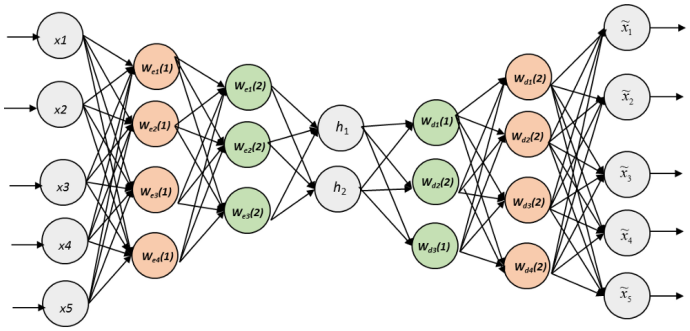
Order Reduction on Dynamic Systems using Machine Learning

Academic Coordinator Andrés Gómez Tato
University Galicia Supercomputing Center (CESGA)

Business Coordinator Ángel Rivero Jiménez
Company Repsol Technology Center

Specialist Pablo Solano López
University Technical University of Madrid (UPM)

Team David García-Selfa (University of Santiago de Compostela), Manuel Díaz Méndez (ITMATI), Francisco Pena and Jaime López García (University of Santiago de Compostela and ITMATI), Marco Martinolli (Politecnico of Milano), Bernadett Stadler (Industrial Mathematics Institute, Johannes Kepler University Linz), Naomi Auer (WIAS Berlin), Rocío Calero Fernández-Cortes (UPM), María del Mar González Nogueras (Autonomous University of Madrid) and Umberto Emil Morelli and Francisco Mora Posada (ITMATI)



Order Reduction on Dynamic Systems using Machine Learning

Ángel Rivero Jiménez[1], Pablo Solano López[2], Andrés Gómez Tato[3], David García-Selfa[4], Manuel Díaz Méndez[5], Francisco Pena[6], Marco Martinolli[7], Bernadett Stadler[8], Naomi Auer[9], and Umberto Emil Morelli[9]

Abstract

This paper summarizes the results of the P3 group of ESGI 139th event. Due to recent results in the area of Deep Learning for solving Partial Differential Equations, REPSOL proposed to study the possibility of using these techniques in the case of chaotic PDEs as Lorenz or Kuramoto-Sivashinsky equations. Starting from [1] and [2], the workgroup was focused on study these methods and the difficulties to applied to some cases. The main conclusions are that Reservoir networks, as stated by [1], allow to approximate the results of these equations once a initial set of results are produced using other classical methods, but they could be difficult to parameterize and have some issues with the scalability. Also, from [2], the usage of autoencoders could reduce the order of the equations. As consequence, it seems that combining both methods (autoencoder plus Reservoir Neural Networks) could provide a plausible method to make an order reduction on chaotic PDEs which deserves more research.

[1] REPSOL Center of Technology
[2] Área de Ingeniería Aeroespacial, Universidad Rey Juan Carlos
[3] Galicia Supercomputing Centre (CESGA); agomez@cesga.es
[4] Group of Nonlinear Physics, Faculty of Physics, University of Santiago de Compostela (USC)
[5] Technological Institute for Industrial Mathematics (ITMATI) and Politecnico di Milano
[6] Applied Mathematics Department (USC)
[7] Johannes Kepler University Linz
[8] WIAS Berlin
[9] Technological Institute for Industrial Mathematics (ITMATI)

1. Introduction

Solving Partial Differential Equations (PDE) is a complex challenge that frequently has to be done numerically because no other analytical option is available. Depending on which problem to solve, one has to choose the appropriate method and its parameters to achieve a high-quality numerical approximation. This is often translated into expensive computations in terms of memory allocation (fine spatial grids) and computation time (small timesteps), further- more, some applications such as weather prediction, require further tuning in order to force the prediction to follow the real data.

To minimize this kind of risks and to speed up the solutions, there are some techniques that transform the inefficient numerical problem into a smaller one which can be solved faster and produce solutions of similar quality, techniques coined as Model Order Reduction (MOR). One example is the so-called Proper Orthogonal Decomposition (POD)[3].

However, during the last years, Deep Learning [4] has emerged as a powerful tool to solve a large variety of problems from classification of pictures to self-driven cars. The usual workflow is to use a set of real data to create a representation of them that can predict some characteristic of a new future set with high accuracy. So, one possible methodology to control the aforementioned physical problems is to use real data instead of differential equations, i.e., a data-driven solution. The advantage to use Deep Learning, specially Neural Networks (NN), is that they can cope with non-linear problems and, once they are trained, getting a new prediction from a set of data is extremely fast. But it is also possible to follow an alternative method, conceptually similar to MOR techniques: generate an initially set of synthetic data using numerical solutions of PDE and, using Deep Learning, devise a reduced model (from the point of view of computation because is faster by itself or because it uses fewer variables).

Recent publications [1, 2] which addressed the problem of solving chaotic PDEs using Deep Learning techniques spiked the interest of REPSOL about their capabilities to solve real problems. In fact, the Problem 3 of 139th ESGI group tries to answer two initial questions related with the use of Deep Learning for solving PDEs:

- Do we need to keep all the degrees of freedom (dof)? In other words, for a given accuracy, what are the most relevant/optimum modes, that reproduce the dynamics?

- Are numerical methods, and in a further glance, the PDEs, the best way to model, solve and predict the dynamics?

This small paper presents the initial conclusions of the workgroup regarding the usage of these methods for solving PDEs faster. During this week the

authors focused the work in understanding the methods that raised the REPSOL interest and their limitations, centering the advances in two well-known chaotic PDEs with different levels of complexity.

The paper is organized in six sections. The first one is a brief overview of the NN used techniques. For a broader introduction to Deep Learning the reader should look other references as [4]. In this section only relevant information to understand the main results of the workgrop is summarized. Next section explains the importance of PDEs for the industrial sector. It is followed by the main achieved results related to the usage of Reservoir NN following [1]. Section five will analize briefly the usage of autoencoder to finish with the last one which includes the conclusions and recommendations.

2. Neural Networks for a System of Partial Differential Equations

Ever since the development of calculus, Partial Differential Equations (PDEs) have proven to be an incredibly useful tool for physicists and engineers to model the reality both as a variety of continuous and discrete systems. Their utility is that the solutions obtained not only match the physical evolution of studied system but also unravel hidden patterns that might not be measured or detected in a first approach. Unfortunately, in general these PDEs cannot be solved analytically and so they require for approximation techniques that transform the problem into numerical operations a computer can process to get an estimate of the PDE's solution. To obtain such a solution, the usual methods are based on spatio-temporal discretization, i.e., sampling the continous problem into a defined grid. Most common methods found in the literature are finite elements (FEM), finite differences (FDM) or finite volumes (FVM). On one hand, these techniques are very efficient for low-dimensional problems and simple and slightly complex geometries, having proved its performance in many applications in the past decades. On the other, the meshing becomes increasingly difficult for its numerical treatment with the complexity of the geometry. Not only that, the computation goes out of reasonable time scales the moment the problem requires for very high accuracy or spatial resolution. And in addition to all this, most of these algorithms only compute the solution at the grid points and the evaluation of the rest of the domain is then obtained by interpolation or by some other reconstruction technique [5].

As a result of the Cybenko theorem [6], feed forward NN can approximate any continuous function on compact subsets of $\mathbb{R}^n$. Consequently, NN methods can solve both partial and ordinary differential equations. The feed forward NN can then be used as a basic approximation element whose parameter are adjusted during the training to minimize an appropriate error function [7].

The NN provide a differentiable solution with good generalization properties

in close analytic form. This is one of the advantages NN methods have over the aforementioned numerical methods. Moreover, for NN the computational effort does not increase quickly with the number of sampling points. The generality of the method implies that it can be applied to systems independently of their geometrical complexity. Finally, the method can be parallelized and offers an opportunity to solve PDE problems in real time.

2.1. Neural Network Architecture

The number of Neural Networks architectures is constantly increasing and describing all of them is beyond the possibilities of this small summary. In this section only a few of them are described with particular attention to reservoir and autoencoder NN which have been used in the present investigation. Other extensive reviews on the subject are available in the literature [8, 9].

The simplest form of NN is the feed forward NN shown in Figure 1. The direction moves from the input nodes, through the optional hidden nodes and to the output nodes. No cycles or loops are implemented in this architecture. Mathematically, each node i of the hidden and output layer calculates

$$y_i = \mathbf{g}(\mathbf{x^T}, \mathbf{w_i} + b_i) \tag{1}$$

where $\mathbf{x}$ is a vector with the inputs comming from the previous layer, $\mathbf{w_i}$ is a vector of unknown parameters (named weights) and b_i is a unknown constant (named as bias), both of them to be fitted, and $\mathbf{g}$ is a bounded non-linear function as *tanh* or *sigmoid*, named activation. When a layer has several nodes,$\mathbf{w_i}$ is transformed in a matrix $\mathbf{W_i}$ where each column is formed by the weights of each node, $\mathbf{b_i}$ is converted in a vector $\mathbf{b_i}$ and the activation function is applied element-wise. This method is applied iteratively layer by layer, resulting on

$$\mathbf{y} = \mathbf{g_i}(\mathbf{g_{i-1}}(\mathbf{g_{i-2}}(...\mathbf{g_1}(x^T \cdot W_1 + \mathbf{b_1})...) \cdot W_{i-1} + \mathbf{b_{i-1}}) \cdot W_i + \mathbf{b_i}) \tag{2}$$

with i is equal to the number of hidden layers plus the output layer. To find the correct values for W_i and b_i, usually a supervised training is executed, where a set of labeled inputs $\mathbf{x_j}, \mathbf{y_j}$ is used to mimimize an objective function which relates the predictions of the NN ($NN(x_j)$) with the labels y_j, frequently applying a gradient descent technique. Because the number of cases is usually very large, the input set is divided randomly in subsets, each one being the input for one optimizacion step. For this reason, the process is known as stochastic gradient descent.

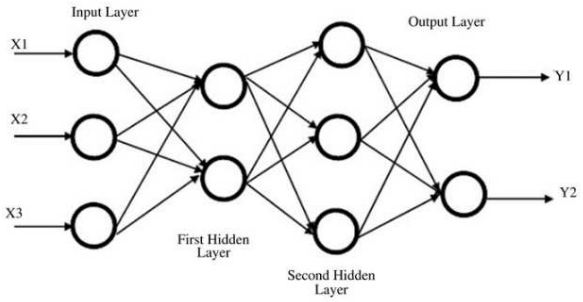


Figure 1: Schematic of a feed forward neural network [7].

Recurrent NN can have connections that go backwards from output nodes to input nodes as well as arbitrary connections between nodes as shown in Figure 2. This kind of networks can preserve sets of data, thus can be said to have a memory. This feature is important when the solution of the problem is dependent from previous inputs.

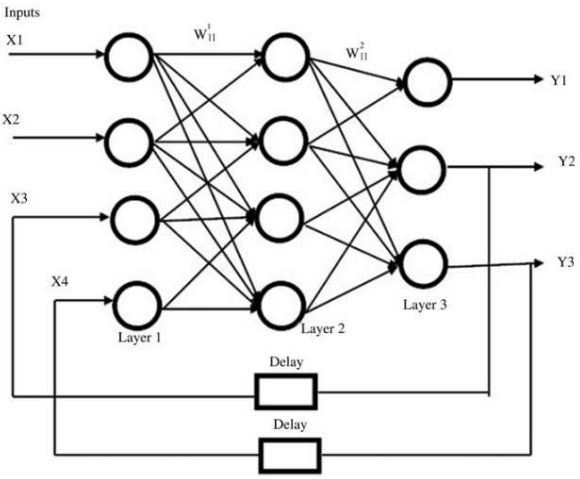


Figure 2: Schematic of a recurrent neural network [7].

One of the neural network used in the present investigation is the reservoir NN. In these networks, an input vector $\mathbf{u}(t)$ of dimension D_{in} is coupled to a high-dimensional dynamical system called "reservoir". As schematically shown in Figure 3, the coupling between the reservoir and the input vector is made by an input-to-reservoir coupler (I/R). From the reservoir, an output vector $\mathbf{v}(t)$ of dimensions D_{out} is coupled through a reservoir-to-output coupler (R/O). The R/O coupler depends on D_r adjustable parameters $\mathbf{p}$. The output $\mathbf{v}(t)$ it creates is assumed to depend linearly to upon the parameters $\mathbf{p}$.

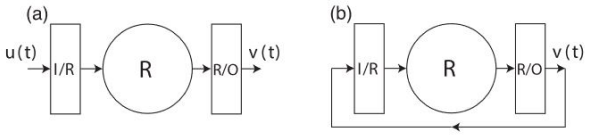


Figure 3: Schematic of a reservoir neural network [1]. (a) Training data gathering phase. (b) Predicting phase.

Being $\mathbf{r}(t)$ the D_r dimensional state vector of the reservoir, it is possible to represent in discrete time ($t = 0, \Delta t, 2\Delta t, \cdots$) the introduced functions as [1]

$$\begin{aligned} \mathbf{r}(t + \Delta t) &= \mathbf{G}\{\mathbf{r}(t), \mathbf{W}_{in}[\mathbf{u}(t)]\} \\ \mathbf{v}(t) &= \mathbf{W}_{out}[\mathbf{r}(t), \mathbf{p}] \end{aligned} \tag{3}$$

where $\mathbf{W}_{in}$ (respectively $\mathbf{W}_{out}$) is a mapping from the D_{in} (D_{out}) dimensional reservoir (output) state space. The system is trained to make $\mathbf{v}(t)$ closely approximate the desired output function $\mathbf{v}_d(t)$ appropriate to the input $\mathbf{u}(t)$.

In the training period $-T \leq t \leq 0$, the training data $\mathbf{u}(t)$ and the resulting $\mathbf{r}(t)$ are used. The output parameters $\mathbf{p}$ are then chosen to minimize the least square difference between $\mathbf{v}_d(t)$ and $\mathbf{v}(t)$ along the training period [1]. Due to the linearity of $\mathbf{v} = \mathbf{W}_{out}[\mathbf{r}, \mathbf{p}]$ with respect to $\mathbf{p}$, $\mathbf{p}$ and $\mathbf{W}_{out}$ are determined through a linear regression [10].

In the present investigation, a reservoir NN is used to predict the evolution of the $\mathbf{u}(t)$. Thus, the network has been trained so that $\mathbf{v}(t)$ is an approximation of $\mathbf{u}(t)$. Consequently after the determination of $\mathbf{p}$, the reservoir system has been to predict the values of $\mathbf{u}(t)$ for $t > 0$.

The response of this NN strongly depends on some parameters that characterize it and that must be accurately selected in the design of a Reservoir NN [11]. The first of this parameter is intuitively the size of the reservoir. Generally, the bigger is the reservoir, the better is the performance of the network. Since the training of the reservoir is cheap compared to other NN, it is common to have size of order 10^4 or bigger [12]. Another parameter is the sparsity of the $\mathbf{W}_{in}$ matrix. This parameter does not have great effect on performance. However, sparsity enables fast reservoir updates when using a sparse matrix representation. [11].

Probably the most important parameter of a reservoir NN is the spectral radius of the reservoir connection matrix $\mathbf{G}$. It scales the width of the distribution of its non zero elements. To ensure echo state properties in practical applications, it is sufficient to ensure that the spectral radius is less than 1. However, in tasks requiring longer memory of the input, the spectral radius should be greater as it affects the influence of previous inputs on the current state. Finally, the input $\mathbf{u}(t)$ can be scaled to keep it bounded and avoid outliers. The input scaling regulates the nonlinearity in the reservoir

representation $\mathbf{r}(t)$ and the effect of the input history. Thus, the input scaling is related to the spectral radius of $\mathbf{G}$.

To conclude this section, a autoencoder NN is described. These are a special case of feed forward NN and are trained to copy the input to the output. It is composed of two parts: an encoder function $\mathbf{h} = F(\mathbf{x})$ and a decoder that produces a reconstruction $\mathbf{r} = G(\mathbf{h})$ [4]. As consequence, $\mathbf{x} - \mathbf{r} \cong 0$.

As feed forward NN, can be trained using minibatch gradient descent following gradients computed by back-propagation but also recirculation can be used [4]. There are different types of autoencoders (sparse, desoising, variational etc.), however they are generally composed of an input layer, set of hidden (encoding) layers reducing the state dimension until the code layer $\mathbf{h}$. Then a set of decoding layers is used to reproduce the input. A schematic of a autoencoder NN is illustrated in Figure 4. For this type of networks, the parameters affecting its performance are the number of layers and the neurons per layer.

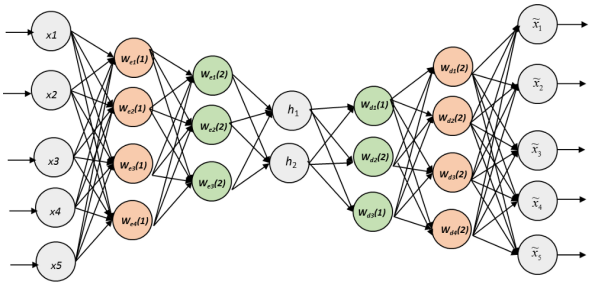


Figure 4: Schematic of a autoencoder NN [13].

3. Challenges for the Industry and the importance of the problem

As we introduced in the previous sections, the rise of the NN techniques. the exponential increase of the know-how and the amount of tools for its development calls for its application into industrial workframes to disrupt classic problems and its solutions. Apart from optimization processes, the recognition of patterns and other machine learning applications more common in the literature, we will focus our attention in the modelling and resolution of physically complex systems.

How so? we will study the implementation of such techniques into the resolution of physical models, written in terms of partial differential equations, with an eye on future learning processes that are only data-driven. This is a twofolded idea: using the efficient NN toolbox as a tool to improve the numerical resolution of PDEs and expanding our modelling capability of

complex systems. We will start and discuss mainly the first aim as it seems a more reachable target for the present work.

Indeed, there is a complicated milestone for the numerical resolution of PDEs: the scalability of the problem. The more spatially and temporary accurate we want our solution to be the bigger and better machines we will need. Not only that, thanks to the better understanding of the subyacent physics, the numerical calculations are now targeting many complex models such as multicomponent flows or visco-elastic materials that increase the expense of the computations even further. One might think that these kind of precissions and studies are only of academic interest, yet the industry is catching up fastly. With the validation of commercial codes, the first order accuracy has become a comodity for simulation and architecture departments of engineering companies and we are walking towards a situation where no more experimental/ test will be required. Indeed, the cutting-edge simulation departments are further and further away from the big number approach.

This pushes into the number of scales the numerical model is required to resolve and the accuracy of that calculation, and unfortunately, even with the lastest algorithms of resolution, the use of parallel supercomputers is still not enough to reach an industrially satisfactory simulation. This issue is often called the high dimensionality problem or even fronteer. The number of dimensions do not necessarily refer to the physical ones but to the number of grid points used for the simulation. In control processes and systems engineering applications might have thousands of such dimensions, or states to be taken into account for the final optimization.

And in those situations the classic finite methodology (differences, elements, volumes, etc.) simply won't work in a feasible a time scale. There have been many attempts in the past regarding this issue, NN is only the last wave of methods that try to deal with this problem. The most recent one are the well-known Monte Carlo techniques, that are nothing but an attempt to apply statistics in a crude manner to reduce the problem. More sofisticated alternatives, with a little more physics inside them are the Modal decomposition, where the solution of the problem is projected into its most fundamental modes or dimensions to then procede with its solution. Although of much more elegancy and even with applications to predict and study the evolution of the problem the mathematic apparatus around them is an important bottleneck, restraining this tools to a more accademic environment.

There is also an important aspect of the scalability problem that made the previous attemps not to reach a fully succesfully agreement: the complexity of the physics that we aim to solve with the numerical model. It was Lorenz one of the first that surprised himself finding out that the system of equations he was using to model the planetary weather was extremely sensible to the initial conditions. Chaos and complexity walk together with the modern models of fluids, making a new challenge appear: we don't only need to solve the

problem, we need to have a temporally accurate solution that includes the correct physics. And classic travel companions such as instabilities and the big elephant in the room, turbulence, makes this point a non-negligible one.

Summing up, the challenge the industry face at this moment is to solve their physical models, that might or not present chaotic and/or unstable behavior, in a very accurate manner in more and more complex geometries and to do this fast and numerically efficient. We believe that NN might help in both reducing the dimensionality of the problem and in the reproduction of the correct physical behavior of the model. Moreover, in a bolder affirmation, in the future even the physical phenomena will be captured or learned by the NN without requiring for a system of PDEs to model it.

4. Reservoir only prediction method

In order to study the applicability of the reservoir only prediction method, we have carried out numerical experiments calculating the usability time or "valid time" for two dynamic systems: **(1)** one of finite dimension, the Lorenz equations, and **(2)** one of infinite dimension, the Kuramoto-Sivashinski (KS) equation. "Valid time" t_v is a quantification of the duration of accurate prediction as defined in [14] with t_v as the elapsed time before the normalized error $E(t_v) < 0.4$.

On the other hand, we have tried to make a calibration of the free parameters of the reservoir looking for that applicability. Used free parameters were:

- β: Tikhonov regularization coefficient
- d: average degree of the random network
- D_r: reservoir size
- ρ: spectral size of the reservoir
- σ: input scaling

4.1. Lorenz equations

The first experiment was made with the Lorenz system, a three-dimensional system that presents chaotic behavior and that is given by the equations,

$$\begin{cases} \dfrac{dx}{dt} = -ax + ay \\ \dfrac{dy}{dt} = bx - y - xz \\ \dfrac{dz}{dt} = -cz + xy \end{cases}, \tag{4}$$

with a=10, b=28 and c=8/3.

For calibration, we have used 20 realizations with different random time intervals for each different parameter sets. And, in order to compare with results obtained in [14], we used the same integration step. The important parameters so far are ρ, σ and D_r [11]. We have taken the values of the parameters whose t_v has been statistically better,

$$\beta = 1.2 \quad , \quad \rho = 0.5 \quad , \quad \sigma = 0.2 \quad , \quad d = 9$$

and we have studied the dependence of t_v with D_r for these values for a training time $T = 20$ Lyapunov times. Training total normalized error is about 0.03. Results are shown in Figure 5.

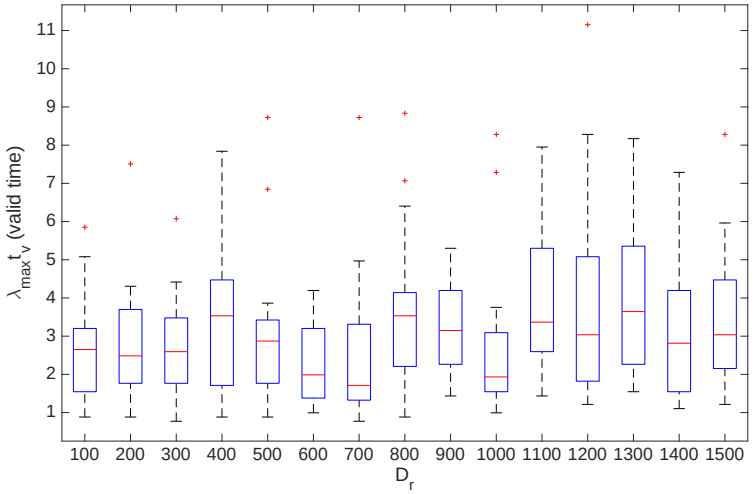


Figure 5: Reservoir size (D_r) dependence of the valid time. The line in the middle is the median value, the top and the bottom of the boxes are the first and the third quartiles, and whiskers delimit 1.5 times the interquartile limits. Crosses mark observations beyond the whiskers.

An example of prediction and the corresponding normalized error are shown in Figure 6, Figure 7 and Figure 8. Predictions starts at $\lambda_{max} t = 0$ and $D_r = 800$.

We can see that the valid time is not very long and covers on average about 2 Lyapunov times. However, dynamics is predicted quite well as we can see in the phase diagram (although rarely loses the dynamics).

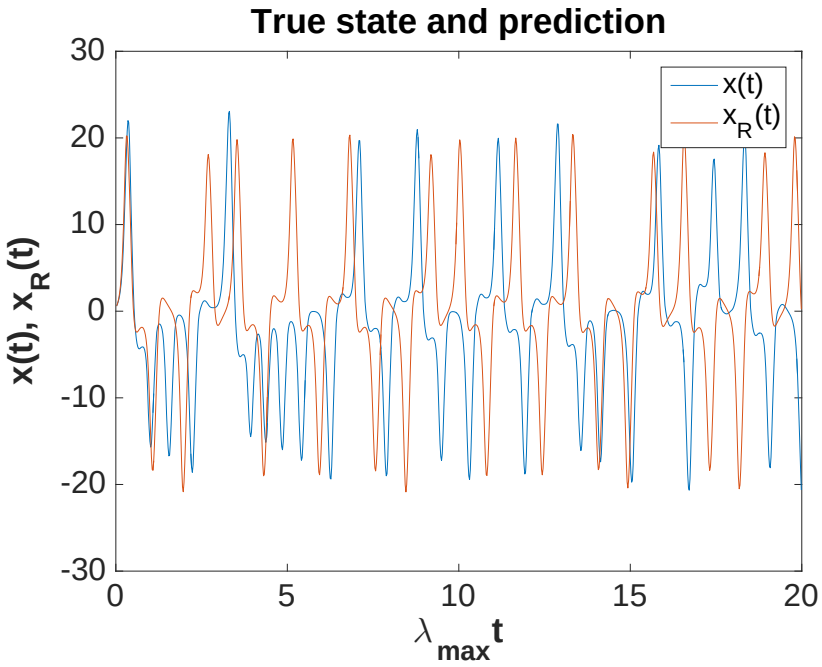


Figure 6: Prediction of the Lorenz system for variable $x(t)$.

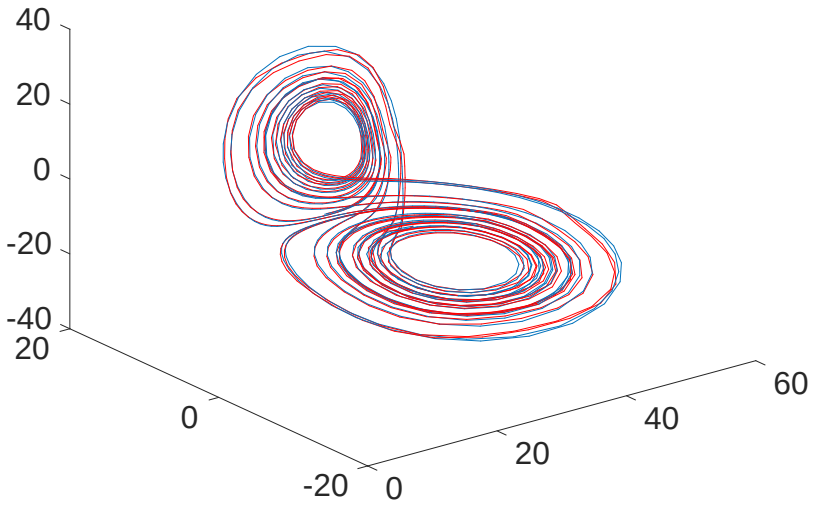


Figure 7: Prediction of the Lorenz system. Phase diagram. Blue line corresponding to exact results and red line corresponding to prediction.

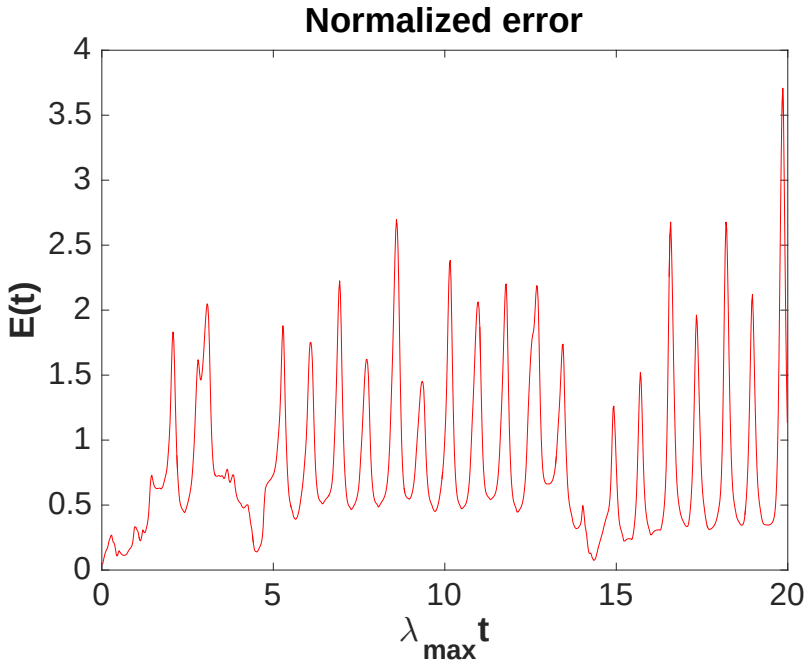


Figure 8: Normalized error for the Lorenz system.

4.2. Kuramoto-Sivashinsky equation

The second experiment was made with the one dimensional Kuramoto-Sivashinsky (KS) equation for $y(x,t)$,

$$y_t = -yy_x - y_{xx} - y_{xxxx} \tag{5}$$

We used the same system size ($L = 34$) with the same spatial discretization ($\Delta x \approx 0.547$) and time sampling ($\Delta t = 0.25$) used in [14].

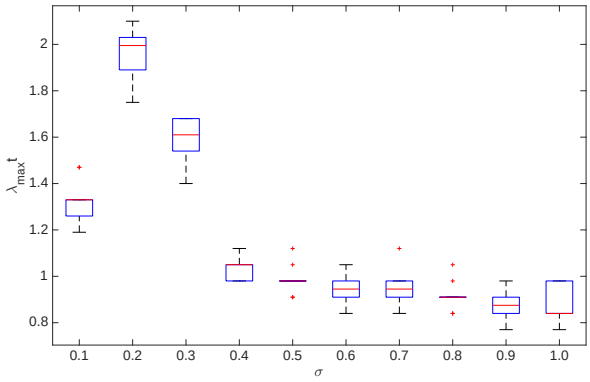
In this experiment we use 10 realizations in the same time interval. Best values found for reservoir parameters were:

$$\beta = 0.1 \quad , \quad \rho = 0.1 \quad , \quad \sigma = 0.2 \quad , \quad d = 3$$

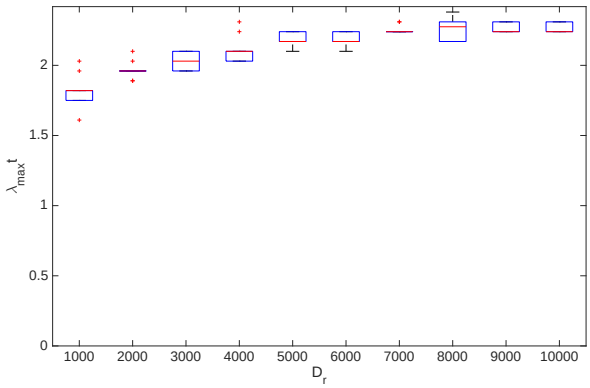
Training time $T = 14$ Lyapunov times and training total normalized error is about 0.05.

Figure 9 shows (a) σ dependence of the valid time and (b) D_r dependence of the valid time. We found that $\sigma = 0.2$ is the best value. Valid time is better for greater values of D_r, but the larger the reservoir size, the greater the computational cost.

Figure 10 shows a good realization of the experiment for the KS equation, with $t_v = 2.31$ Lyapunov times.



(a) σ dependence

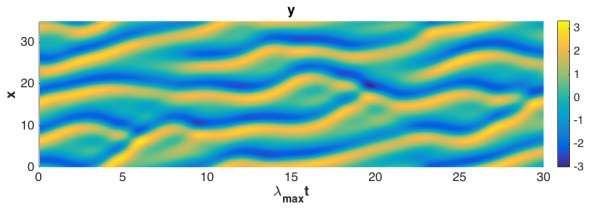


(b) D_r dependence

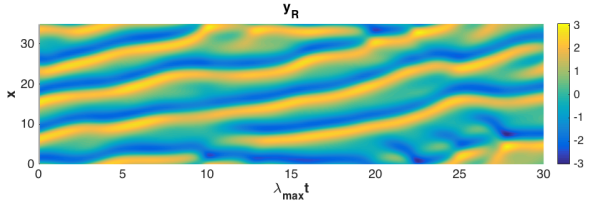
Figure 9: (a) σ dependence of the valid time with $D_r = 1500$ and (b) D_r dependence of the valid time with $\sigma = 0.2$. Other parameters values: $\beta = 0.1$, $\rho = 0.1$, $d = 3$.

5. Autoencoder

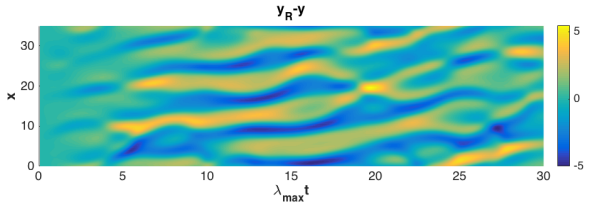
Another possibilitity to reduce the computational effort using NN is to define an autoencoder to reduce the variables where to solve the problem. This is the proposal from [2]. They published the usage an autoencoder which is initialized using a Restricted Boltzmann Machine and in a second step is fine trained as usually with a stochastic gradient descent. Using the encoding part of this autoencoder, they solve a time series reduced problem. Its output can be converted to the original dimensions using the decoding part. Similar approach has been proposed in [15].



(a) True state



(b) Predicted



(c) Difference (error)

Figure 10: Parameters values: $\beta = 0.1$, $\rho = 0.1$, $\sigma = 0.2$, $d = 3$, $D_r = 8000$. In this experiment: $t_v = 2.31$ Lyapunov times.

Unfortunately, due to the lack of enough information on the publication, the exact result could not be reproduced. Different autoencoders where tested with several KS results, being extremally difficult to have a valid result, needing a large time to train and they tend to overfit if the caothic regime has to be reproduced accuratelly. Due to this training time, this solution could be valid as a possible technique only if the autoencoder can be trained with a general case and can be used later (maybe with a fast and small retraining) for a problem with different initial conditions on the same domain.

6. Conclusion

Solving PDEs is still a computational challenge. The initial results of this workgroup have shown that Neural Networks could help, reproducing in simple cases the numerical solutions obtained with classical methods for Lorenz and KS equations. However, usage of these computational techniques have also an important work before they can be used in real problems.

Echo State Networks, although currently are not in the mainstream of the

Deep Learning techniques, can alleviate significantly the computational effort for solving the equations for future steps when enough training data have been generated. When this recurrent network has a reasonable number of hidden neurons, its training is very fast, because it needs mainly only two steps: proper initialization and an inverse of one matrix. However, the response of the new trained model depends strongly on some parameters that must be adjusted for each problem[11]:size of the reservoir, sparsity, distribution of nonzero elements, spectral radius, input scaling and leaking rate. These parameters must be calibrated for each problem although the author of [11] assesses that parameters for small cases can be used on bigger ones using transfer learning techniques. Unfortunately, there was no time enough to evaluate this assertion. In any case, the obtained results for Lorenz and KS equations seems that this method can be robust and fast enough to deserve more investigation. Additionally to the necessity of calibrating those hyper-parameters, the main barrier is the scalability of the Reservoir with the size of the problem. Bigger problems will require larger reservoirs. One possible solution is to use several on parallel, using a partition of the domain, as proposed by [1].

On the other size, creating a reduced model using autoencoders is another possible way to reduce the computational cost to solve this set of equations. Following [2], it is possible to reduce the initial spatial domain to a few set of modes or variables which encapsulate the dynamic of the problem, as is done commonly with more classical MOR techniques as POD. The advantage of using Neural Networks based autoencoders is that they can cope with nonlinear functions. In fact, autoencoders have been proposed as a substitute of PCA (related with POD) for this kind of problems. Again, using autoencoders is a computational challenge that need to find the correct values for the hyper-parameters which define the deep network: number of layers, number of neurons for each layer, activations, learning rates, etc. Only if the same configuration and initial weights can be used for any initial conditions of the same problem (or at least, can be transferred with a small fine tuning), the method can really be useful. However, this could not be checked during this week.

If we can assume that there is a general autoencoder for one specific problem independent of its initial conditions and only depending on the geometry and meshing of the domain, there exists the possibility of combining both methods to speed up the solution of PDEs. Initial data is generated solving the equations with traditional methods and after some delay, these data are reduced using the encoding part of the autoencoder, training in this reduced space a Reservoir network which will produce the solution which can be decoded using the second part of the autoencoder. This possibility will be part of the future work.

Additionally to the analyzed cases, during the survey of scientific literature, other methods for solving caothic PDEs have been identified which deserve a future analysis about their usability in the described scenarios. Among them,

[16] proposes to use reinforcement learning techniques. [17] defines a deep learning model based on convolutional and Long Short-Term Memory (LSTM) networks to predict the evolution of physical functions, applying it to fluid flows. The convolutional network reduces the dimensionality of the problem and LSTM are used to make the inference about the temporal evolution of this reduced set of variables.[18] combines a data-driven deep model with a model-driven model to predict the future evolution of a system. [19] follows a different approach to create a neural network from data. He trains a network using a cost function during the training which includes the physical model (i.e., the differential equation), so the NN is forced to approximate the data and to verify the differential equation simultaneously.

In summary, currently the authors cannot answer definitively any of the proposed questions. However, from the results of the executed experiments, the combination of both techniques seems to be a method to be explored deeply because can represent an alternative to current MOR techniques.

References

[1] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, "Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach," *Phys. Rev. Lett.*, vol. 120, p. 024102, Jan 2018.

[2] M. Wang, H.-X. Li, X. Chen, and Y. Chen, "Deep Learning-Based Model Reduction for Distributed Parameter Systems," *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS*, vol. 46, no. 12, 2016.

[3] G. Berkooz, P. Holmes, and J. L. Lumley, "The proper orthogonal decomposition in the analysis of turbulent flows," *Annual review of fluid mechanics*, vol. 25, no. 1, pp. 539–575, 1993.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[5] J. Berg and K. Nyström, "A unified deep artificial neural network approach to partial differential equations in complex geometries," *ArXiv e-prints*, Nov. 2017.

[6] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, Dec 1989.

[7] N. Yadav, A. Yadav, and M. Kumar, *An introduction to neural network methods for differential equations*. Springer, 2015.

[8] M. Egmont-Petersen, D. de Ridder, and H. Handels, "Image processing with neural networks—a review," *Pattern recognition*, vol. 35, no. 10, pp. 2279–2301, 2002.

[9] W. Cao, X. Wang, Z. Ming, and J. Gao, "A review on neural networks with random weights," *Neurocomputing*, vol. 275, pp. 278 – 287, 2018.

[10] X. Yan and X. G. Su, *Linear Regression Analysis*. WORLD SCIENTIFIC, 2009.

[11] M. Lukoševičius, "A Practical Guide to Applying Echo State Networks," in *G. Montavon, G. B. Orr, and K.-R. Müller (eds.) Neural Networks: Tricks of the Trade, 2nd ed. Springer LNCS 7700, 659-686*, 2012.

[12] F. Triefenbach, A. Jalalvand, B. Schrauwen, and J. pierre Martens, "Phoneme recognition with large hierarchical reservoirs," in *Advances in Neural Information Processing Systems 23* (J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, eds.), pp. 2307–2315, Curran Associates, Inc., 2010.

[13] "Deep autoencoders for collaborative filtering."

[14] J. Pathak, A. Wikner, R. Fussell, S. Chandra, B. Hunt, M. Girvan, and E. Ott, "Hyryd forecating of chaotic processes: Using machine learning in conjunction with knowledge-based model," *ArXiv e-prints*, Mar 2018.

[15] C. Qi and H.-X. Li, "Nonlinear dimension reduction based neural modeling for distributed parameter processes," *Chemical Engineering Science*, vol. 64, pp. 4164–4170, 2009.

[16] S. Wei, X. Jin, and H. Li, "General solutions for nonlinear differential equations: a deep reinforcement learning approach,"

[17] S. Wiewel, M. Becher, and N. Thuerey, "Latent-space Physics: Towards Learning the Temporal Evolution of Fluid Flow," feb 2018.

[18] Y. Long, X. She, and S. Mukhopadhyay, "HybridNet: Integrating Model-based and Data-driven Learning to Predict Evolution of Dynamical Systems," jun 2018.

[19] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations," nov 2017.