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 ParisTech, Centrale Lille, FRE 2017 - LMFL - Laboratoire de Mécanique des fluides de Lille - Kampé de Feriet, 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 multi-step 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, an artificial neural network assimilate online data to detect and reconstruct 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 proving 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 multi-step prediction of such a system is challenging because governing equations may be unknown or too costly to evaluate. For instance, 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]. Providing the interpolative model is not overfitted to training examples, the data-driven model can also be used to **predict** *i.e.* extrapolate the future state of the system. Main techniques in the literature include autoregressive techniques [8], dynamical mode decomposition (DMD) [9], Hankel alternative view of Koopman (HAVOK) [10] or unsupervised methods (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], predict multi-step ahead [13] or approximate Koopman eigenfunctions [14].

However, errors in modeling can lead to bad multi-step 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 [15]. 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

*Corresponding author: pierre.dubois@onera.fr

a better description of initial or boundary conditions by finding the best compromise between experimental measurements and CFD predictions [16]. 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 [17]. To correct modeling errors, an artificial neural network (ANN) whose design is based on Kalman filtering techniques is developed. Results are promising for predicting multi-step 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.

It is important to note that the strategy presented in this paper will be used and adapted in future works for use in CFD. The final application involve the continuous prediction of a flow field via a data-driven model, using only an initial sequence of the state (the flow field) and real-time punctual measurements (pressure at some points for instance).

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. The last section gives concluding remarks and presents future investigations.

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 the first m states of the system. This sequence is denoted $[\mathbf{s}]_0^{m-1}$ where $\mathbf{s}$ is the state of the system.
- ▷ Predict n future states using a RNN with long and short-term memory (LSTM). This gives a predicted sequence $[\mathbf{s}^b]_m^{m+n-1}$ where superscript b indicates a prediction.
- ▷ Measure the predicted sequence. This gives a sequence $[\mathbf{y}^b]_m^{m+n-1}$ where $\mathbf{y}$ is a measurement of the state. The mapping between the state space and the measurement space is performed by an ANN called the shallow encoder (SE).
- ▷ Assimilate the exact sequence of measurements $[\mathbf{y}]_m^{m+n-1}$ to update the predicted sequence of states. This work is performed by an ANN which gives an updated sequence $[\mathbf{s}^a]_m^{m+n-1}$ where superscript a stands for "analyzed". The network is called the data assimilation network (DAN).
- ▷ Use the updated sequence as a new input and repeat the procedure.

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 introducing nonlinearities. These functions are classically a sigmoid $\sigma(x) = \dfrac{1}{1+e^{-x}}$, a hyperbolic tangent $\tanh(x)$ or a rectified linear unit $\mathrm{relu}(x) = \max(0, x)$. When neurons are organized in fully connected layers, the resulting network is called an artificial neural network (ANN). The universal approximation theorem [18] states that any function can be approximated by a sufficiently large and deep

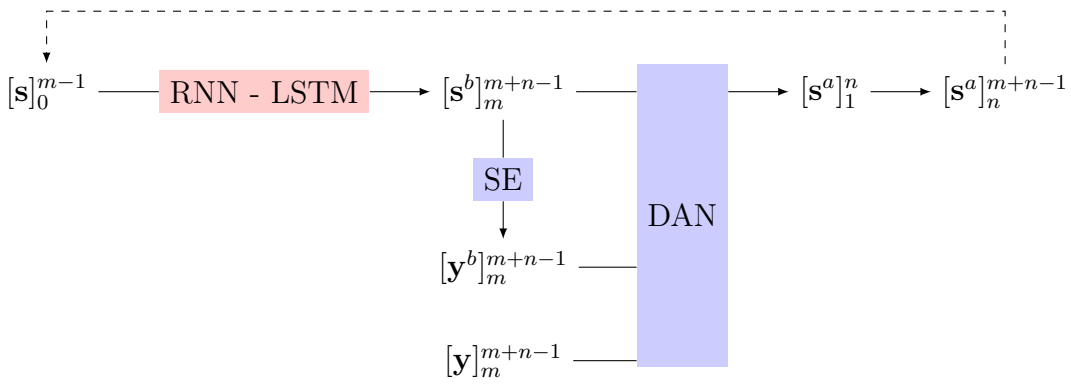


Figure 1: Summary of the data-driven method to make predictions of a chaotic system. A data-driven dynamical model (RNN-LSTM) predicts n future states of the system and the predict sequence is updated according to a real sequence of measurements.

network: numerous neurons in each layer increase nonlinearities (by positioning more s-shapes in the output) and several hidden layers create a better hierarchical representation of features. Just like a linear regression $y = ax + b$ aims 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 training set of data.

Although they are universal approximators, artificial neural networks face some limitations: they may suffer from vanishing or exploding gradient (derivatives of activation functions are non zero in a small range [19]), are prone to overfitting (weights and biases too representative of training data) and all inputs are taken at the same time (no time dependencies). Other architectures 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 sequences of inputs. In its simplest form, h is computed using a tanh function thus leading to the same limitations as artificial neural networks. The renewed interest in recurrent networks is largely attributed to the development of Long Short-Term Memory (LSTM) cells [20] which deploy cell states (long memory) and gating mechanisms to store or forget information about past inputs (see figure 2).

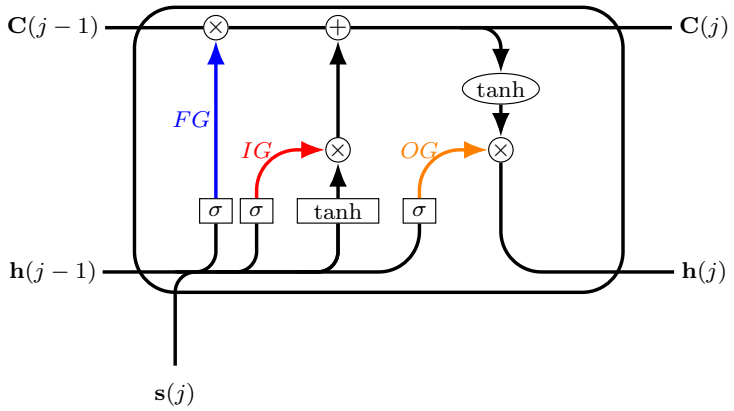
Several techniques exist to learn parameters in neural networks. The most common is the gradient descent, which iteratively update parameters according to the gradient of the cost function with respect to weights and biases. The computation of gradients is made by backwarding errors in the network, using backpropagation for ANN or backpropagation through time for RNN [6]. Equations of ANN, RNN and backpropagation can be found in [21] for the curious reader. In this paper, all neural networks are implemented using Keras library [22].

2.3. 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 3 and is composed of two networks. First, a recurrent neural network with $2m$ LSTM cells in the recurrent unit (making the cell state a $2m$ dimensional vector) treats each time step in the input sequence $[\mathbf{s}]_{t-m+1}^{t}$. The number of cells have been chosen to echo results of [1] where best scores were obtained by considering twice as many neurons than the history window. This results in a final output $o(t) = h(t)$ summarizing all relevant informations from the input sequence. Second, an artificial neural network with two hidden layers predict n future states $[\mathbf{s}^b]_{t+1}^{t+n}$ using the final output of the RNN. The procedure for learning the model is as follows:

(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.

1. Simulate the system to get data $t \to \mathbf{s}(t)$.
2. Split data into training and testing sets. In this work, two-thirds of the data are used to form the training set.
3. Form supervised problems by writing data as $[\mathbf{s}]_{t-m+1}^{t} \to [\mathbf{s}]_{t+1}^{t+n}$. The number of training examples is increased by considering a sliding window of one-time step i.e. training set is composed of $[\mathbf{s}]_{0}^{m-1} \to [\mathbf{s}]_{m}^{m+n-1}$, $[\mathbf{s}]_{1}^{m} \to [\mathbf{s}]_{m+1}^{m+n}$, 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 [23]. They are numerous parameters to find, including all weights and biases for each LSTM cells in the RNN and all parameters in the ANN. During the optimization process, the mean square error is also computed on the testing set. Errors evaluated on training and testing sets should be close to avoid

overfitting and ensure that weights and biases learned during training are relevant for extrapolative tasks.

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 using all predicted and real states by:

$$NMSE = \overline{\left[\frac{||\mathbf{s} - \mathbf{s}^b||^2}{||\mathbf{s}||^2}\right]}$$

Where $\bar{.}$ is the mean over all states operator and $||.||$ is the l_2 operator.

2.4. Data assimilation

To make a continuous forecast of the state using a data-driven dynamical model, it is almost always required to sequentially update predictions and limit the accumulation of errors [24]. 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 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 a state. The objective here is to adapt the strategy to update 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. The 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. Note that this architecture largely depends on the problem but for the Lorenz system, it is reasonable to explain the complete acceleration or complete velocity or a component in the velocity ($p = 3$ or $p = 1$) by the complete state ($n_f = 6$) with a shallow network. The training and testing of this H approximation is performed using data $\mathbf{s}(t) \rightarrow \mathbf{y}(t)$. The determination coefficient R^2 is used as a metric to quantify how good the regression is.

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 an artificial neural network whose architecture is summarized in figure 5. The process can be summarized as:

1. Simulate the system to get $t \rightarrow \mathbf{s}(t), \mathbf{y}(t)$. Measurements are exact.
2. Split into training and testing sets.
3. Form supervised problems for training and testing sets. The architecture being composed of dense layers, a sliding window of n is enough for both training and testing sequences. Training and testing sets are then composed of $[\mathbf{s}]_{t+1}^{t+n}$ and $[\mathbf{y}]_{t+1}^{t+n}$.
4. Perform test 1 (subsection 2.3) using training sequences and apply shallow encoder to get associated measurements. This leads to a set composed of $[\mathbf{s}^b]_{t+1}^{t+n}$ and $[\mathbf{y}^b]_{t+1}^{t+n}$. This set represents all possible predictions when using exact training inputs.
5. This step aims at learning the statistics of prediction and measurement errors. To do so, each training sequence $[\mathbf{y}]_{t+1}^{t+n}$ is associated to 20% random sequences from step 4. The objective is to blend a real sequence of measurement with randomly selected 20% of all possible predictions to produce the real sequence $[\mathbf{s}]_{t+1}^{t+n}$. This gives the final training set. Ideally, 100% of all possible predictions should be use but this would drastically increase the computational time to prepare the training set.

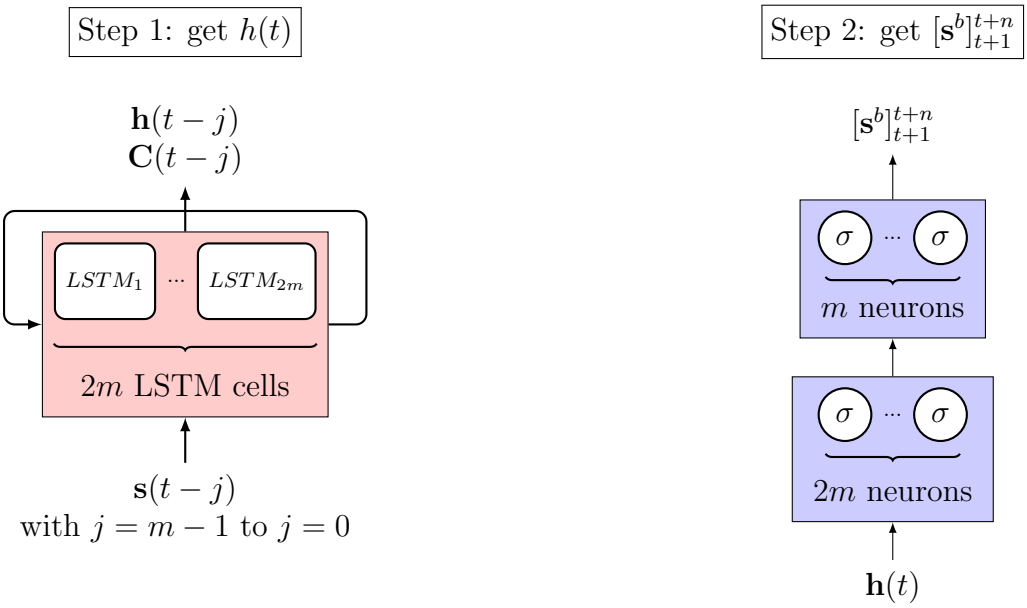


Figure 3: Architecture of the dynamical model, composed of a recurrent neural network and an artificial neural network. Idea behind the design: $2m$ cells are used to echo the results obtained in [1] where best prediction scores were obtained when considering two times the history window.

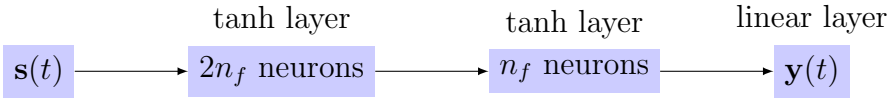


Figure 4: Shallow artificial neural network to map a state to its measurement i.e. approximation of H operator. Idea behind the design: flatten the state by a factor of 2 and make nonlinear combinations (hidden layer 1), get back to the dimension of the state (hidden layer 2), encode to the measure.

6. Perform gradient descent to optimize weights and biases.
7. Evaluate the performance of the final model using test data. Test 3 is summarized in figure 6. The metric is still the normalized mean square error.

The assimilation technique proposed here is a nonlinear regression learned on training data. It is a completely data-driven procedure whose success is tailored by the quality of the training set. This is different from Kalman filtering techniques where the Kalman gain only rely on statistics of errors and whose formula does not depend on training data.

3. Results

3.1. Lorenz system

The Lorenz system of equations is a simplified model for atmospheric convection [2] [25]. 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-

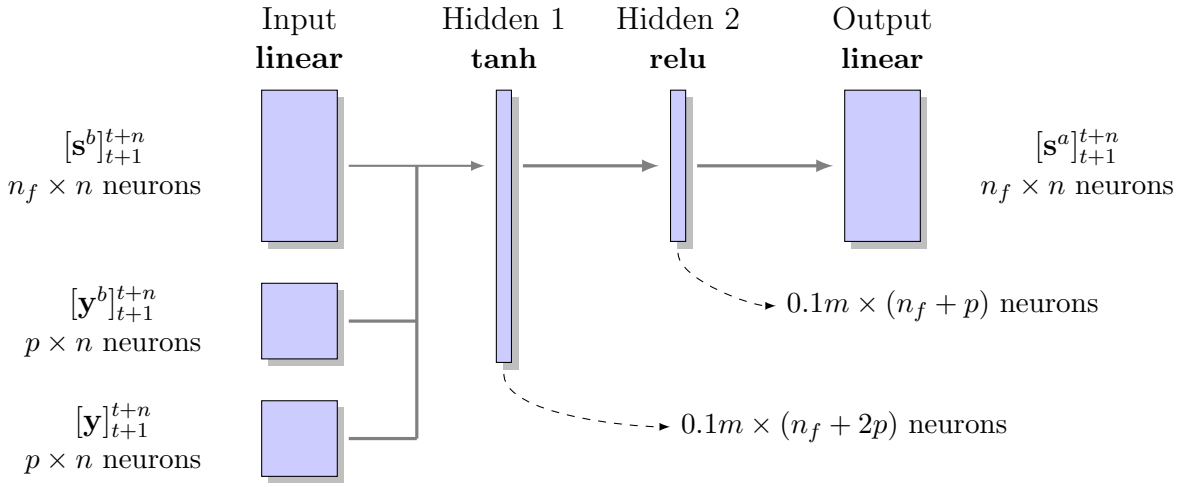


Figure 5: Data assimilation network. The nonlinear regression correct predicted sequences of states by assimilating sequences of real measurements. Idea behind the design: reduce all input information by a factor of $0.1m$ (hidden layer 1), find a kind of a residual by passing from $2p$ to p (hidden layer 2), flatten to get the updated sequence.

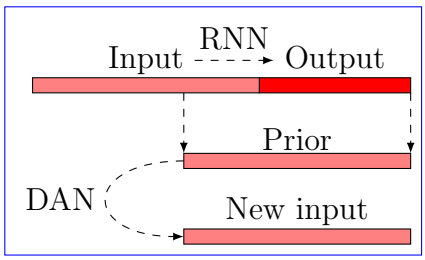
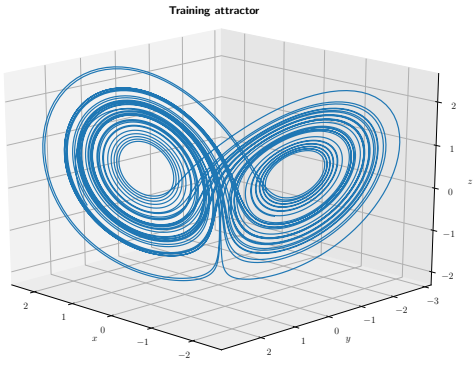


Figure 6: Procedure to test the data assimilation network. The dynamical model is used to predict n future states of the system using a history of m states. The predicted sequence is updated using the data assimilation network and the reconstructed input is used as a new input. All predicted sequences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

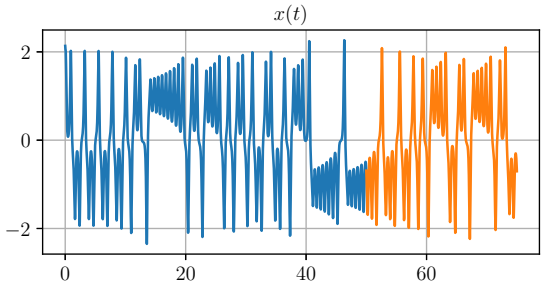
gaussian, with long tails corresponding to rare intermitting forcing preceding switching events (see figures 7a and 7c). The system is simulated using a runge kutta 4 method, a random initial condition and a time step of 0.005s, for a total of 15000 samples. The data is **normalized** i.e. with zero mean and unit standard deviation for convenience and to ease future learning of neural networks. 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 time-series of x feature, plot in figure 7b, clearly shows the lobe switching process (positive values when the particle travels on the right ear and negative values when it travels on the left ear). 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. Observed variables can be $\mathbf{y} = (\ddot{x}, \ddot{y}, \ddot{z})$ 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 m and n, a first investigation concerns the impact of discretizing the attractor on the global error. Considering that wrong prediction are more likely to appear when predicting lobe switchings, two sources of errors can be found:

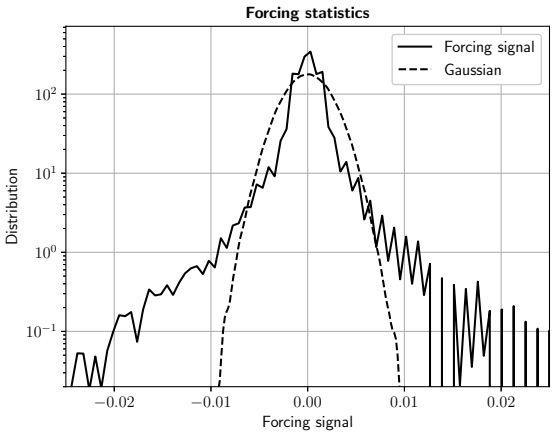
- ▷ Source 1 → the ratio between the mean position of switching in a switching sequence and the prediction horizon. The smaller the ratio, the bigger the impact on the global error.
- ▷ Source 2 → the ratio between the number of sequences with switchings and the number of training sequences. The bigger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



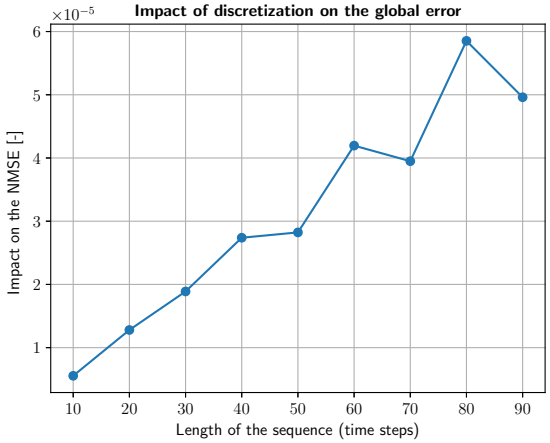
(a) Lorenz attractor



(b) Time-series of x feature.



(c) Distribution of forcing signal



(d) Impact of discretization

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.

Figure 7d shows the impact of discretizing the training attractor for $n \in [10, 90]$. As expected, increasing the forecast window leads to a bigger impact on the global score (s_2/s_1 increasing) because prediction errors accumulate on longer and less numerous sequences. However, the impact is not strictly monotonous, indicating a dependance on the initial position of the particle. For the considered starting point, a discretization $n = 80$ has a bigger impact on NMSE than 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 $m = 100$ is chosen to capture at least a lobe switching or not.

3.2. Testing the dynamical model

Nine dynamical models are established with $m = 100$ and $n \in [10, 90]$. In each case, learning is stopped when the mean square error no longer evolves for 3 epochs in a row. All learning curves show converged and close training and testing errors. The use of dropout layers or regularization techniques is not necessary since no overfitting is noted. The NMSE is less than 1% for Test 1 but always exceeds 100% for Test 2 (see step 5 in section 2.3 for the definition of tests). It means that the dynamical model has great performance when fed with an exact input sequence but fails to con-

tinuously predict the state of the system. 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 predicted dynamics resembles the expected one with a good prediction of the attractor.

3.3. Testing the data assimilation network

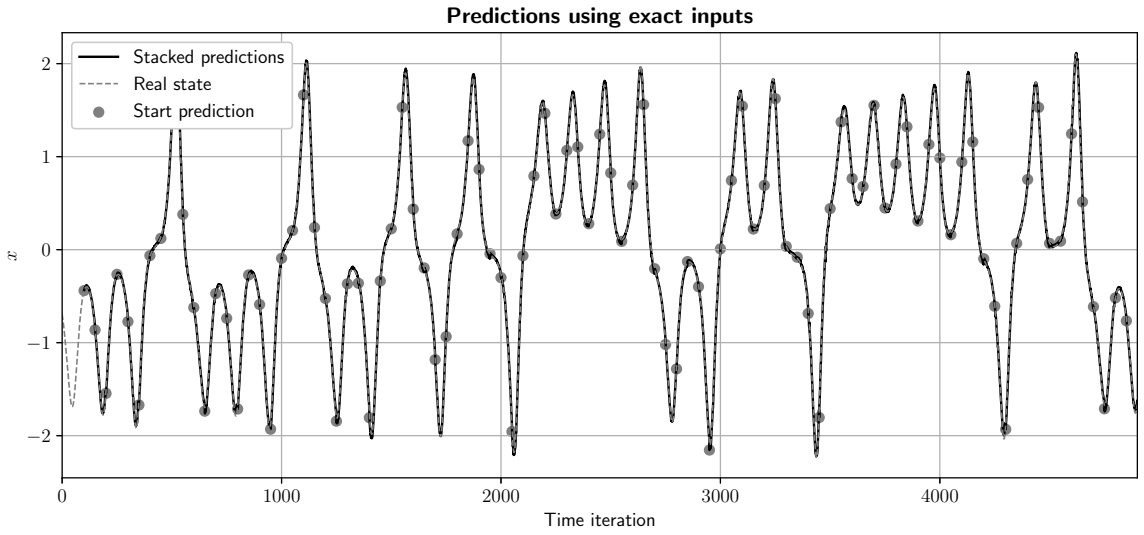
Concerning the shallow encoder (to map a state to its measurement), training is extremely fast and accurate with a determination coefficient close to 1. It means that the nonlinear regression performed by the neural network recovers nearly all the variance observed in training and testing data. This is not a surprise since the relationship 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 predictions using acceleration are showed in figure 10. Several comments can be made:

1. Qualitatively, most of bad predictions are followed by good predictions: wrong predictions are correctly reconstructed 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 reconstruct.
3. Using only v_x leads to bad reconstruction 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 appears that small sequences of v_x are linearly correlated to all features in the state (r close to 1), which is no longer the case for medium and large sequences where nonlinearities arise (r between 0.6 and 0.7). Therefore, the data assimilation network has a too complex architecture for updating small sequences: lots of neurons are troublemakers during learning because the state can entirely be recovered by a linear regression on online v_x.
4. Bad reconstruction results are obtained for $n = 80$ which is directly linked to the discretization process and the dependance of the initial state to generate training data.

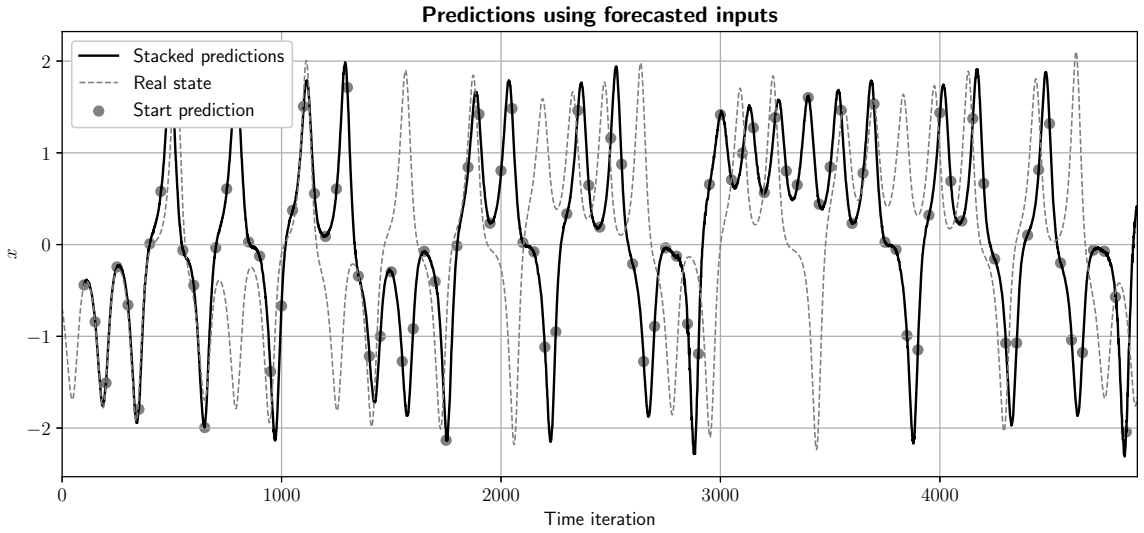
We now suppose that the initial sequence is noisy (gaussian noise with $\sigma_0 = 0.3$) just like online measurements (gaussian noise with $\sigma_y = 0.2$). The objective is to compare the proposed strategy with a simple Kalman filter update. Tests are restricted:

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 complete velocity as online measurements.
2. The seed for generating random numbers is the same for DAN and Kalman filter tests. This helps reducing testing time by not considering ensemble techniques.
3. A Kalman filter requires the covariance of the prediction error to be advanced in time. This normally uses the jacobian of the dynamical model but in this work, the dynamical model is a neural network mapping sequences to sequences. The nonlinearity makes it hard to compute the gradient of the output with respect to the input **and** the mapping between sequences makes it difficult to use standard Kalman filtering methods working with $t \rightarrow t + dt$ models. To address these problems, the covariance of the prediction error is supposed to be known at each new prediction and the predicted sequences are updated as a whole instead of treating each time step in the sequence individually. Note that some strategies to address the mapping between sequences exist ([26], [24]) but they are out of the scope of this paper.

Quantitative results are shown in figure 11. We can observe that the DAN performs better for medium and large sequences but has poor performance on small sequences compared to the Kalman



(a) Predictions of x feature when performing test 1.



(b) Predictions of x feature 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.

filter. This result was expected: for small sequences, the DAN gives high weights to online measurements so that if noise is applied to small sequences, the input to the DAN does not have the regularity detected and learned by the neural network, resulting in bad behavior. 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 in the initial sequence has no influence when using small sequences since no weights are attributed to predicted sequences in the DAN architecture. When considering the complete velocity, results obtained from the DAN are slightly better than 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 cor-

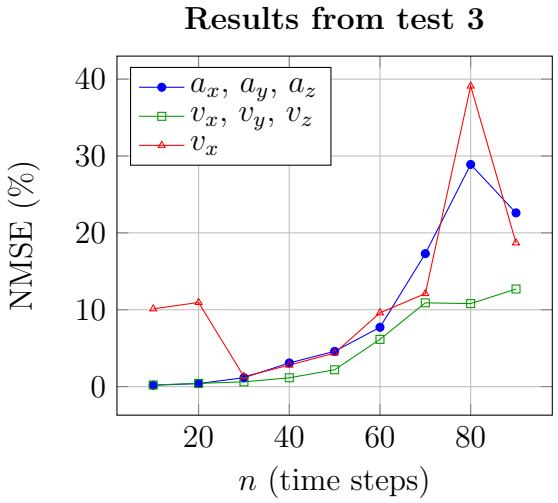


Figure 9: Results from test 3 using the complete acceleration, the complete velocity or v_x alone to update predictions.

rection does not rely on a single neuron.

4. Conclusion

In this paper, we investigated the use of neural networks to predict multi-steps 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% of errors. However, when running the dynamical model with predicted states as new inputs, errors accumulated at each new prediction, leading to a good prediction of the dynamics but not of 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. It consisted in establishing a network mapping in a nonlinear manner a predicted sequence of states with a real sequence of online measurements. This strategy proved to be efficient 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 sensible (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 and future works could include the tuning of hyperparameters (to have an optimal design for each neural networks; a genetic algorithm would be suitable for this task) and the application on a high dimensional attractor where, similarly to Lorenz system, extreme events could be encountered.

5. Acknowledgments

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

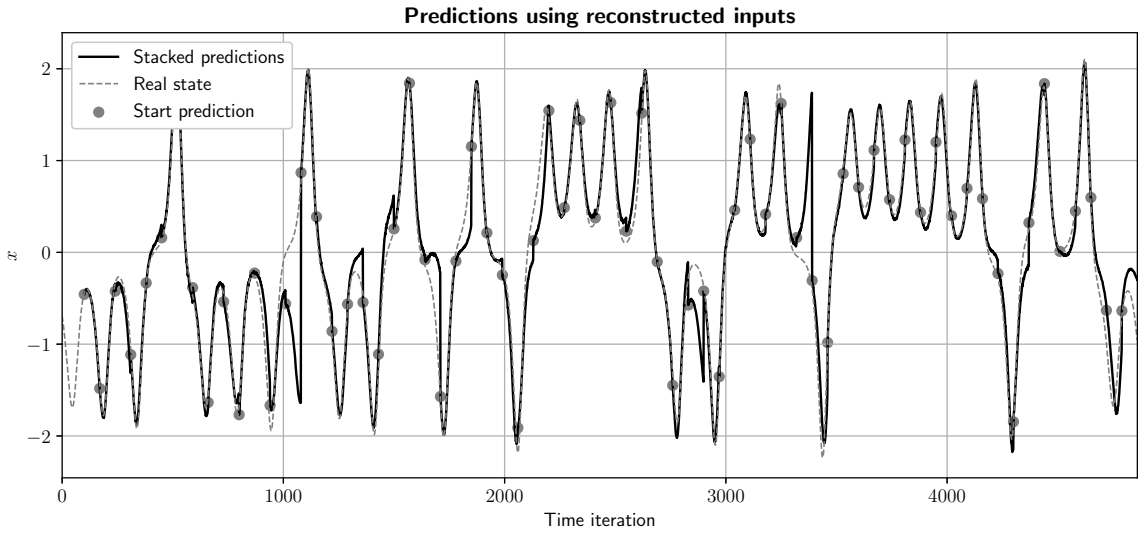
6. References

[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.

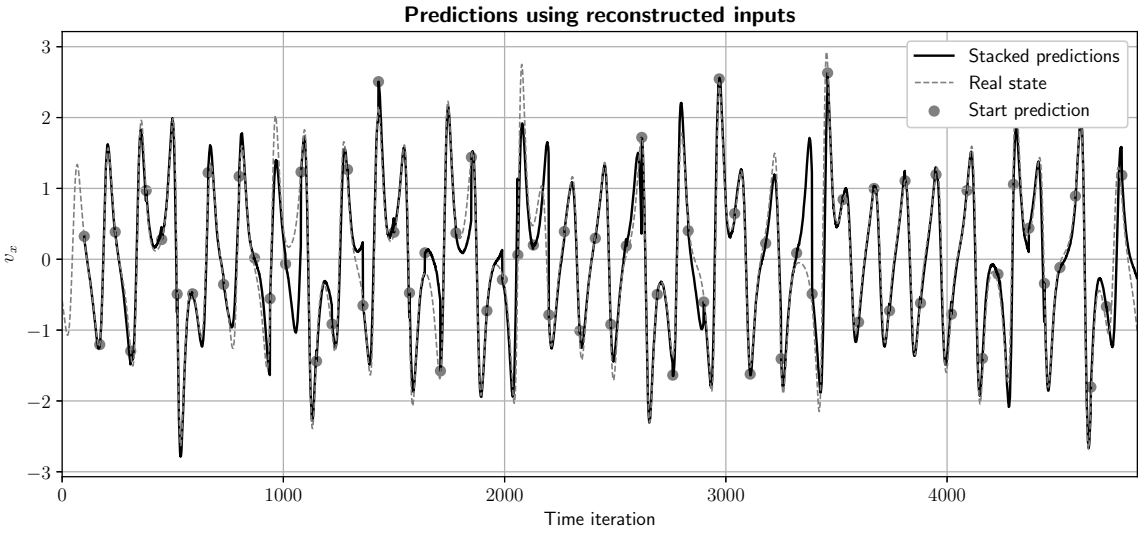
[2] E. N. Lorenz, Deterministic nonperiodic flow, Journal of the atmospheric sciences 20 (2) (1963) 130–141.

[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.



(a) Predictions of x feature when performing test 3.



(b) Predictions of v_x feature 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).

[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.

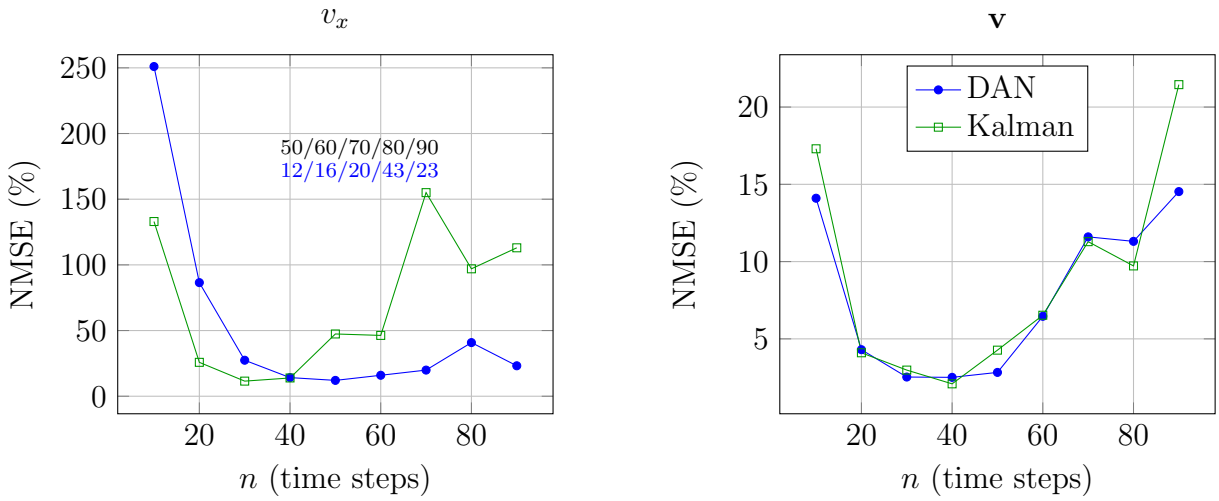
[7] N. B. Erichson, L. Mathelin, Z. Yao, S. L. Brunton, M. W. Mahoney, J. N. Kutz, Shallow learning for fluid flow reconstruction with limited sensors and limited data, arXiv preprint arXiv:1902.07358.

[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



(a) Using noisy v_x as online measurement

(b) Using noisy velocity as online measurement

Figure 11: Results from test 3 using a Kalman filter update or the data assimilation network. The initial condition is noisy as well as online measurements.

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] B. Lusch, J. N. Kutz, S. L. Brunton, Deep learning for universal linear embeddings of nonlinear dynamics, Nature communications 9 (1) (2018) 4950.

[15] 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.

[16] 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.

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

[18] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural networks 2 (5) (1989) 359–366.

[19] R. Pascanu, T. Mikolov, Y. Bengio, Understanding the exploding gradient problem, CoRR, abs/1211.5063 2.

[20] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.

[21] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, S. Valaee, Recent advances in recurrent neural networks, arXiv preprint arXiv:1801.01078.

[22] F. Chollet, et al., Keras, `https://github.com/fchollet/keras` (2015).

[23] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

[24] 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.

[25] Z.-M. Chen, W. Price, On the relation between rayleigh–bénard convection and lorenz system, Chaos, Solitons & Fractals 28 (2) (2006) 571–578.

[26] 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.

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.