Data-driven predictions of the Lorenz system

Pierre Dubois, Thomas Gomez, Laurent Planckaert, Laurent Perret

Data-driven predictions of the Lorenz system

Pierre Dubois[a,*], Thomas Gomez[a], Laurent Planckaert[a], Laurent Perret[b]

[a] *Univ. Lille, CNRS, ONERA, Arts et Metiers Institute of Technology, Centrale Lille, UMR 9014 - LMFL - Laboratoire de Mécanique des fluides de Lille - Kampé de Fériet, F-59000 Lille, France*
[b] *Centrale Nantes, LHEEA UMR CNRS 6598, Nantes, France*

Abstract

This paper investigates the use of a data-driven method to model the dynamics of the chaotic Lorenz system. An architecture based on a recurrent neural network with long and short term dependencies predicts multiple time steps ahead the position and velocity of a particle using a sequence of past states as input. To account for modeling errors and make a continuous forecast, a dense artificial neural network assimilates online data to detect and update wrong predictions such as non-relevant switchings between lobes. The data-driven strategy leads to good prediction scores and does not require statistics of errors to be known, thus providing significant benefits compared to a simple Kalman filter update.

Keywords: data-driven modeling, data assimilation, chaotic system, neural networks

1. Introduction

Chaotic dynamical systems exhibit characteristics (nonlinearities, boundedness, initial condition sensitivity) [1] encountered in real-world problems such as meteorology [2] and oceanography [3]. The multiple time steps ahead prediction of such a system is challenging because governing equations may be unknown or too costly to evaluate. For instance, the Navier Stokes equations require prohibitive computational resources to predict with great accuracy the velocity field of a turbulent flow [4].

Data-driven modeling of dynamical systems is an active research field whose objective is to infer dynamics from data [5]. Regressive methods in machine learning [6] are particularly suitable for such tasks and have proven to reliably reconstruct the state of a given system [7]. If parameters are not overfitted to training examples, the data-driven model can also be used for predictive tasks, providing the input lies in the input domain used for training. Main techniques in the literature include autoregressive techniques [8], dynamical mode decomposition (DMD) [9], Hankel alternative view of Koopman (HAVOK) [10] or unsupervised methods such as CROM [11]. Neural networks are also of increasing interest since they can perform nonlinear regressions that are fast to evaluate. Architectures with recurrent units are recommended for time-series predictions because memory is incorporated in the prediction process. Neural networks can then learn chaotic dynamics [12] and predict with great accuracy the future state [13].

However, errors in modeling can lead to bad multiple time steps ahead predictions of chaotic dynamical systems: a tiny change in the initial condition results in a big change in the output [12]. To overcome the propagation of uncertainties from the dynamical model (bad regression choice in a data-driven approach or bad turbulence modeling in CFD for instance) data assimilation (DA) techniques have been developed [14].

*Corresponding author: pierre.dubois@onera.fr

They combine the predicted state of a system with online measurements to get an updated state. Such methods have successfully been applied in fluid mechanics to obtain a better description of initial or boundary conditions by finding the best compromise between experimental measurements and CFD predictions [15]. Nevertheless, the dynamical model can be slow to evaluate (limiting the use to offline assimilations) and errors (initial condition, dynamical model, measurements and uncertainties) can be hard to estimate in real-world applications.

In this paper, a data-driven approach is used to discover a dynamical model for the Lorenz system. To handle the chaotic nature of the system, a recurrent neural network (RNN) dealing with long and short term dependencies (LSTM) is considered [16]. To correct modeling errors, a dense neural network (denoted hereafter DAN) whose design is based on Kalman filtering techniques is developed. Results are promising for predicting multiple steps ahead the position and velocity of a particle on the Lorenz attractor, using only the initial sequence and real-time measurements of the complete acceleration, the complete velocity or a single component of the velocity.

The paper is organized as follows. In Section 2, the overall strategy is presented with a quick understanding of how neural networks work. In Section 3, results about the low dimensional Lorenz system are shown, with a particular interest in the impact of forecast horizon and noise. A discussion is given in Section 4 before giving concluding remarks.

2. Strategy

2.1. Proposed methodology

This paper investigates the use of neural networks to continuously predict a chaotic system using a data-driven dynamical model and online measurements. The method is summarized in Figure 1 and contains the following steps:

- ▷ Consider m temporal states of the system. The sequence is denoted $[\mathbf{s}]_{t-m-1}^{t}$ where $\mathbf{s}$ is the state of the system and whose dimension is n_f.
- ▷ Predict n future states using a RNN with long and short-term memory (LSTM). This gives a predicted sequence $[\mathbf{s}^b]_{t+1}^{t+n}$ where superscript b indicates a prediction.
- ▷ Predict the measured sequence. This gives $[\mathbf{y}^b]_{t+1}^{t+n}$ where $\mathbf{y}^b$ is the predicted measure of the state. The mapping between the state space and the measurement space is performed by a dense neural network called the shallow encoder (SE).
- ▷ Assimilate the exact sequence of measurements $[\mathbf{y}]_{t+1}^{t+n}$ and update the predicted sequence of states. This work is performed by a dense neural network which gives an updated sequence $[\mathbf{s}^a]_{t+1}^{t+n}$ where superscript a stands for "analyzed". The network is called the data assimilation network (DAN).
- ▷ Construct $[\mathbf{s}^a]_{t+n-m+1}^{t+n}$ by adding $m-n$ updated states from the previous iteration. This gives a new input that can be used to cycle and continue the forecasting process.

In this section, we give a quick overview of neural networks and explain architectures behind the dynamical model (RNN-LSTM), the measurement operator (SE) and the data assimilation process (DAN).

2.2. Quick overview of neural networks

A neuron is a unit passing a sum of weighted inputs through an activation function that introduce nonlinearities. These functions are classically a sigmoid $\sigma(x) = \dfrac{1}{1+e^{-x}}$, a hyperbolic tangent $\tanh(x)$ or a rectified linear unit $\text{relu}(x) = \max(0, x)$. When neurons are organized in fully connected layers, the resulting network is called a dense neural network. The universal approximation theorem [17] states that any function can be approximated by a sufficiently large network i.e. one hidden layer with a large number of neurons. Just like a linear regression $y = ax + b$ aims

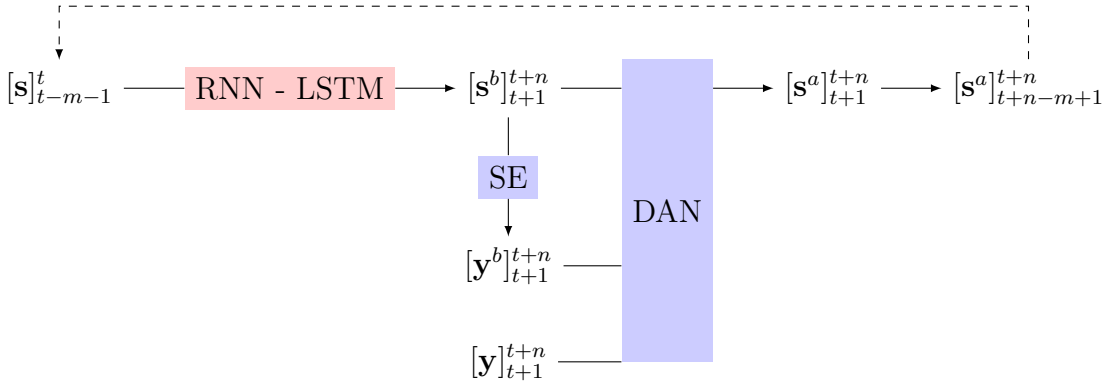


Figure 1: Summary of the data-driven method to make predictions of a chaotic system. A data-driven dynamical model (RNN-LSTM) predicts n future states of the system and the predicted sequence is updated according to a real sequence of measurements.

at learning the best a and b parameters, a neural network regression $y = NN(x)$ aims at learning the best weights and biases in the network by optimizing a loss function evaluated on a set of training data.

Although they are universal approximators, dense neural networks face some limitations: they may suffer from vanishing or exploding gradient (arising from derivatives of activation functions, see [18]), are prone to overfitting (fitting that corresponds too much to training data) and inputs are not individually processed. Other architectures of artificial neural networks have then been developed, including convolutional networks (CNN, for image recognition) or recurrent neural networks (RNN, inputs are taken sequentially). Recurrent networks use their internal state (denoted h) to process each input from the sequence of inputs. This internal state is computed using an activation function but to avoid limitations from dense networks, its form is more elaborate. For example, Long Short-Term Memory (LSTM) cells [19] are combinations of classical activation functions (sigmoids and tanh) that incorparate a long and short term memory mechanism through the cell state (see Figure 2).

Several techniques exist to learn parameters in neural networks. The most common is the gradient descent, which iteratively update parameters according to the gradient of the cost function with respect to weights and biases. The computation of gradients is made by backpropagating errors in the network, using backpropagation for dense neural networks or backpropagation through time for RNN [6]. The equations can be found in [20] for the curious reader. In this paper, all neural networks are implemented using the Keras library [21].

In this paper, hyperparameters are not tuned. No grid search or genetic optimization is intended and number of neurons, number of hidden layers and activation functions are found by successive trials. Defined architectures must not then be considered as a rule of thumb.

2.3. Novelty of the work

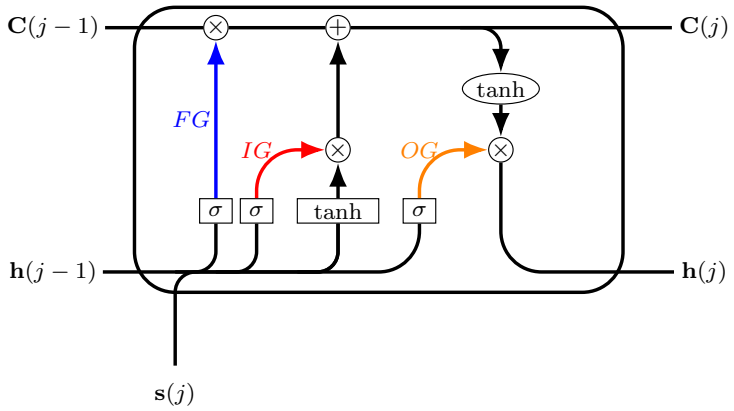
This paper proposes a regressive framework for assimilating data as opposed to standard data assimilation techniques whose architecture does not depend on the problem. Besides, the present paper considers time marching of an entire sequence of the state while the most standard approaches involve a time marching of the predicted state at regular time units. More details about existing works are given in Section 4.

2.4. Dynamical model

The first step is to establish a dynamical model mapping m previous states $\mathbf{s}(t)$ to n future states. The chosen architecture is summarized in Figure

(a) RNN

(b) RNN - LSTM



(c) LSTM cell. The recurrent unit is composed of a cell state and gating mechanisms. The cell state C is modified when fed with a new time step from the input sequence, forgetting past information (via Forget Gate FG), storing new information (via Input Gate IG) and creating a short-memory (via Output Gate OG). Mathematical details are given in the appendix.

Figure 2: Two types of recurrent neural networks: simple RNN handling short-term depedencies via a hidden state $\mathbf{h}$ (subfigure a) and RNN-LSTM handling short and long-term depedencies via a hidden state $\mathbf{h}$, a cell state $\mathbf{C}$ and gating mechanisms (subfigures b and c). Each time step $\mathbf{s}(j)$ from the input sequence is combined with $\mathbf{h}(j-1)$ (and $\mathbf{C}(j-1)$ for LSTM-RNN) which was (were) computed at previous time step.

3. In the recurent layer, $2m$ LSTM cells (making the cell state a $2m$ dimensional vector) processes the input sequence $[\mathbf{s}]_{t-m+1}^{t}$. This results in a final output $o(t) = h(t)$ summarizing all relevant information from the input sequence. In dense layers, the final output from the recurent layer is used to predict n future states $[\mathbf{s}^b]_{t+1}^{t+n}$ Concerning the the number of recurent units, it has been chosen to echo results of Faqih et al. [1] where best scores were obtained by considering twice as many neurons than the history window. Authors made this conclusion after trying to predict multiple steps ahead the state of the Lorenz 63 system using a dense neural network with radial basis functions. About the training of the model, the procedure is as follows:

1. Simulate the system to get data $t \to \mathbf{s}(t)$. For the considered Lorenz system, only one trajectory is simulated but it covers a good region of the phase space.
2. Split data into training and testing sets. In this work, 2/3 of the data is used for the

training and 1/3 is used for the testing. To detect possible overfitting, cross validation is first performed by considering 20% of training data as validation data. Depending on the evolution of both training and validation errors, dropout layers can be added to neural networks. The model is then trained on the whole training data set and errors are computed on the yet unseen testing data set. This step is necessary to ensure that test errors are representative of generalization errors.

3. Define supervised problems by writing data as $[\mathbf{s}]_{t-m+1}^{t} \rightarrow [\mathbf{s}]_{t+1}^{t+n}$. The number of training examples is increased by considering a sliding window of one time step i.e. training set is composed of $[\mathbf{s}]_{0}^{m-1} \rightarrow [\mathbf{s}]_{m}^{m+n-1}$, $[\mathbf{s}]_{1}^{m} \rightarrow [\mathbf{s}]_{m+1}^{m+n}$, etc. For the testing set, a sliding window of n time steps is used.
4. Find optimal weights and biases in the network by minimizing the mean square error evaluated on batches of training data. The chosen optimization algorithm is ADAM [22]. They are numerous parameters to find, including all weights and biases for each LSTM cell in the reccurent architecture and all parameters in dense layers. During the optimization process, the mean square error is also computed on the validation set. To avoid overfitting and ensure that weights and biases learned during training are relevant for future use on test set, errors evaluated on training and validation sets should be close.
5. Evaluate the performance of the final model using test data. Test 1 uses exact input sequences $[\mathbf{s}]_{t-m+1}^{t}$ to compute $[\mathbf{s}^b]_{t+1}^{t+n}$. Test 2 uses the first exact sequence $[\mathbf{s}]_{0}^{m-1}$ to compute $[\mathbf{s}^b]_{m}^{m+n-1}$ which is used as a new input and so on. The metric to quantify errors is the normalized mean square error which indicates how far predictions are from expectations on average. It is computed at the end of the process, using temporal testing states as a reference. Its formula is based on all predicted states $\mathbf{s}^b$ and corresponding true states $\mathbf{s}$:

$$NMSE = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} \frac{||\mathbf{s}(t_i) - \mathbf{s}^b(t_i)||^2}{||\mathbf{s}(t_i)||^2}$$

Where n_{test} is the total number of test states and $||.||$ is the l_2 operator to compute the norm of the considered vector (dimension n_f).

2.5. Data assimilation

To make a continuous forecast of the state using a data-driven dynamical model, it is necessary to limit the accumulation of prediction errors [23] by incorporating online data in the prediction process. Consider $\mathbf{y}(t)$ an exact measurement of the state at t. The mapping between the state space and the measurement space is done using the measurement operator H. In Kalman filtering techniques, a predicted state $\mathbf{s}^b(t)$ is updated according to $\mathbf{s}^a(t) = \mathbf{s}^b(t) + K_t[\mathbf{y}(t) - H(\mathbf{s}^b(t))]$ where the Kalman gain K_t blends errors from the prediction and the measurement. Such a method is based on the Bayes theorem which helps to compute the density probability function of the state conditioned by the measurement. However, these techniques require statistics of errors to explicitly be known and work on a sequence of states only when considering the sequence as the state. The objective here is to adapt the strategy to directly update, in a regressive manner, a sequence of states using a sequence of measurements.

The first stage is to establish a relationship between the state and its measurement i.e. find an approximation of H operator. This task is performed by a shallow encoder which nonlinearly explains a measurement by its state. Figure 4 summarizes the retained architecture, with n_f describing the number of features in the state and p being the number of observed variables. For simple and known mappings, this step is not necessary. However, to develop a complete data-driven framework, we choose to keep the shallow encoder despite the simplicity of the true measurement operator for the considered Lorenz system.

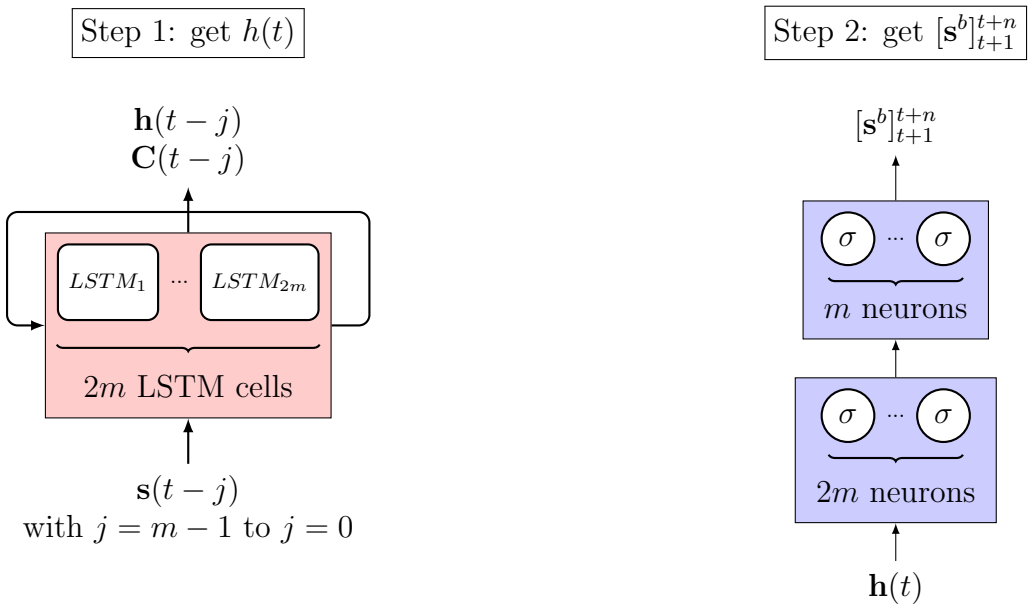


Figure 3: Architecture of the dynamical model, a network composed of a recurrent layers and dense layers. Idea behind the design: $2m$ cells are used to echo the results obtained in [1] where best prediction scores were obtained when considering two times the history window.

The training and testing of the shallow encoder is performed using data $\mathbf{s}(t) \rightarrow \mathbf{y}(t)$. The determination coefficient R^2 is used as a metric to assess the quality of the regression. It is defined by:

$$R^2 = 1 - \frac{\sum_{i=1}^{n_s} ||\mathbf{y}(t_i) - \mathbf{y}^b(t_i)||^2}{\sum_{i=1}^{n_s} ||\mathbf{y}(t_i) - \bar{\mathbf{y}}||^2}$$

Where n_s is the number of samples (n_{train} for assessing quality of the fit, n_{test} for assessing the quality of prediction) and $\bar{\mathbf{y}}$ is the temporal mean measurement vector.

The second stage is to blend a predicted sequence $[\mathbf{s}^b]_{t+1}^{t+n}$ with its associated sequence of measurements $[\mathbf{y}^b]_{t+1}^{t+n}$ and the real sequence of measurements $[\mathbf{y}]_{t+1}^{t+n}$ to produce the updated sequence $[\mathbf{s}^a]_{t+1}^{t+n}$. This job is done by a dense neural network whose architecture is summarized in Figure 5. The training process is as follows:

1. Simulate the system to get $t \rightarrow \mathbf{s}(t), \mathbf{y}(t)$. Measurements are supposed to be exact and no noise is applied yet.
2. Split temporal states and measurements into training and testing sets. The procedure is the same than for the dynamical model training preparation.
3. Get supervised formulation of training data. The sliding window is supposed to be n. Outputs sequences are $[\mathbf{s}, \mathbf{y}]_{jn}^{(j+1)n+m-1}$ where index j describe the j-th output.
4. Perform test 1 (see subsection 2.4) and shallow encoder on defined outputs. This leads to possible predicted states and measurements $[\mathbf{s}^b, \mathbf{y}^b]_{jn}^{(j+1)n+m-1}$.
5. Each training measurement sequence $[\mathbf{y}]_{t+1}^{t+n}$ is associated to 20% random pairs from the set defined in step 4. In doing so, a real sequence of measurement is associated to randomly selected 20% of all possible predictions in order to produce sequence $[\mathbf{s}]_{t+1}^{t+n}$. All possible predictions could be used but this would increase the computational time to prepare the training set.
6. Perform gradient descent to optimize weights and biases.
7. Evaluate the performance of the final model using test data. This is denoted as Test 3 which is summarized in Figure 6. Like in Test 2, inputs are fed back in recursively to assess the continuous prediction of test

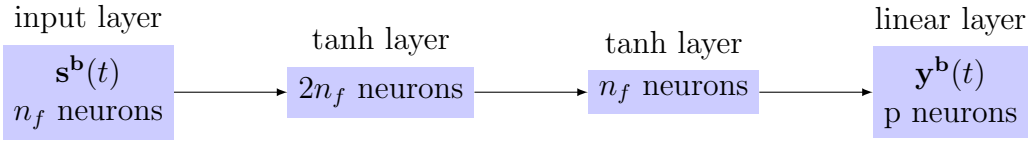


Figure 4: Shallow artificial neural network to map a state to its measurement i.e. approximation of H operator.

data. The metric is still the normalized mean square error.

The assimilation technique proposed here is a nonlinear regression learned on training data. This is different from Kalman filtering techniques where the Kalman gain only relies on statistics of errors and whose formula does not depend on the considered case.

3. Results

3.1. Lorenz system

The Lorenz system of equations is a simplified model for atmospheric convection [2] [24]. Close initial conditions lead to very different trajectories, making the Lorenz system a choatic dynamical system. The system is defined by:

$$\begin{cases} \dot{x} &= \sigma(y-x) \\ \dot{y} &= x(\rho-z)-y \\ \dot{z} &= xy-\beta z \end{cases}$$

Parameters σ, ρ and β are respectively set to 10, 28 and 8/3. The trajectory of a particle lies in an attractor whose shape resembles a butterfly. In [10], Brunton proposed a method to write a chaotic system as a forced linear system. Following this method, forcing statistics appear non-gaussian, with long tails corresponding to rare intermitting forcing preceding switching events (see Figures 7a and 7c). The system is simulated using a Runge Kutta 4 method, a random initial condition and a time step of 0.005s, for a total of 15000 samples. The chosen state is $\mathbf{s} = (x, y, z, \dot{x}, \dot{y}, \dot{z})$ which is the position and velocity of the particle on the attractor. The state is normalized using statistics from training data. The time-series of x, plotted in Figure 7b, clearly shows the lobe switching process (positive values when the particle travels on the right wing and negative values when it travels on the left wing). The objective is to extract from the simulated data a dynamical model mapping m past states to n future states. To account for modeling error, predictions are enforced using sequences of measurements. Our choice of observed variables is $\mathbf{y} = (\ddot{x}, \ddot{y}, \ddot{z})$ or $\mathbf{y} = (\dot{x}, \dot{y}, \dot{z})$ or $y = \dot{x}$. Measurements are directly linked to the state and data-driven models should automatically detect these relations.

Before training models and specify the prediction window n, the impact of n on the global error is investigated. Considering that wrong predictions are more likely to appear when predicting lobe switchings, two sources of errors, quantified on training data, can be found:

- ▷ Source 1, denoted e_1 → the ratio between the mean position of switching in a switching sequence and the prediction horizon n. The smaller the ratio, the bigger the impact on the global error.
- ▷ Source 2, denoted e_2 → the ratio between the number of sequences with switchings and the number of training sequences. The bigger the ratio, the bigger the impact on the global error.

Figure 7d shows the impact of global error for $n \in [10, 90]$. As expected, increasing the forecast window leads to a bigger impact on the global score (e_2/e_1 increasing) because prediction errors accumulate on longer sequences. However, the impact is not strictly monotone, indicating a dependance on the initial position of the particle. For the considered starting point, a forecast horizon $n = 80$ has a bigger impact on NMSE than

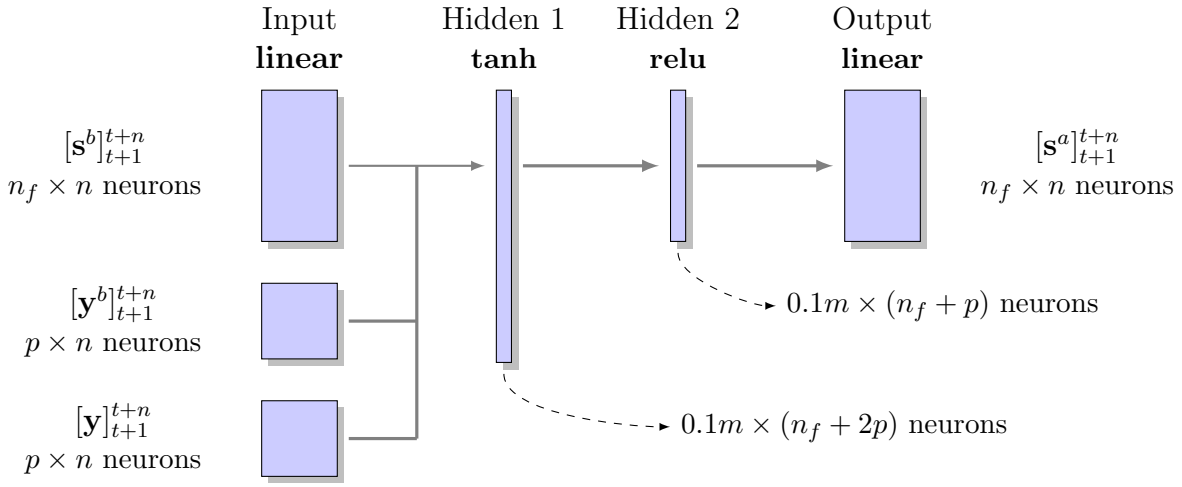


Figure 5: Data assimilation network. The nonlinear regression correct predicted sequences of states by assimilating sequences of real measurements. Hyperparameters must not be considered as a rule of thumb and are well tailored for the Lorenz 63 system.

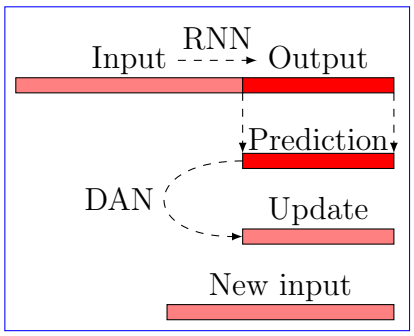


Figure 6: Procedure to test the data assimilation network. The dynamical model is used to predict n future states of the system using a history of m states. The predicted sequence is updated using the data assimilation network and next input is formed. All predicted sequences are then compared to all expected sequences using the normalized mean square error as a metric.

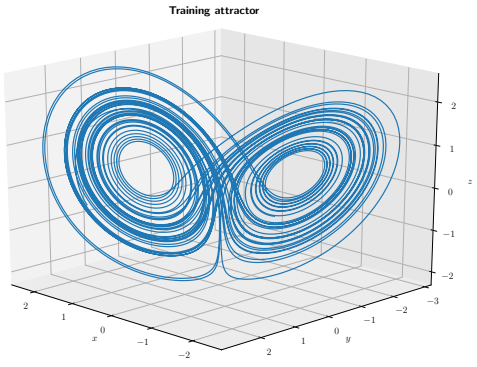
for $n = 90$, indicating that lobe switchings (so possibly wrong predictions) are more likely to appear at the beginning of a new sequence to predict for $n = 80$ and in the middle of the sequence for $n = 90$. In next sections, a history window $m = 100$ to capture one lobe switching or zero.

3.2. Testing the dynamical model

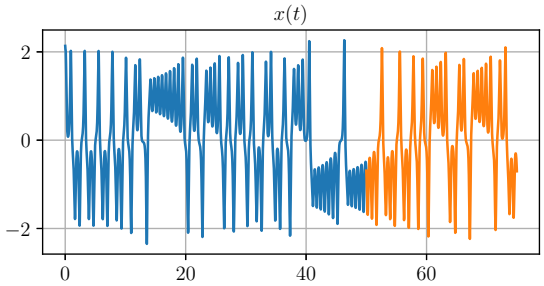
Nine dynamical models are established with $m = 100$ and $n \in [10, 90]$. Learning is stopped when the validation mean square error do not decrease for 3 epochs in a row. All learning curves show converged and close training and validation errors. The use of dropout layers or regularization techniques is then not necessary since no overfitting is noted. With models trained on the whole training data set, errors calculated on testing data are less than 1% for Test 1 but always exceeds 100% for Test 2 (see step 5 in Section 2.4 for the definition of tests). It means that the dynamical model alone has a great performance only for small term predictions. The Figure 8 shows predictions of x feature for both tests with a forecast horizon of 50-time steps. It is worth noting that despite the global score for Test 2, the dynamical model successfully recovers the region of phase space it was learnt on.

3.3. Testing the data assimilation network

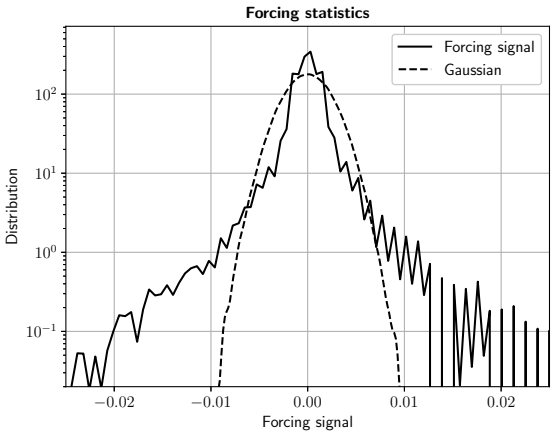
Concerning the shallow encoder (to map a state to its measurement), training is extremely fast and accurate with a training and testing determination coefficient close to 1. It means that the nonlinear regression performed by the neural network recovers nearly all the variance observed in training and testing data. This is not a surprise since the relationship between $\mathbf{s}^n$ and $\mathbf{y}^b$ is simple (derivation for the acceleration or selecting features for velocity). Concerning the data assimilation network, quantitative results of Test 3 are shown in Figure 9. Qualitative results for x and v_x



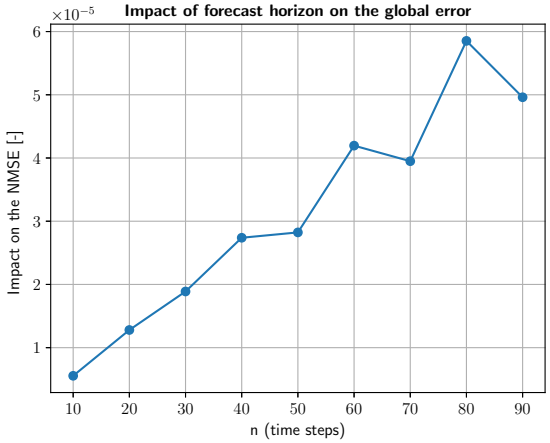
(a) Lorenz attractor



(b) Time-series of x feature.



(c) Distribution of forcing signal



(d) Impact of forecast horizon

Figure 7: Analysis of training data. The attractor (Figure a) can be seen as a forced linear system with a non-gaussian distribution for the forcing signal (Figure c, method from [10]). Lobe switchings are visible in time-series of x feature (Figure b) where the blue signal corresponds to training data and the orange signal corresponds to testing data. The forecast horizon n has an impact on the global score as shown in subfigure d.

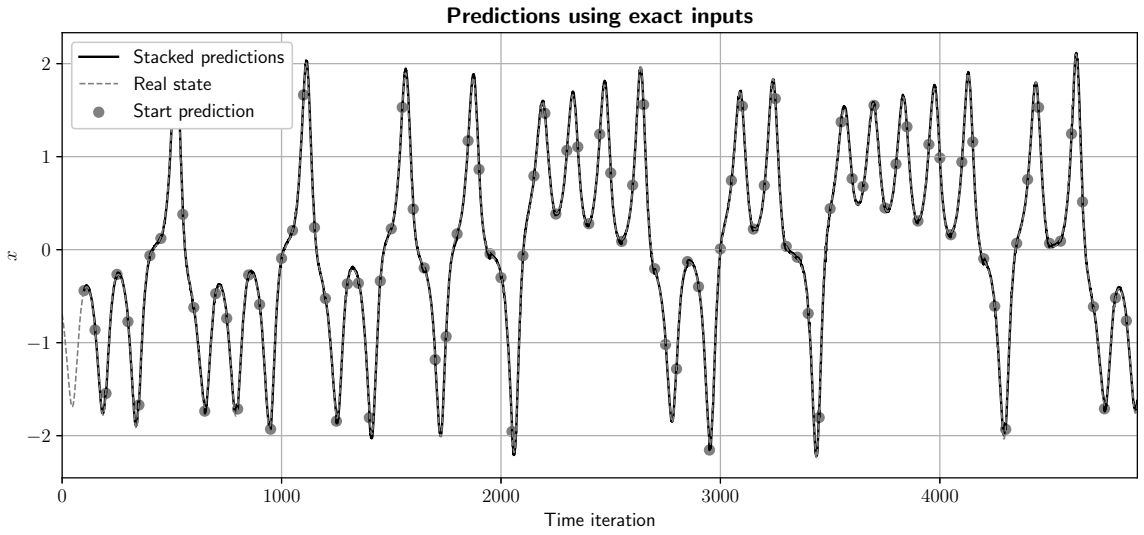
predictions using acceleration are shown in Figure 10. Several comments can be made:

1. Qualitatively, most of bad predictions are followed by good predictions: wrong predictions are detected and corrected by the DAN given online measurements.
2. Using the complete velocity leads to slightly better results than using the complete acceleration which seems reasonable because giving the velocity means giving three features out of six in the sequence to update.
3. Using only v_x leads to bad results for small sequences. To understand this behavior, the mean linear correlation coefficient between sequences of v_x and sequences of the state has been studied. It is defined by:

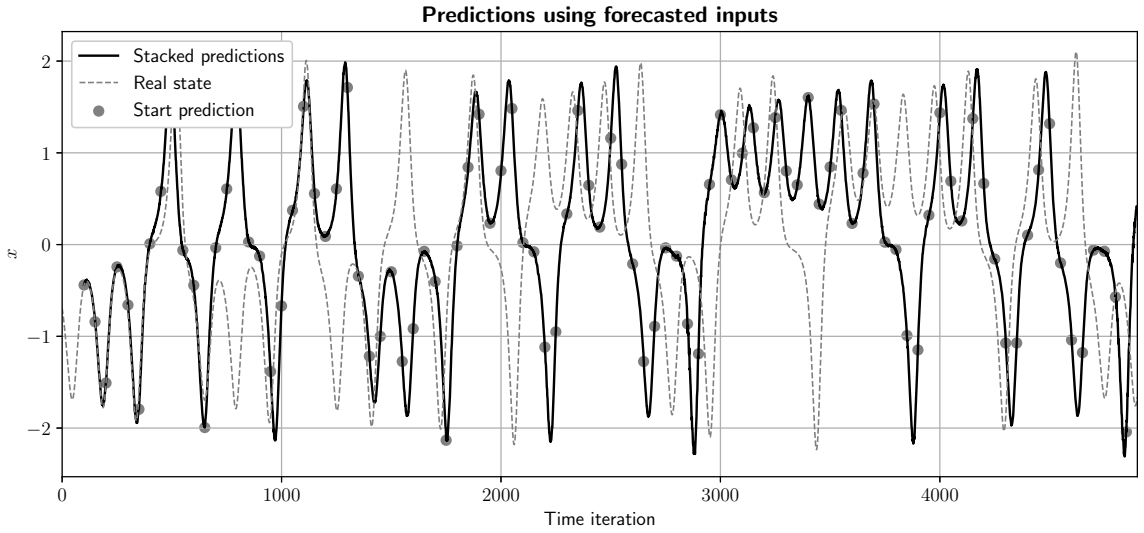
$$\bar{r}(v_x, s_k) = \frac{1}{n_{s_{train}}} \sum_{i=1}^{n_{s_{train}}} r([v_x]_i, [s_k]_i)$$

$$\text{with } r(X, Y) = \frac{cov(X, Y)}{\sigma_X \sigma_Y}$$

Where $n_{s_{train}}$ is the number of output training sequences of size n, $[v_x]_i$ is the i-th training sequence of v_x (size n) and $[s_k]_i$ is the i-th training sequence of the k-th feature of the state. Results are shown in Figure 11. It appears that small sequences of v_x



(a) Predictions of x (test set) when performing Test 1.



(b) Predictions of x (test set) when performing Test 2.

Figure 8: Test of the dynamical model for $n = 50$. Dots indicate the start of a new prediction, using m past exact states (Test 1) or m past predicted states (Test 2) as input.

are linearly correlated to all features in the state (linear correlation coefficient close to 1), which is no longer the case for medium and large sequences where nonlinearities arise (linear correlation coefficient between 0.6 and 0.7). Therefore, the data assimilation network has a too complex architecture for updating small sequences: a lot of unecessary parameters must be calculated during learning because the state could entirely be recovered by a linear regression on online v_x. This is a form overfitting and hyperparameters should be optimized but this is not the scope here.

4. Worst results are obtained for $n = 80$ which is linked to the dependance on the initial state to generate training data (Figure 7d).

We now suppose that the initial sequence is

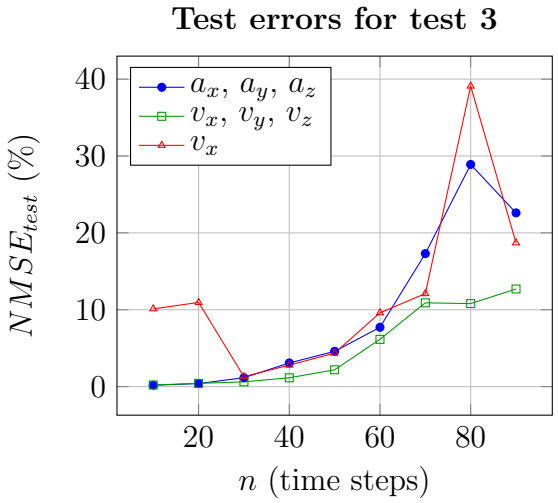


Figure 9: Testing errors computed from Test 3 using the complete acceleration, the complete velocity or v_x alone to update predictions.

noisy (Gaussian noise with standard deviation $\sigma_0 = 0.3$) just like online measurements (Gaussian noise with standard deviation $\sigma_y = 0.2$). The objective is to compare the proposed strategy with a simple Kalman filter update. Tests are restricted:

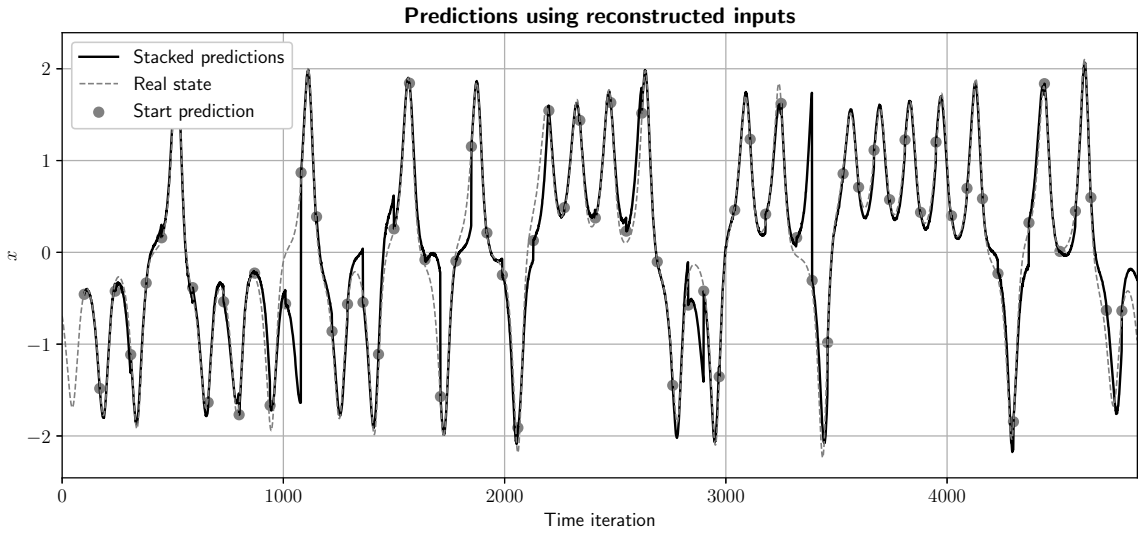
1. The simplest Kalman filtering technique requires the mapping between the state and its measurement to be linear (i.e. the H operator is simply a matrix). Tests will then concern v_x or the complete velocity as online measurements.
2. A Kalman filter requires the covariance of the prediction error to be advanced in time. In this paper, the covariance of the prediction error is supposed to be known at each new prediction and the predicted sequences are updated as a whole state.

Quantitative results are shown in Figure 12. We can observe that the DAN performs better for medium and large sequences but has poor performance on small sequences compared to the Kalman filter. This result was expected: the limited size of the noisy input sequence makes it harder to detect and learn regularity, resulting in sub-optimal performance. This effect is not noticeable for large sequences because a pattern can still be detected in the noisy sequence. It is also worth noting that the noise applied to a small initial sequence has no influence on performance since no weights are attributed to predicted sequences in the DAN architecture. When considering the complete velocity, results obtained from the DAN are close to those obtained by Kalman filter (which requires the error to be known while the DAN does not). The influence of noise on small sequences is not as important as when using v_x alone since the correction does not rely on a single neuron.

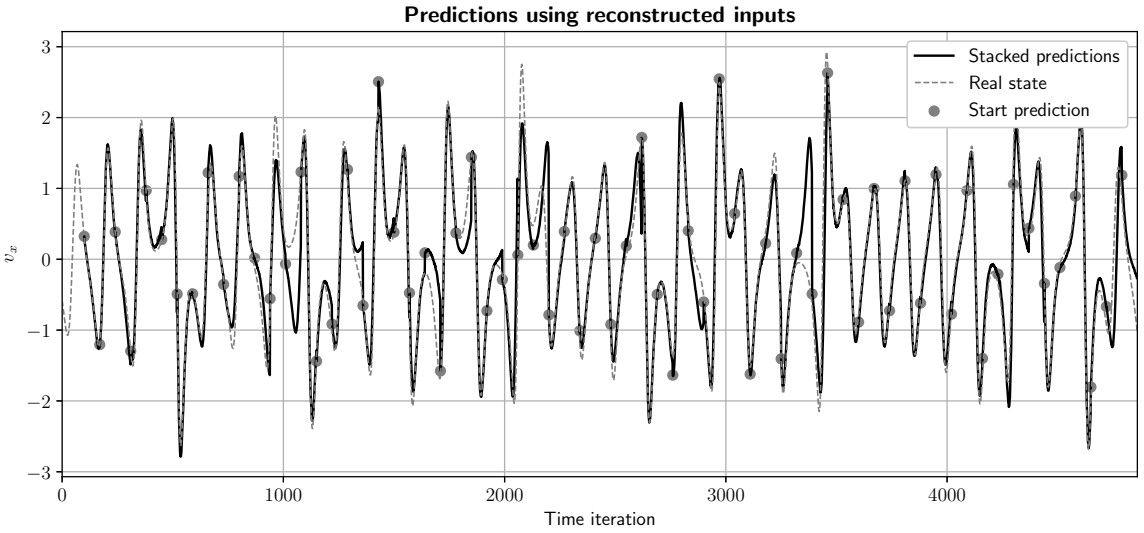
4. Discussions

The data-assimilation framework proposed in this paper is based on regressive methods. Its success is then tailored to the good choice/design of the method and the relevance of training data. If the trajectory to predict lies in the learned phase space's region, one can expected the DAN to have great performance. Otherwise, Kalman filtering techniques which do not depend on the problem should be prefered. The reader is referred to Mons et al. [14] for an overview about existing techniques.

Concerning the combination of neural networks with kalman filtering techniques, some works have already been done in the litterature. Most innovations concern the estimation of uncertainties. For instance, Coskun et al. [25] use LSTM cells to predict the internals of the Kalman filter. In doing so, the authors implicitely learn a dynamical model and covariance errors to use for a kalman update. In Loh et al. [23], authors update LSTM predictions of flow rates in gaz wells using an ensemble kalman filter, thus estimating errors via the covariance of an ensemble of predictions. In Becker et al. [26], a new architecture called reccurent kalman network is developped: using an encodeur - decodeur network, authors are able to estimate uncertainties in the features. Finally, Vashista [27] directly train a RNN - LSTM network to simulate ensemble kalman filter data assimilation using the differentiable architecture search framework. In this paper, we proposed a framework to directly work on sequences of data, separating the dynamical model from the data assimila-



(a) Predictions of x (test set) when performing Test 3.



(b) Predictions of v_x (test set) when performing Test 3.

Figure 10: Test of the data assimilation network for $n = 70$. Dots indicate the start of a new prediction, using m past reconstructed states as input (test 3).

tion process. Future investigations could include the explicit estimation of uncertainties using for instance bootstraping methods or gaussian processes [28]

Before applying the proposed strategy to a higher dimensional system, several challenges need to be adressed. First, the relevant phase region where to learn regressive models may be hard to detect as more features are involved. Second, optimal hyperparameters may not be easily guessable, thus requiring the need to grid search or genetic algorithm. Third, the dimensionality of the system could be reduced (by projecting it on a well defined basis) but some information would be sacrifcied, thus raising the problem of what relevant features should be kept. Vlachas et al. [29] adresses some of the problems by establish-

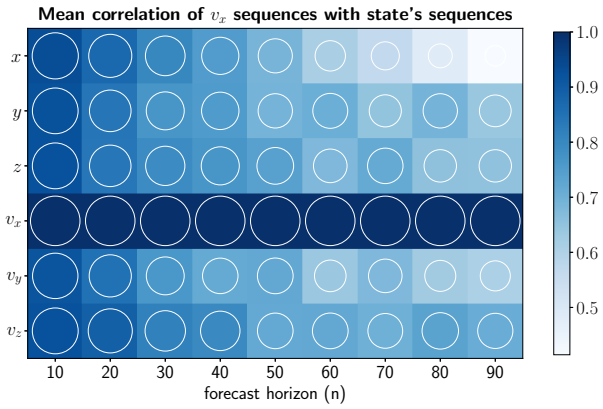


Figure 11: Mean correlation coefficient between output training sequences of v_x measurements and each feature in output training sequences of the state. Small sequences of v_x measurements are linearly correlated to all features in small sequences of the state. Nonlinearities arise for higher forecast horizons.

ing RNN - LSTM to forecast the Lorenz 96 system but no data assimilation coupled to their dynamical model has been yet intended.

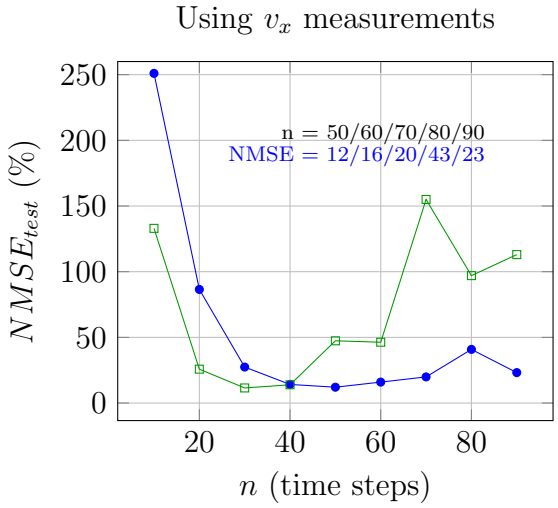
Finally, we are enthousiastic to use this framework for simple fluid mechanics problems: after reducing the state to estimate (using POD technique, see [30]), we aim to use data assimilation to continuously predict a simple flow. Neural networks involved in the dynamical or the data assimilation process will be improved by incorporating physics. For instance, a physics neural network is proposed by Ling [31] to establish a deep learning Reynolds Average Navier Stokes model embedding the invariance of the anisotropy stress tensor.

5. Conclusion

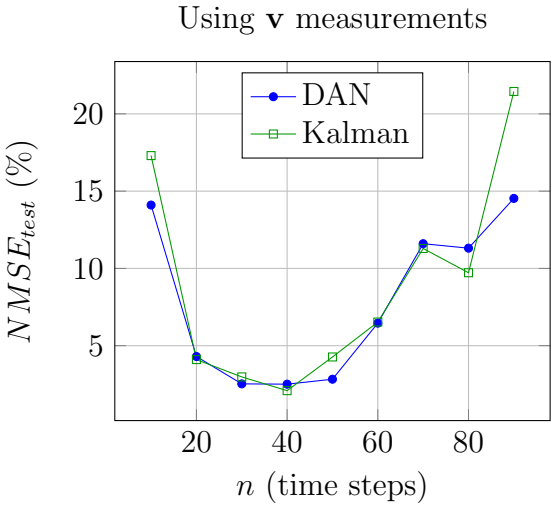
In this paper, we investigated the use of neural networks to predict multiple steps ahead the state of the Lorenz system. The first stage consisted in establishing a dynamical model mapping previous states to future states. A recurrent neural network handling long and short-term dependencies was used for this purpose. Supposing the input sequence was exact, the output proved to be accurate with less than 1% error. However, when running the dynamical model with predicted states as new inputs, errors accumulated at each new prediction, leading to a bad prediction the time-series: the system being chaotic with extreme events, a small error in the initial condition leads to a radically different output. To overcome this accumulation of errors and make a continuous forecast of the Lorenz system, a data assimilation strategy based on sequential techniques was developed. This consisted in a nonlinear network mapping between a predicted sequence of states and a corresponding sequence of online measurements. This strategy proved to be effective when starting with the exact initial sequence and feeding the system with exact online measurements, notably when using the complete acceleration or the complete velocity. A deeper analysis of the DAN structure showed that this strategy was less relevant when using a single measurement or when working with small forecast windows. Besides, the DAN proved to be sensitive (at least for small forecast windows) to noise in measurements but not to noise in the initial condition. The DAN remains a good alternative to a simple Kalman filter where the estimation of errors may be a difficult task, especially when updating sequences. It nonetheless must be noted that the success of the DAN is mainly due to the quality of training data and extra care must be taken when learning regression parameters. All in all, the global strategy developed here seems promising to continuously forecast other chaotic systems evolving on an attractor. Future works could include the tuning of hyperparameters (to have an optimal design for each neural networks) and the application to a high dimensional attractor where, similarly to Lorenz system, extreme events could be encountered.

6. Acknowledgments

The authours wish to thank ONERA and Hauts-De-France region for their funding.



(a) Using noisy v_x as online measurement



(b) Using noisy velocity as online measurement

Figure 12: Results from Test 3 using a Kalman filter update or the data assimilation network. Noise was incorporoted in both the initial sequence (starting point of the continuous forecast) and online measurements.

7. References

References

[1] A. Faqih, B. Kamanditya, B. Kusumoputro, Multi-step ahead prediction of lorenz's chaotic system using som elm-rbfnn, in: 2018 International Conference on Computer, Information and Telecommunication Systems (CITS), IEEE, 2018, pp. 1–5.

[2] E. N. Lorenz, Deterministic nonperiodic flow, Journal of the atmospheric sciences 20 (2) (1963) 130–141.

[3] J. Overland, J. Adams, H. Mofjeld, Chaos in the north pacific: spatial modes and temporal irregularity, Progress in Oceanography 47 (2-4) (2000) 337–354.

[4] K. T. Carlberg, A. Jameson, M. J. Kochenderfer, J. Morton, L. Peng, F. D. Witherden, Recovering missing cfd data for high-order discretizations using deep neural networks and dynamics learning, Journal of Computational Physics (2019).

[5] J. N. Kutz, Data-driven modeling & scientific computation: methods for complex systems & big data, Oxford University Press, 2013.

[6] S. Brunton, B. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, arXiv preprint arXiv:1905.11075 (2019).

[7] N. B. Erichson, L. Mathelin, Z. Yao, S. L. Brunton, M. W. Mahoney, J. N. Kutz, Shallow learning for fluid flow reconstruction with limited sensors and limited data, arXiv preprint arXiv:1902.07358 (2019).

[8] C.-K. Ing, Multistep prediction in autoregressive processes, Econometric theory 19 (2) (2003) 254–279.

[9] P. J. Schmid, Dynamic mode decomposition of numerical and experimental data, Journal of fluid mechanics 656 (2010) 5–28.

[10] S. L. Brunton, B. W. Brunton, J. L. Proctor, E. Kaiser, J. N. Kutz, Chaos as an intermittently forced linear system, Nature communications 8 (1) (2017) 19.

[11] E. Kaiser, B. R. Noack, L. Cordier, A. Spohn, M. Segond, M. Abel, G. Daviller, J. Östh, S. Krajnović, R. K. Niven, Cluster-based reduced-order modelling of a mixing layer, Journal of Fluid Mechanics 754 (2014) 365–414.

[12] R. Yu, S. Zheng, Y. Liu, Learning chaotic dynamics using tensor recurrent neural networks, in: Proceedings of the ICML, Vol. 17.

[13] J. Wang, Y. Li, Multi-step ahead wind speed prediction based on optimal feature extraction, long short term memory neural network and error correction strategy, Applied energy 230 (2018) 429–443.

[14] V. Mons, J.-C. Chassaing, T. Gomez, P. Sagaut, Reconstruction of unsteady viscous flows using data assimilation schemes, Journal of Computational Physics 316 (2016) 255–280.

[15] A. Gronskis, D. Heitz, E. Mémin, Inflow and initial conditions for direct numerical simulation based on adjoint data assimilation, Journal of Computational Physics 242 (2013) 480–497.

[16] J.-S. Zhang, X.-C. Xiao, Predicting chaotic time series using recurrent neural network, Chinese Physics Letters 17 (2) (2000) 88.

[17] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators,

Neural networks 2 (5) (1989) 359–366.
[18] R. Pascanu, T. Mikolov, Y. Bengio, Understanding the exploding gradient problem, CoRR, abs/1211.5063 2 (2012).
[19] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.
[20] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, S. Valaee, Recent advances in recurrent neural networks, arXiv preprint arXiv:1801.01078 (2017).
[21] F. Chollet, et al., Keras, `https://github.com/fchollet/keras` (2015).
[22] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).
[23] K. Loh, P. S. Omrani, R. van der Linden, Deep learning and data assimilation for real-time production prediction in natural gas wells, arXiv preprint arXiv:1802.05141 (2018).
[24] Z.-M. Chen, W. Price, On the relation between rayleigh–bénard convection and lorenz system, Chaos, Solitons & Fractals 28 (2) (2006) 571–578.
[25] H. Coskun, F. Achilles, R. DiPietro, N. Navab, F. Tombari, Long short-term memory kalman filters: Recurrent neural estimators for pose regularization, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5524–5532.
[26] P. Becker, H. Pandya, G. Gebhardt, C. Zhao, J. Taylor, G. Neumann, Recurrent kalman networks: Factorized inference in high-dimensional deep feature spaces, arXiv preprint arXiv:1905.07357 (2019).
[27] H. V. Vashistha, RNN/LSTM Data Assimilation for the Lorenz Chaotic Models, University of Maryland, Baltimore County, 2018.
[28] X. Qiu, E. Meyerson, R. Miikkulainen, Quantifying point-prediction uncertainty in neural networks via residual estimation with an i/o kernel, arXiv preprint arXiv:1906.00588 (2019).
[29] P. R. Vlachas, W. Byeon, Z. Y. Wan, T. P. Sapsis, P. Koumoutsakos, Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 474 (2213) (2018) 20170844.
[30] A. T. Mohan, D. V. Gaitonde, A deep learning based approach to reduced order modeling for turbulent flow control using lstm neural networks, arXiv preprint arXiv:1804.09269 (2018).
[31] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, Journal of Fluid Mechanics 807 (2016) 155–166.

Appendix A. LSTM cell

Long-Short Term Memory cells are recurrent units deploying a cell state and gating mechanisms. Equations of forget (f_t), input (i_t), output (o_t) and activation (a_t) gates are as follows:

$$\begin{cases} f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\ a_t &= \tanh(W_a x_t + U_a h_{t-1} + b_a) \end{cases}$$

Where W are weights associated to the input x_t, U are weights associated to the hidden input h_{t-1} and b are biases. Outputs are the cell state c_t and the hidden state h_t, computed according to:

$$\begin{cases} c_t &= c_{t-1} \odot f_t + a_t \odot i_t \\ h_t &= o_t \odot \tanh(c_t) \end{cases}$$

Where $\odot$ is the pointwise product. Given a new information $[x_t, h_{t-1}]$, the long-term memory c_t forgets information (via $f_t \odot c_{t-1}$) and stores a part of new information (via $a_t \odot i_t$). The short-term memory h_t depends on the long-term memory (via $\tanh(c_t)$) and the activation of the cell (via o_t) given new information.