Accuracy of neural networks for the simulation of chaotic dynamics: precision of training data vs precision of the algorithm

S Bompas, B. Georgeot, D. Guery-Odelin

Accuracy of neural networks for the simulation of chaotic dynamics: precision of training data vs precision of the algorithm

S. Bompas,[1, 2] B. Georgeot,[2] and D. Guéry-Odelin[1]

[1)] *Laboratoire Collisions, Agrégats, Réactivité, IRSAMC, Université de Toulouse, CNRS, UPS, France*

[2)] *Laboratoire de Physique Théorique, IRSAMC, Université de Toulouse, CNRS, UPS, France*

(Dated: July 9, 2020)

We explore the influence of precision of the data and the algorithm for the simulation of chaotic dynamics by neural networks techniques. For this purpose, we simulate the Lorenz system with different precisions using three different neural network techniques adapted to time series, namely reservoir computing (using ESN), LSTM and TCN, for both short and long time predictions, and assess their efficiency and accuracy. Our results show that the precision of the algorithm is more important than the precision of the training data for the accuracy of the predictions. This result gives support to the idea that neural networks can perform time-series predictions in many practical applications for which data are necessarily of limited precision, in line with recent results. It also suggests that for a given set of data the reliability of the predictions can be significantly improved by using a network with higher precision than the one of the data.

I. INTRODUCTION

Techniques of machine learning have been shown lately to be efficient in a huge variety of tasks, from playing the game of Go[1] to speech recognition[2] or automatic translation[3]. In many cases, such breakthroughs correspond to complicated tasks with complex decision-making processes. However, it was highlighted recently that such tools can also be useful in tasks which are much more adapted to standard algorithms, such as simulation of physical systems. Indeed, in[4–6], it was shown that a certain type of machine learning algorithms called reservoir computing[7] was able to forecast the evolution of complex physical systems, namely a fully chaotic model (see also Ref.[8] and references therein). Remarkably enough, the simulation is made from the time series of the previous states of the system, without solving explicitly the equations defining the model. It was also shown that other types of neural networks may be efficient as well in predicting the behaviour of such systems[9–12].

So far, the results have shown that different machine learning techniques can simulate chaotic dynamics, both for short and long times. However, it is important for future applications to assess the accuracy of these techniques in a precise way. In this paper, we explore the role of precision of the data used for the training of the network and of the algorithm itself on the accuracy of the simulation. We do so on a specific case of reservoir computing (Echo State Network-ESN) as well as on two other standard machine learning techniques used in this context, commonly called LSTM[13] and TCN[14] techniques. We compare the accuracy of these methods to the explicit integration of the equations of motion, both for short time and long time predictions of a well known chaotic system originating from meteorology, the Lorenz system. Our results show that the precision of the algorithm is more important than the precision of the training data for the accuracy of the simulation. This has interesting consequences for applications, since the precision of the algorithm is by far easier to control than the one of the training data. We also discuss the training by considering trajectories of different size and by computing the time required to train the networks.

II. SYSTEM STUDIED

The Lorenz system was introduced in 1963 by Edward Lorenz[15] as an extremely simplified model of meteorology. It corresponds to a set of three nonlinear coupled equations for the variables x, y and z as a function of time:

$$\dot{x} = \sigma(y - x) \tag{1}$$

$$\dot{y} = x(\rho - z) \tag{2}$$

$$\dot{z} = xy - \beta z. \tag{3}$$

Throughout the paper we choose the standard set of parameters: $\sigma = 10, \rho = 28$ and $\beta = 8/3$.

This nonlinear and dissipative model displays chaotic features. In particular, trajectories converge to a low dimensional but complex structure referred to as a strange attractor, and in this specific case as the Lorenz attractor. In Fig. 1, we show an example of a trajectory of the system and the attractor.

We distinguish two types of predictions. The short term predictions are similar to meteorological predictions: one starts from a specific initial point, and the aim is to follow a specific trajectory of the system for as long as possible. For strongly chaotic systems, this kind of predictions is limited by the exponential growth of perturbations: the distance between two nearby trajectories increases exponentially with time. This process, quantified by the (maximal) Lyapunov exponent, limits the numerical prediction of such systems since small imprecisions in the initial state will quickly increase to a macroscopic size. This phenomenon, noticed by Lorenz in the first paper on the system and often dubbed the "butterfly effect" is associated to a Lyapunov time which is logarithmic in the precision and sets a limit to numerical simulations with a given precision. This is shown in Fig. 2 in which we represent the distance to a reference trajectory computed with a

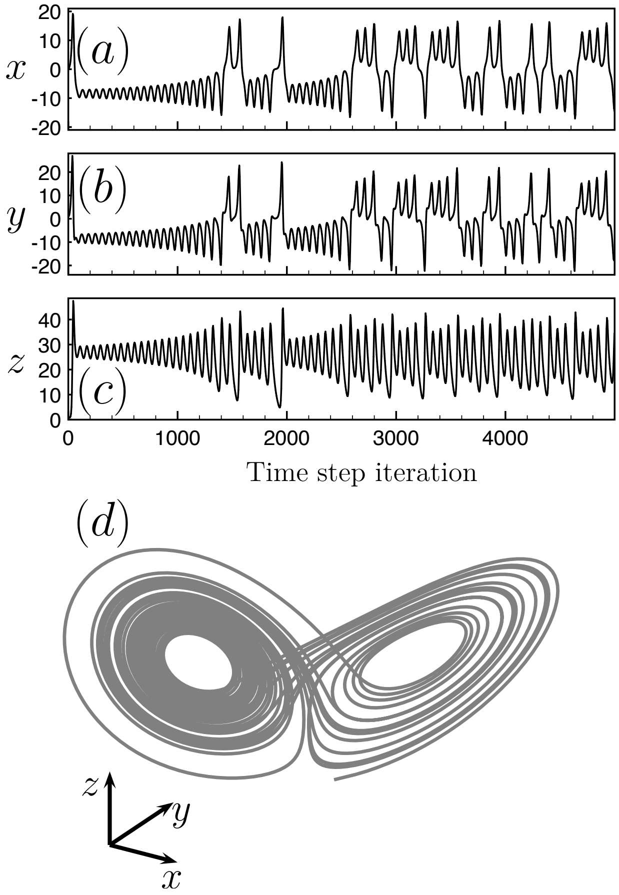

FIG. 1. (a),(b),(c): example of trajectory of the Lorenz system, showing $x(t), y(t)$ and $z(t)$, with a time step $dt = 0.02$; (d): long-time convergence towards the so-called Lorenz attractor. Parameters: $\sigma = 10, \rho = 28$ and $\beta = 8/3$. Initial conditions: $x(0) = 0, y(0) = 1.0$ and z(0) $= 1.05$.

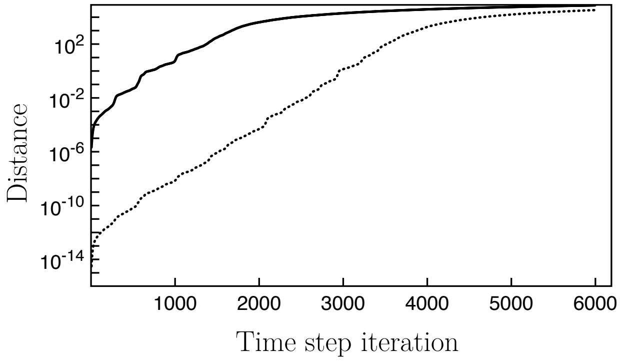

FIG. 2. Euclidean distance between the reference trajectory of the Lorenz system obtained with quadruple precision with the double precision trajectory (dotted line) and the simple precision trajectory (solid line), with a time step $dt = 0.02$.

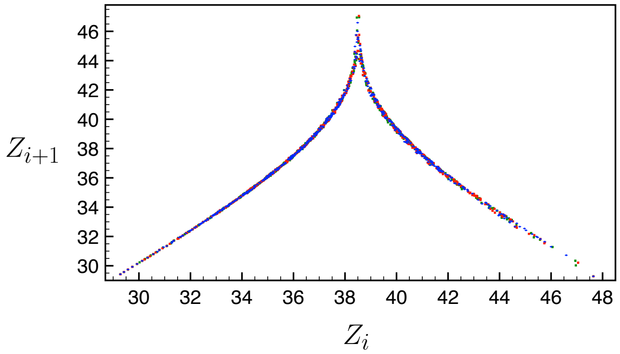

FIG. 3. Comparison of return map of the Lorenz system (long term behavior) with quadruple precision (blue dots) and double (red dots) and simple precision (green dots). The points are nearly superimposed revealing that the long term prediction is almost the same independently of the precision.

Runge-Kutta integration method of order 4 (RK4) in quadruple precision for trajectories computed using RK4 with lower precision (i. e. separated initially by 10^{-16} or 10^{-8}). They strongly depart after a certain time from the high precision trajectory. The separation time clearly increases only logarithmically with the precision.

This property makes numerical simulation of specific trajectories for chaotic systems very difficult: increasing by exponentially large factors the precision only increases linearly the prediction time.

However, one may ask a different type of questions. Even if the short term behavior of a specific trajectory is hard to obtain numerically in a reliable manner, is it still possible to get accurate results on statistical properties of the system for long term? To answer this question, we calculate the first return application. This application, introduced by Lorenz, consists in plotting the successive maxima of $z(t)$ over a long period of time. For that, it is enough to locate the maxima Z_i of the curve and plot the position of a given maxima Z_{i+1} as a function of the preceding one, Z_i. These data are related to the structure of the Lorenz attractor to which trajectories converge for long time. Figure 3 compares such long term predictions using the RK4 algorithm to integrate Eqs. (1)-(3) with different precision. We observe that the statistical properties at long time are not dramatically sensitive to the precision at which the calculation is performed. Even if individual trajectories are not accurately described, their global properties are correctly described. This is similar to what distinguish climate simulations from meteorological simulations: even if individual trajectories cannot be simulated beyond a few weeks to predict the weather, long term global properties of the system (climate characteristics) can be obtained for much longer periods (years or decades).

To evaluate quantitatively the accuracy of long term simulations, we made a polynomial fit of the return map obtained with quadruple precision on each side of the peak of the return map in the window of parameters delimited by the blue dashed lines on the left side and by the red dashed lines on the right side (see Fig. 4). We have then computed the relative error ξ between the fit and the data. The mean percentage error remains below 0.2 % in the zones delimited by the dashed

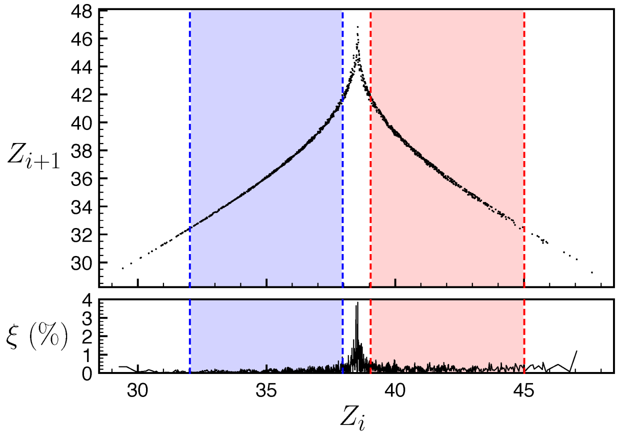

FIG. 4. The return map is calculated using a RK4 integration algorithm in quadruple precision (upper panel). We fit the data in between the blue (red) dashed line with a polynomial of degree 10. We plot the relative difference ξ between the fit and the return map in the lower panel.

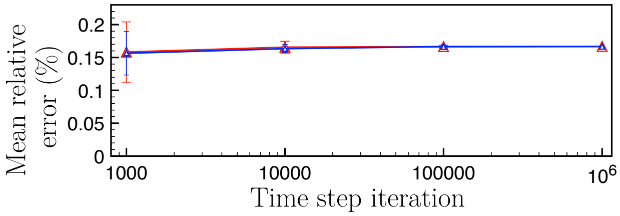

FIG. 5. Mean relative error in percentage in the distance of the return map points (calculated from the RK4 algorithm) from the polynomial fit with time step $dt = 0.02$ (triangle) and for simple precision (large red symbol) and double precision (small blue symbol).

line.

We then compute for simple and double precision the distance towards the fit as a function of the number of iteration points considered (see Fig. 5). The results show that he mean relative error converges to less than 0.2% for sufficiently large databases, in both simple and double precision. The large relative error for a small number of iteration steps is due to the fact that the system has not yet reached the asymptotic behavior of the return map. The data shown in Fig. 5 indicate that the long term prediction characterized here by the return map is almost insensitive to the precision with which the trajectory is computed.

III. RESULTS: ACCURACY OF PREDICTIONS FOR THE LORENZ MODEL

In order to evaluate the accuracy of the machine learning approaches to predict the behavior of the Lorenz system, we use three different methods: a reservoir computing model as pioneered in this context in[4–6], called Echo State Network (ESN) and two other approaches based on Recurrent Neural Networks (RNN) used in[9–12], called LSTM and TCN. The characteristics of the networks we used are detailed in the Appendix.

In this Section, we compare the predictions and performances of each network, focusing especially on the effects of precision of both data and algorithm. Networks are trained on trajectories generated by the RK4 integration method and having thousands of points separated by a time step $dt = 0.02$. Predictions are performed starting immediately after the last point used in the training trajectories. Subsequent predictions are systematically done from the point previously returned by the network.

A. Resources needed for the simulation by the three neural networks

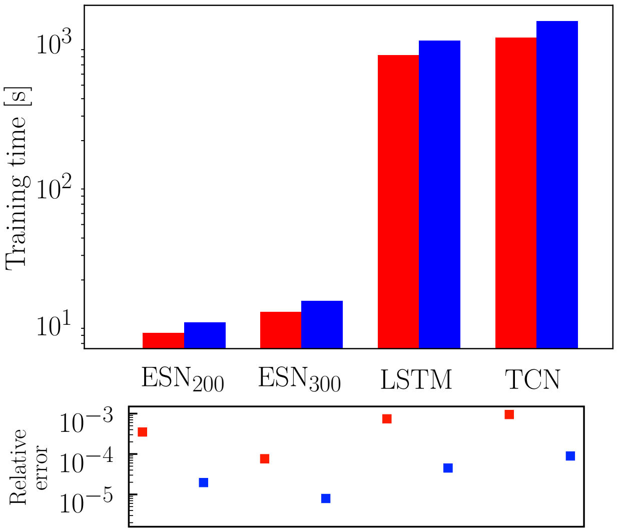

FIG. 6. Upper panel: Comparison of the training time for different neural networks: ESN$_{200}$ (the reservoir contains 200 neurons),ESN$_{300}$, a LSTM network (with a single hidden layer having 64 neurons) and a TCN network. The red (blue) color is used for a computation of the networks parameters in simple (double) precision. Lower panel: Figure of merit of each neural network representing the mean relative error in the estimate of the training trajectories.

Figure 6 gives an overview of the different resources consumed during the training phase by the three networks for achieving a similar converged simulation on the same computer once the network has been set up. It is worth noticing that the performance are for a standard processor. We have not used GPU cards. For the training time, we use the same set of training trajectories (100 trajectories, each trajectory contains 50 000 points separated by a time interval $dt = 0.02$). We compare an ESN with a reservoir size having 200 neurons (ESN$_{200}$), 300 neurons (ESN$_{300}$), a LSTM network (with a single hidden layer having 64 neurons) and a TCN network (similar structure as the LSTM network). Note that the LSTM and TCN are trained 10 times on the training data set while the ESN scans the training data set only once. In addition, the

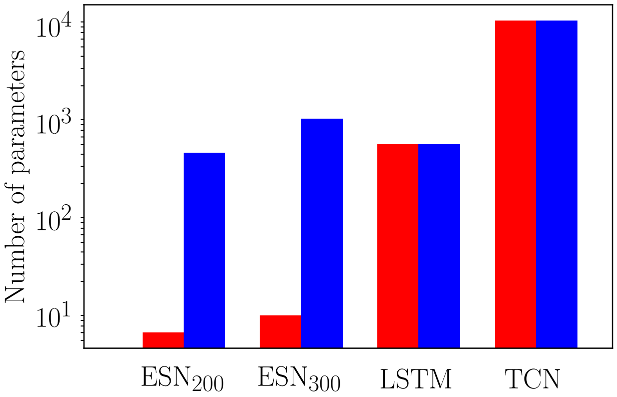

FIG. 7. Comparison of number of parameters for the different neural networks considered in Fig. 6. Red is the training size, blue the total size.

number of parameters that are updated are significantly different depending on the reservoir type as illustrated in Figure. 7. The LSTM and TCN networks adapt themselves by modifying all the network parameters. This is to be contrasted with the ESN that updates only the connections towards the output as discussed in the appendix, making the training size much smaller than the total size.

The figure of merit of each neural network is represented in the lower panel of Figure 6 where we represent the mean relative distance between the trajectories provided by the network compared to the training one. This quantity is here averaged over all the training trajectories. When this relative error is equal to 0.01, it means that the average relative error is on the order of 1 %. As expected, for each neural network the computation of the parameters in double precision yields better results. We also see that the ESN network seems more accurate at reproducing the training trajectory. We conclude that the ESN turns out to be significantly more efficient that the LSTM and TCN networks with respect to the training time and moreover seems to better reproduce the training trajectory.

B. Short term predictions

We now turn to the accuracy of the predictions of the different networks as compared to a quadruple precision simulation by integration of the equations (3).

We first look at short term predictions, i.e. accurate description of a single specific trajectory. That is the type of predictions where chaotic systems are the most difficult to handle. It is similar to meteorological predictions in weather models, since one wants a precise state of the system starting from a specific initial state. We recall that the data are generated via the RK4 method, with a time step of 0.02 and a sampling of thousands of points. Our reference trajectory is calculated in quadruple precision for the same time step and sampling.

A parameter set specific to each network architecture has been established allowing each network to converge. They

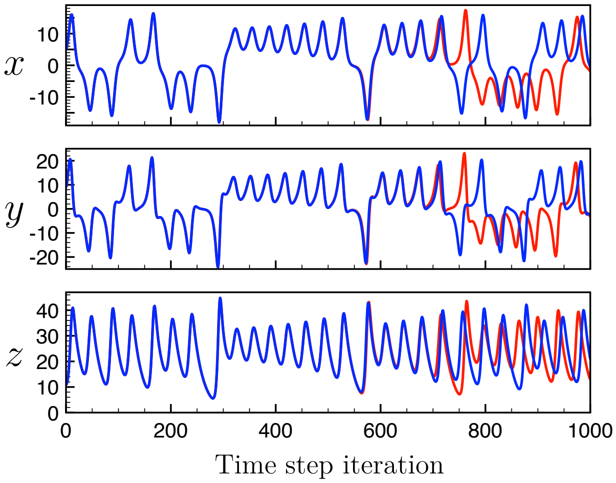

FIG. 8. Comparison between the quadruple precision RK4 simulation (red line) and the prediction of the ESN in double precision with a reservoir of size $N = 300$ (blue line). The initial conditions are $x(0) = 0.00$, $y(0) = 0.45$ and $z(0) = 1.41$, and the time step is $dt = 0.02$. The ESN has been trained over 50000 time step iterations before the prediction for the subsequent iterations represented in this figure.

can be used to predict future points beyond the training set. As said before, the protocol is the same for the three types of networks. The output associated to input vector at time $t = T$ defines the next point for the trajectory at time $T + dt$. This procedure is iterated to get the prediction over large amount of time. We provide an example in Fig. 8 for an ESN neural network which turns out to be able to provide an accurate prediction of the trajectory over the short term for relatively long time.

To be more quantitative, we evaluate for each simulation a limit time, τ_{lim}, defined as the time when the simulation departs from the correct trajectory by at least 5%. This quantity is plotted in Fig. 9 for the three networks considered, as a function of the size of training data (number of points of the exact trajectory which are used to train the network). In all cases, one sees an increase of the limit time with increasing dataset, until it reaches a plateau where increasing the dataset does not help any more. This defines a sort of ultimate limit time for this kind of simulation. All three neworks are effective at predicting the dynamics, giving accurate results for hundreds of time steps. The LSTM and TCN networks give very similar results, and are significantly and systematically less effective than the ESN network used in the seminal paper[5], with prediction times 20% smaller. We recall (see preceding subsection) that the LSTM and TCN networks are not only significantly less effective at predicting the dynamics than the ESN, they are also more costly in resources. The main difficulty for an ESN network is in the search for a viable parameter.

We note that although these neural network methods are effective, they are less efficient than standard classical simulations like RK4 with lower precision (see Fig.2). We should

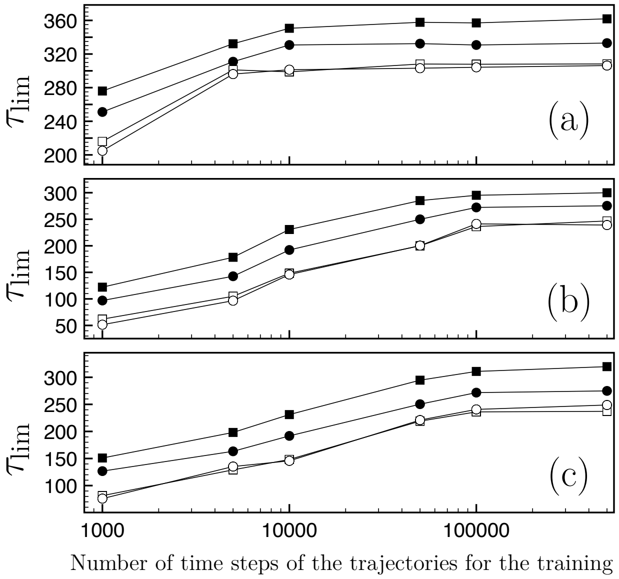

FIG. 9. Impact of the precision of training data and of the neural network on the short term quantified by the time $\tau_{\rm lim}$ above which the prediction departs by more than 5% from the quadruple precision trajectory: (a) ESN, (b) LSTM and (c) TCN. Data and network double precision (filled square), data simple precision and network double precision (filled disk), data double precision and network simple precision (empty square), data and network simple precision (circle). Time step is $dt = 0.02$

note however that neural networks techniques are still new and far from optimized compared to integration methods. In addition, the neural network techniques do not need the equations and do not depend on approximation which can have been used to construct them.

Figure 9 also enables to assess the question of the impact of precision on the predictive abilities of the neural networks. We have changed independently the precision of the datasets used to train the network, and the precision of the network algorithm itself. We see that in all cases the precision of the network will impact the accuracy of the prediction. Indeed, for these short term predictions, a double precision network always give better results than a single precision network. Interestingly enough, with a single precision network, increasing the precision of the training data does not help. On the other hand, using a double precision network even on single precision data is more advantageous than a single precision network on any type of data. These results are valid for the three types of networks over the full range of training sets used. It therefore seems that the precision of the network is crucial for the accuracy of the prediction, and more so than the precision of the data. It is especially important in view of the fact that the precision of the data can be less easily controlled than the precision of the network.

C. Long term predictions

We now turn to long term predictions, i.e. accurate description of a statistical quantities corresponding to many trajectories, as opposed to a single specific trajectory.

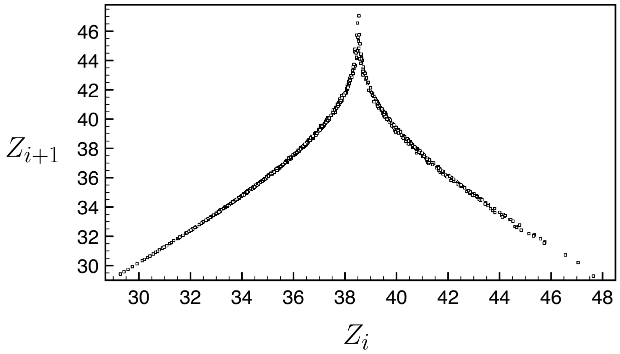

FIG. 10. Return map of the Lorenz system obtained by an ESN network simulation.

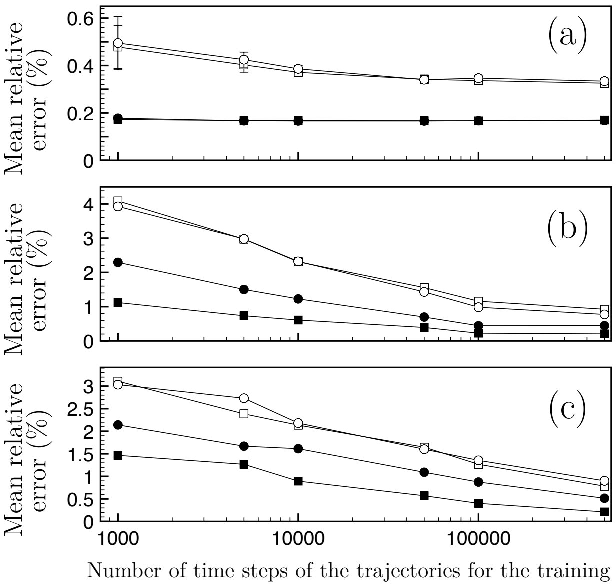

FIG. 11. Impact of precision of training data and network type on the precision of the simulation for long times. Same notations as for Fig. 9. Time step is $dt = 0.02$

Figure 10 displays an example of return map constructed from ESN predictions, showing that, despite the fact that individual trajectories are not accurately simulated for long times, the long time dynamics is correctly described, giving the general shape of the return map.

To be more quantitative, Fig. 11 uses the measure defined in Section II (see Fig. 4) to assess the efficiency of the neural network methods for long term dynamics. Despite the fact that the LSTM and TCN networks are more cumbersome to implement and take more running time, the results are clearly better for the ESN network, which can achieve an accuracy similar to the one of the RK4 simulations (see Fig. 11a). For the LSTM and TCN networks, the results presented in Figs. 11b and c show that these networks are able to reproduce the long term dynamics, but the accuracy is less good than for ESN networks or RK4, even for large sizes of the training dataset.

As in the case of short term predictions, the results presented in Figs. 11 allow us to estimate the effects of the precision on long term predictions. For the ESN network, it is clear that the precision of the results is entirely controlled by the precision of the network, independently of the precision of the training data. For LSTM and TCN networks, we can see an effect of the precision of both the network and the data, but in all cases the precision of the network is the dominant factor: even with low precision data, the high precision network fares better than a low precision network with high precision data.

IV. CONCLUSION

The results presented in this paper confirm previous works, showing that neural networks are able to simulate chaotic systems, both for short term and long term predictions. We also show that the ESN network (reservoir computing) seems globally more efficient in this task than LSTM or TCN networks, in line with the recent work[11]. Our investigations allow to assess the effect of the precision of the training data and precision of the network on the accuracy of the results. Our results show than in a very consistent manner, the precision of the network matters more than the precision of the data on which it is trained. In view of the exponential instability of chaotic dynamics, where small errors are exponentially amplified by the dynamics, this may seem surprising. However, this is good news for practical applications, such as meteorology or climate simulations: the precision of the dataset is in many instances given by the precision of observations, that may be hard to ameliorate, while the precision of the network is controlled at the level of the algorithm used and may be increased at a cost of more computing time.

ACKNOWLEDGMENTS

We thank Gael Reinaudi for discussions. We thank CalMiP for access to its supercomputer. This study has been supported through the EUR grant NanoX ANR-17-EURE-0009 in the framework of the "Programme des Investissements d'Avenir". The data that support the findings of this study are available from the corresponding author upon reasonable request.

Appendix A: The three machine learning approaches used

In the Appendix, we give an overview of the main features of the three neural networks that have been used in the article, namely the ESN, LSTM and TCN networks.

1. Reservoir computing: ESN network

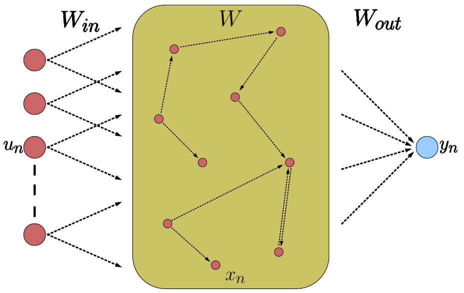

FIG. 12. Schematic representation of an Echo State Network (ESN).

The first network we use corresponds to reservoir computing. We focus on a specific model called Echo State Networks (ESN). Reservoir computing methods were developed in the context of computational neuroscience to mimick the processes of the brain. Their success in machine learning comes from the fact that they are relatively cheap in computing time and have a simple structure. Their complexities lie in their training and the adjustment of parameters to obtain the desired results. The structure of ESN networks is schematized on Fig. 12.

To train our ESN on a time-dependent signal u_n with $n = 1, ..., T$ where T is the duration of the sequence in discretized time, we must minimize a cost function between y_n^{ref} and y_n. Here y_n^{ref} is the output that we want to obtain with u_n, and y_n is the output of the network when we give it u_n as input. For the Lorenz problem, u_n, y_n^{ref} and y_n are 3D vector. Generally, the cost function that one seeks to minimize is the error between the output of the network and the reference signal. This function is often in the form of a mean square error or, in our case, of the mean standard deviation.

The output of the network is calculated as follows:

$$y_n = W_{out}[1; u_n; x_n], \tag{A1}$$

where W_{out} is the output weight matrix that we are trying to train, $[.;.]$ represents the concatenation, u_n is our vectorial input signal and x_n the vector corresponding to the reservoir neuron activations. It has the dimension N of the reservoir and is calculated as follows:

$$x_n = (1-\alpha)x_{n-1} + \alpha\tilde{x}_n \tag{A2}$$

with $\tilde{x}_n$ corresponding to the new value of x_n :

$$\tilde{x}_n = \tanh(W_{in}[1; u_n] + W x_{n-1} + \varepsilon_0 + \mu_0) \tag{A3}$$

α is the leaking rate, $\varepsilon_0 = -1.154$ is an offset optimized on our set of data, μ_0 is a random Gaussian variable of standard deviation equal to 2.25×10^{-5}, W is the system reservoir and W_{in} is the input weight matrix of the reservoir. The dimension of W_{in} is $N \times (3+3)$ the $+3$ term accounts for the bias added to the input (see Fig. 12). The initialization consists in setting x and y to zero.

There are several important parameters that must be adjusted depending on the problem we are studying if we want our ESN to be able to predict our system. The first parameter we can play on is the size of the reservoir itself. The more complex the problem that we want to deal with, the more the size of the reservoir will have an impact on the capacities of the network. A large reservoir will generally give better results than a small reservoir. Once the size of our reservoir has been chosen, we can play on the central parameter of an ESN: the spectral radius of the reservoir. Often denoted by $\rho(W)$, this is the maximal absolute value of eigenvalues of the matrix W. The spectral radius determines how quickly the influence of an input data dissipates in the reservoir over time. If the problem being treated has no long-term dependency, there is no need to keep data sent far in advance. We can therefore ensure that the spectral radius is unitary. In some cases, if our problem has long-term dependencies, it is possible to have $\rho(W) > 1$ to keep the data sent in the network longer. The last parameter we can play on is the leaking rate α. It characterizes the speed at which we come to update our reservoir with the new data that we provide over time.

The matrices W and W_{in} are initialized at the start but are not modified during training. Only the output matrix W_{out} is driven:

$$W_{out} = Y^{ref} X^T (XX^T + \beta I)^{-1} \tag{A4}$$

where, for our Lorenz problem, $X = (x_1, ..., x_T)$ (dimension $N \times T$), $Y^{ref} = (y_1^{ref}, ..., y_T^{ref})$ (dimension $3 \times T$) and I is the $N \times N$ identity matrix. As a result, the dimension of W_{out} is $3 \times N + 4$.

The fact that connectivities from input to hidden layer and from hidden to hidden layer are fixed and randomly assigned from the beginning reduces considerably the number of parameters to be trained. As a result, the training speed of the network is small compared to other networks specialized in learning specific temporal patterns (see below). By increasing the size of the training data, the network becomes more sensitive to the small fluctuations that accumulate during W_{out} calculation. The parameter β makes it possible to limit this dependence by penalizing the too large values of W_{out}. This is all the more true for a single precision network which is more sensitive to these fluctuations and whose β must vary by several orders of magnitude depending on the size of the training data. In double precision (float64), β varies from 10^{-8} for 5000 training points to 10^{-7} for 5.10^5 training points against 10^{-4} to 10^{-1} in simple precision (float32). As the reservoir is not changed during training, one must choose the initialization hyperparameters to ensure that one has a consistent output with the expected results. This requires adjusting the values of the leaking rate, spectral radius and input scaling as a priority. The optimization of these parameters has been done via a grid search where we decrease the search step as we find good parameters.

The initialization parameters are for W_{in} a density equal to $d = 0.464$ with values randomly distributed from a Gaussian function of standard deviation equal to $\sigma = 3.352$. For the reservoir matrix W, we have chosen $d_W = 0.483$, $\sigma_W = 2.901$, and a spectral radius $\rho_W = 0.625$.

2. LSTM and TCN networks

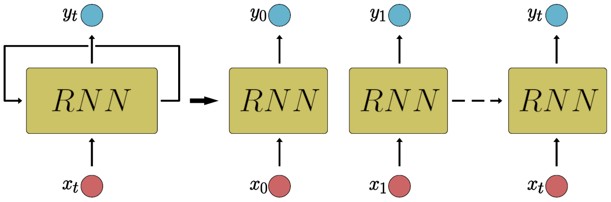

FIG. 13. General structure of Recurrent Neural Networks (RNN).

The two other networks we use are based on Recurrent Neural Network architectures (RNN). RNNs can be represented as a single module chain (see Fig. 13). The length of this chain depends on the length of the sequence that is sent to the input. The output of the previous module serves as input for the next module in addition to the data on which we train our network. This allows the network to keep in memory what has been sent previously.

The major problem in this type of network is the exponential decrease of the gradient during the training of the network. This is due to the nature of back-propagation of the error in the network. The gradient is the value calculated to adjust the weights in the network, allowing the network to learn. The larger the gradient, the greater the adjustments in the weights, and vice versa. When applying back-propagation to the network, each layer calculates its gradient from the effect of the gradient in the previous layer. If the gradient in the previous layer is small, then the gradient in the current layer will be even weaker. The first layers in the network therefore have almost no de facto adjustment in their weight matrices for a very small gradient.

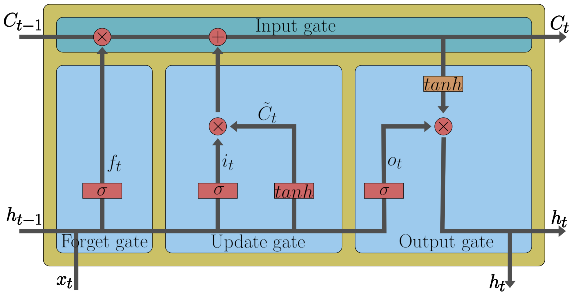

FIG. 14. Schematic representation of a Long Short Term Memory (LSTM) network: structure of one elementary cell.

To solve this problem of attenuation of the corrections, the LSTM networks (Long Short Term Memory networks) have been explicitly developed for this purpose. They can also be represented as a module chain, but unlike conventional RNNs, they have a more complex internal structure, composed of four layers which interact with each other (see Fig. 14).

The first layer is called "input gate", and is represented by a horizontal line that runs through the entire cell. It allows data to easily browse the entire network. This structure represents the state of the cell over time. On this line there are other structures which will be used to modify the data which go through the cell.

The next step in our network is the forget gate structure. It consists of a neural network with an activation function of the sigmoid type and makes it possible to decide which part will be kept in the cell:

$$f_t = \sigma(W_f[h_{t-1}, u_t] + b_f), \tag{A5}$$

where W_f and b_f are the weights and bias of the network for the update gate layer, u_t is the input data at time t and h_{t-1} is the hidden state output by the previous cell.

The second step is to decide what to store. This structure consists of two parts. The first part is a neural network with an activation function of the sigmoid type, and will allow us to choose which value will be updated in our structure:

$$i_t = \sigma(W_i[h_{t-1}, u_t] + b_i), \tag{A6}$$

where W_i and b_i are the weights and bias of the sigmoid network for the update gate layer. u_t is the input data at time step t and h_{t-1} is the hidden state output by the previous cell. The second part is another neural network with this time an activation function of the hyperbolic tangent type and that allows to create the new state candidate of our vector C_t as follows:

$$\tilde{C_t} = \tanh(W_c[h_{t-1}, u_t] + b_c), \tag{A7}$$

where W_c and b_c are the weights and bias of the hyperbolic tangent network for the update gate layer, u_t is the input data at time t and h_{t-1} is the hidden state output by the previous cell. The new cell state C_t is then computed by adding the different outputs from the input gate, the forget gate and the update gate as follows:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C_t}, \tag{A8}$$

where f_t is the output of the forget gate layer, i_t is the input layer choosing which values are going to be updated, $\tilde{C_t}$ is the new cell state candidate, C_{t-1} is the cell state of the previous cell and $*$ denote elementwise operation. The structure described above is then repeated from cell to cell.

A final structure makes it possible to determine what will be the output of the network. The output will be based on the state of the cell to which we have applied a network with a sigmoid activation function to choose which part will be returned. Then we apply a hyberbolic tangent function to the cell state and multiply it with the previous value to get the new cell state output:

$$o_t = \sigma(W_o[h_{t-1}, u_t] + b_o), \tag{A9}$$

$$h_t = o_t * \tanh(C_t). \tag{A10}$$

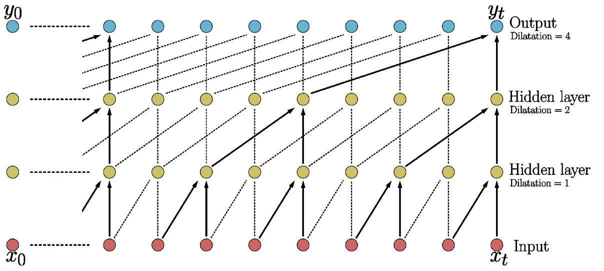

FIG. 15. Schematic representation of a Temporal Convolutional Network (TCN).

h_t is then sent into a linear layer for prediction of y_t.

The third architecture we use consists in TCN networks[14], which use causal convolutions, meaning that at time step t, the network can only access data points at time step t and earlier, ensuring no information leakage from future to past (see Fig. 15). The ability of causal convolution to keep previous data points in memory depends linearly on the depth of the network. This is why we are using dilated convolution to model long term dependencies of our system as shown in[16] as it enables an exponentially large receptive field depending on the depth of the network. This enables TCN to function in a way similar to RNN. For an input sequence $\mathbf{U}$ of size T (with elements u_n), the dilated causal convolution H we use is defined as

$$H(u)_n = (U *_d h)(u)_n = \sum_{i=0}^{k-1} h(i) u_{n-d.i} \tag{A11}$$

where d is the dilatation factor, h is a filter $\in \mathbb{R}^{k-1}$ where k is the filter size and the indices $n - d.i$ represents the direction of the past. Using larger dilatation factor enables an output at the top level to represent a wider range of inputs, thus effectively expanding the receptive field of the network.

The LSTM and TCN networks are more complex networks and more demanding in computation than ESN. We have set up these networks with the Tensorflow library. For a trajectory made of N_i time step iterations, we use 35 successive points of the trajectory to predict the next step. In this way, we build a predicting vector of dimension $N_i - 35$. We use batches of 32 successive values of this vector to update the network parameters with the gradient back propagation algorithm (using the Adam optimizer with an exponential learning rate decay). This process is performed over all the values of the predicting vector, and repeated 10 times (number of epochs equal to 10). One has indeed to make several passes on the training data to get good results. On average, an epoch takes 30 seconds. Testing the different possible architectures therefore takes more time than for the ESN.

[1] D. Silver et al., *Mastering the game of Go with deep neural networks and tree search* Nature **529**, 484 (2016).

[2] G. Hinton et al., *Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups*, IEEE Signal Process. Mag. **29**, 82 (2012).

[3] Y. Wu et al *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*, preprint arXiv:1609.08144 (2016).

[4] J. Pathak, Z. Lu, B.R. Hunt, M. Girvan, E. Ott, *Using machine learning to replicate chaotic attractors and calculateLyapunov exponents from data* Chaos **27**, 121102 (2017).

[5] J. Pathak, B.R. 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**, 024102 (2018).

[6] Z. Lu, B.R. Hunt and E. Ott, *Attractor reconstruction by machine learning*, Chaos **28**, 061104 (2018).

[7] M. Lukosevicius and H. Jaeger, *Reservoir computing approaches to recurrent neural network training*, Comput. Sci. Rev. **3**, 127 (2009).

[8] Yang Tang, Jürgen Kurths, Wei Lin, Edward Ott, and Ljupco Kocarev, *Introduction to Focus Issue: When machine learning meets complex systems: Networks, chaos, and nonlinear dynamics*, Chaos **30**, 063151 (2020).

[9] P. R. Vlachas, W. Byeon, Z. Y. Wan, T. P. Sapsis and P. Koumoutsakos, *Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks*, Proc. R. Soc. A **474**, 20170844 (2018).

[10] P. G. Breen, C. N. Foley, T. Boekholt, S. P. Zwart, *Newton versus the machine: solving the chaotic three-body problem using deep neural networks* Monthly Notices of the Royal Astronomical Society **494**, 2465 (2020)

[11] P.R. Vlachas, J. Pathak, B. R. Hunt, T.P. Sapsis, M. Girvan, E. Ott and P. Koumoutsakos, *Backpropagation Algorithms and Reservoir Computing in Recurrent Neural Networks for the Forecasting of Complex Spatiotemporal Dynamics*, Neural Networks **126**, 191 (2020).

[12] A. Chattopadhyay, P. Hassanzadeh, and D. Subramanian, *Data-driven prediction of a multi-scale Lorenz 96 chaotic system using deep learning methods: Reservoir computing, ANN, and RNN-LSTM* preprint at https://www.nonlin-processes-geophys-discuss.net/npg-2019-61/

[13] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, J. Schmidhuber, *LSTM: A Search Space Odyssey* IEEE Transactions on Neural Networks and Learning Systems **28** , 10 (2017).

[14] S. Bai, J.Z. Kolter, V. Koltun *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling* arXiv preprint arXiv:1803.01271 (2018).

[15] E. N. Lorenz, *Deterministic Nonperiodic Flow*, J. Atmos. Sci. **20**, 130 (1963).

[16] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, *WaveNet: A Generative Model for Raw Audio* preprint arXiv:1609.03499 (2016).