One 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 reported in the Methods. The neural network 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.

Other than the theoretical relevance of the insights derived from this study, the problem here considered can potentially be used as a benchmark to evaluate the performance of novel ML tools (neural architectures, learning procedures, ...). Indeed, the proposed task is scalable in terms of complexity and dimensionality (the order of the system, $n_{\max}$, and the number of data available, N). Extensions of the problem to other quantitative features representative of the dynamical systems (e.g., Lyapunov exponents, fractal dimension, and embedding dimension) are relatively straightforward and would result in even more challenging benchmarks.

Bibliography

- [1] Tan, X., Chen, J., Liu, H., Cong, J., Zhang, C., Liu, Y., Wang, X., Leng, Y., Yi, Y., He, L., et al.: Naturspeech: End-to-end text-to-speech synthesis with human-level quality. *IEEE T Pattern Anal* (2024)
- [2] Tampuu, A., Matiisen, T., Semikin, M., Fishman, D., Muhammad, N.: A survey of end-to-end driving: Architectures and training methods. *IEEE T Neur Net Lear* **33**(4), 1364–1384 (2020)
- [3] Si, K., Xue, Y., Yu, X., Zhu, X., Li, Q., Gong, W., Liang, T., Duan, S.: Fully end-to-end deep-learning-based diagnosis of pancreatic tumors. *Theranostics* **11**(4), 1982 (2021)
- [4] Pancerasa, M., Sangiorgio, M., Ambrosini, R., Saino, N., Winkler, D.W., Casagrandi, R.: Reconstruction of long-distance bird migration routes using advanced machine learning techniques on geolocator data. *J Roy Soc Interface* **16**(155), 20190,031 (2019)
- [5] Mühlroth, C., Grottke, M.: Artificial intelligence in innovation: how to spot emerging trends and technologies. *IEEE T Eng Manage* **69**(2), 493–510 (2020)
- [6] Jin, C., Chen, W., Cao, Y., Xu, Z., Tan, Z., Zhang, X., Deng, L., Zheng, C., Zhou, J., Shi, H., et al.: Development and evaluation of an artificial intelligence system for covid-19 diagnosis. *Nat Commun* **11**(1), 5088 (2020)
- [7] Hussein, A., Watanabe, S., Ali, A.: Arabic speech recognition by end-to-end, modular systems and human. *Comput Speech Lang* **71**, 101,272 (2022)
- [8] Takens, F.: Detecting strange attractors in turbulence. In: *Dynamical systems and turbulence*, Warwick 1980, pp. 366–381. Springer (1981)
- [9] Sangiorgio, M., Dercole, F., Guariso, G.: 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*, pp. 11–29. Springer (2022)
- [10] Tan, E., Algar, S., Corrêa, D., Small, M., Stemler, T., Walker, D.: Selecting embedding delays: An overview of embedding techniques and a new method using persistent homology. *Chaos* **33**(3) (2023)
- [11] Pathak, J., Lu, Z., Hunt, B.R., Girvan, M., Ott, E.: Using machine learning to replicate chaotic attractors and calculate lyapunov exponents from data. *Chaos* **27**(12) (2017)
- [12] Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Comput* **9**(8), 1735–1780 (1997)
- [13] Sangiorgio, M., Dercole, F.: Robustness of LSTM neural networks for multi-step forecasting of chaotic time series. *Chaos Soliton Fract* **139**, 110,045 (2020)
- [14] Sangiorgio, M., Dercole, F., Guariso, G.: Forecasting of noisy chaotic systems with deep neural networks. *Chaos Soliton Fract* **153**, 111,570 (2021)
- [15] Goodfellow, I., Bengio, Y., Courville, A.: *Deep learning*. MIT press (2016)