Physics-constrained data-driven methods for chaotic flows. Part II.

Luca Magri* & Nguyen Anh Khoa Doan†

December 3, 2019

Contents

*lm547@cam.ac.uk, University of Cambridge, Engineering Department, Trumpington St, Cambridge
†doan@tfd.mw.tum.de, Department of Mechanical Engineering, Technical University of Munich

Preambule

The present notes are an "alpha" version. They are not supposed to be a full review and the description and introduction are thus far from exhaustive. The main purpose of these notes is to illustrate some achievements of machine learning and to discuss how to implement these approaches. The interested reader is referred to the review works mentioned in the introduction for a more thorough overview of existing work. Additionally, notations may not be fully consistent throughout the manuscript.

1 Introduction to machine learning for turbulence

The recent growth in data-driven methods and machine learning techniques can be attributed, among other things, to the convergence of (i) the availability of *big data*, (ii) the advances in computational hardware and (iii) the development of sophisticated algorithms [1]. This made the use of machine learning algorithms practical and renewed the interest in these given that they provide a modular and agile modelling framework. However, while successful in many fields, the applicability of machine learning to fluid mechanics remains relatively limited and is still a very active area of research.

Compared to traditional machine learning tasks such as image recognition, Go playing or speech recognition, turbulence suffers from being a small data field. This originates from the fact that, on one hand, in real experiments, measurements are only sparsely available and that, on the other hand, numerical simulations remain limited to relatively short physical time. Additionally, turbulence is a chaotic and strongly nonlinear phenomenon. This implies that it is extremely sensitive to initial conditions and any perturbations in its state. Hence, being able to accurately predict its evolution becomes particularly challenging as any error will results in an exponentially diverging evolution from the real fluid system evolution. Finally, how to ensure the physicality of the machine learning prediction becomes a crucial question.

Despite these difficulties, machine learning has had several success in the study of turbulence. A review of these previous work and the challenges still to be resolved are well detailed in several recent reviews [1–4]. The main achievements of machine learning for fluid mechanics can roughly be separated in two categories: (i) flow feature extraction and (ii) flow modelling. Regarding the first category, these various works mainly uses convolution neural networks or autoencoders to either discover a low-order representation of the flow or to infer flow features from coarse or partial information [5–7]. For flow modelling, the most common family of work have been devoted to developing closure models for the Reynolds stress in Reynolds-Averaged Navier Stokes (RANS) simulations or subgrid scale models for Large Eddy Simulation (LES) [2, 8–10]. Another approach explored recently is based on developing networks which can model and predict the flow itself [11, 12]. The examples cited above are far from the complete list of areas of turbulence research where machine learning can have an impact nor do they represent all the work done in this area. Nonetheless, they illustrate areas where machine learning has already allowed for some progress. As this is still a very active field of research, these advanced applications will not be tackled in the present course and the focus, here, will be on how to embed physical knowledge in machine learning algorithm which is one of the fundamental challenge facing machine learning in turbulence research.

In particular, the present notes will focus on two aspects where neural networks have achieved good results when used in combination with physical knowledge. In section 2, we will present how feedforward neural networks can be used to approximate the solution of partial differential equations (PDEs). In section 3, we introduce recurrent neural networks, and in particular Echo State Networks, and illustrate how they can be used to model dynamical systems. In both of these examples, emphasis is put on how to include physical knowledge during the network training to ensure the physicality of the prediction of these networks.

2 Partial Differential Equation (PDE) solution approximation with artificial neural networks

Many chaotic systems, including turbulent flows, are governed by partial differential equations. So, a natural first approach to predict the behaviour of these turbulent flows is to make use of the universal approximator property of feedforward neural network [13] and to use these to approximate the solution of a given PDE. In this section, feedforward neural networks are first briefly described. Then a method to implement physical constraints during the training of these networks is described.

2.1 Artificial feedforward neural network

An artificial neural network (ANN) consists of interconnected neurons. A single neuron is a function $y = f(\boldsymbol{u}, \Theta)$ where $\boldsymbol{u}$ is the input vector, Θ the neuron parameters vector (also called weights) and f is the neuron activation function. In principle, any function can be used as an activation function but general choice are usually limited to the sigmoid function, the hyperbolic tangent (tanh) or the rectified linear unit (ReLU).
Several connected neurons form an artificial neural networks which can be arranged in a series of layers thereby becoming a deep neural network. An example of a deep neural network is represented in Fig. 1. In the following description, only feed-forward neural networks will be considered, meaning that the outputs of a layer are not fedback to the previous layer. Additionally, each neuron is fully connected meaning that all neurons in the layer $i-1$ are connected to all neurons of layer i through a weights matrix $\boldsymbol{W}_i$. From this, the intermediate output of the hidden layer i, $\boldsymbol{Z}_i$, can be written as:

$$\boldsymbol{Z}_i = \boldsymbol{W}_i^T \boldsymbol{x}_{i-1} + \boldsymbol{b}_i \tag{1}$$

where $\boldsymbol{x}_{i-1}$ is the output of layer $i-1$ and b_i is a bias term. The output of layer i is then taken as the term-wise activation of $\boldsymbol{Z}_i$:

$$\boldsymbol{x}_i = f(\boldsymbol{Z}_i) = f\left(\boldsymbol{W}_i^T \boldsymbol{x}_{i-1} + \boldsymbol{b}_i\right) \tag{2}$$

The output of the final layer is the "prediction" from the entire neural network, here noted $\widehat{\boldsymbol{y}}$.

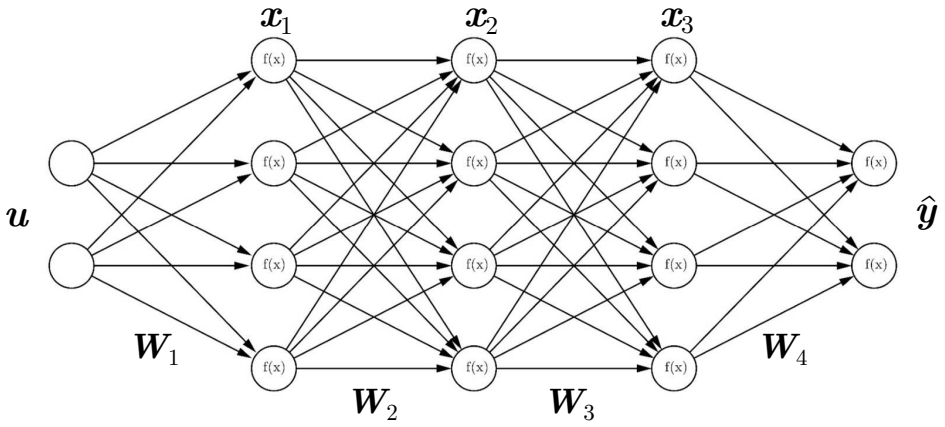

Figure 1: Representation of a deep neural network.

ANNs have been shown to be universal approximators meaning that a large enough ANN can approximate any analytical function in its subspace [13]. This makes them powerful tool to approximate the solution to any problem, but this theorem does not indicate if there is an easy way of finding the necessary parameters to ensure that the ANN approximates the desired function. Typically, to obtain these parameters, ANNs are trained using input and expected outputs data. This allows for the ANN to approximate the underlying function linking the input and output data of the physical system to be modelled.

Hence, the training of a neural network takes the form of an optimization problem where the neurons weights and biases have to be determined so as to ensure a minimal error between the real output values and the predicted output by the ANN. For this minimization, one typically uses the Mean Squared Error (MSE) error:

$$MSE = \frac{1}{N}\sum_{i}^{N} ||\boldsymbol{y_i} - \widehat{\boldsymbol{y}}_i||^2 \tag{3}$$

where $||\cdot||$ is the L2-norm, N is the number of points in the training dataset, $\boldsymbol{y}_i$ is the exact output for the i-th input in the dataset and $\widehat{\boldsymbol{y}}_i$ is the ANN prediction for that same input. The optimization is generally performed using a stochastic gradient descent method with backpropagation. Details on the various training algorithms for neural network are outside the scope of these lecture notes, but the interested reader is referred to [14] for a detailed discussion on those topics.

2.2 Physics-informed neural network

Various approaches have been proposed in the past to embed physical knowledge into neural networks. These can roughly be categorised into two categories depending on how this embedding is performed, either via a modification of the loss function or either via a specific architecture of the network. Both approaches are briefly summarized next.

1. Physical loss function: In this approach, the loss function used for the training of the neural network is modified by adding physical constraints. This approach has been used for example by [15] and is described next. Let us assume that the physical system under study is governed by the following partial differential equation:

$$\mathcal{F}(\boldsymbol{y}) \equiv \frac{d\boldsymbol{y}}{dt} + \mathcal{N}(\boldsymbol{y}) = 0 \tag{4}$$

solved over the domain $\boldsymbol{x} \in \Omega$ and for $t \in [0, T]$, with associated boundary conditions. $\mathcal{N}$ is a nonlinear operator. One can then generate a set of N_p collocation points, $\{(\boldsymbol{x}_i^p, t_i^p)\}_{i=1,\ldots,N_p}$, where the physical residual $\mathcal{F}$ can be estimated and this residual also has to be minimized to ensure that the trained ANN satisfies the physics of the system. This additional loss term then takes the form:

$$L_p = \frac{1}{N_p}\sum_{i=1}^{N_p} ||\mathcal{F}(\widehat{\boldsymbol{y}}(\boldsymbol{x}_i^p, t_i^p))||^2 \tag{5}$$

In the above expression, the choice of the collocation points plays a crucial role in determining the quality of the trained network. A rule of thumb indicates that using more points generally improves the trained network, but it results in a larger training time. Additionally, the collocation points should be picked judiciously so as to cover a wide range of the domain over which the PDE has to be solved. More points should be used in regions with strong gradients. Practically, to estimate the derivatives in $\mathcal{F}$ to obtain L_p, one should use the advances in automatic differentiation for machine learning [16].

In addition, for PDEs, boundary and initial conditions are known too and a similar loss associated to these conditions can be estimated at the boundary and initial points, $\left\{(\boldsymbol{x}_i^B, t_i^B)\right\}_{i=1,\ldots,N_{BC}}$:

$$L_{BC} = \frac{1}{N_{BC}}\sum_{i=1}^{N_{BC}} ||\mathcal{B}(\widehat{\boldsymbol{y}}(\boldsymbol{x}_i^{BC}, t_i^{BC}))||^2 \tag{6}$$

where $\mathcal{B}$ is the appropriate expression depending on the types of boundary conditions and the definition of the initial conditions.

Given the points above, the loss function for a physics-informed feedforward neural network is:

$$\begin{aligned} L =& MSE + L_p + L_{BC} \\ =& \frac{1}{N}\sum_{i=1}^{N} ||\boldsymbol{y_i} - \widehat{\boldsymbol{y}}_i||^2 + \frac{1}{N_p}\sum_{p=1}^{N_p} ||\mathcal{F}(\widehat{\boldsymbol{y}}(\boldsymbol{x}_p, t_p))||^2 + \frac{1}{N_{BC}}\sum_{i=1}^{N_{BC}} ||\mathcal{B}(\widehat{\boldsymbol{y}}(\boldsymbol{x}_i^{BC}, t_i^{BC}))||^2 \end{aligned} \tag{7}$$

The training of the physics-informed ANN can still be made using a normal stochastic gradient descent method or a simple gradient descent method. However, an important aspect to be remembered is that the ANN will only be able to approximate functions in its subspace. This means that as the solution of the PDE becomes more and more complicated, the size (number of neurons) and depth (number of layers) of the ANN will have to increase markedly. This makes the training of the ANN increasingly costly and complicated.

2. Architecture of the network: This approach relies on embedding the physical constraint into the architecture of the network by adding specific layers. While very convenient, as it allows the same training method to be maintained, as for conventional neural network, such as backpropagation, the type and architecture of those "physical" layers are specific to the problem under study. Hence, they will not be described in details here. Given that this is still a relatively recent method, only few examples exist among which [9] where a layer mimicking the isotropic tensors in turbulence is used to impose their invariance.

3 Recurrent neural network for chaotic systems

In this section, we will discuss approaches to the modelling of chaotic systems using a specific type of neural network. Compared to the previous section, the objective is to develop a network that is a good approximation of a dynamical system given a training time series of this system. More precisely, given an input training signals $\boldsymbol{u}(n)$ of dimension N_u and outputs $\boldsymbol{y}(n)$ of dimension N_y occupying a discrete time domain $n = 0, ..., (N-1)\Delta t$, the objective is to develop a network with output $\widehat{\boldsymbol{y}}(n)$ which matches the $\boldsymbol{y}(n)$ as closely as possible.
Given that this problem deals with time series, recurrent neural networks (RNNs) which are specifically designed to deal with such type of data will be discussed here. In a RNN, similarly to feedforward neural network, neurons (or units) are connected through links which enable activations to propagate through the network. However, in contrast to feedforward neural networks, the connection within RNN possesses cycles meaning that the neurons contain a feedback loop. The existence of these cycles enables the RNN to develop a self-sustained temporal activation dynamics (hence RNNs are dynamical systems) and to possess a dynamical memory of the input excitation. In this section, we will only describe two specific RNNs: the Long Short Term Memory (LSTM) neuron and the Echo State Network (ESN).

3.1 Long short-term memory neuron

Only a brief description of LSTM is provided here as they are not the focus of these notes. The long short term memory is one of the most widespread recurrent neural network used. It was introduced by Hochreiter and Schmidhuber [17]. A typical representation of an LSTM neuron is presented in Fig. 2.
The idea behind the LSTM is to not entirely update the whole hidden state each time and protect the state from being overwritten with useless information. As a result, the LSTM is selective in the information used to update its state $\boldsymbol{x}_n$ and the output $\boldsymbol{y}_n$. The evolution of the state and output in the LSTM is interpreted as follows:

- What to forget (forget gate):

$$\boldsymbol{r}_n = f\left(\boldsymbol{W}_r \boldsymbol{y}_{n-1} + \boldsymbol{U}_r \boldsymbol{u}_n + \boldsymbol{b}_r\right) \tag{8}$$

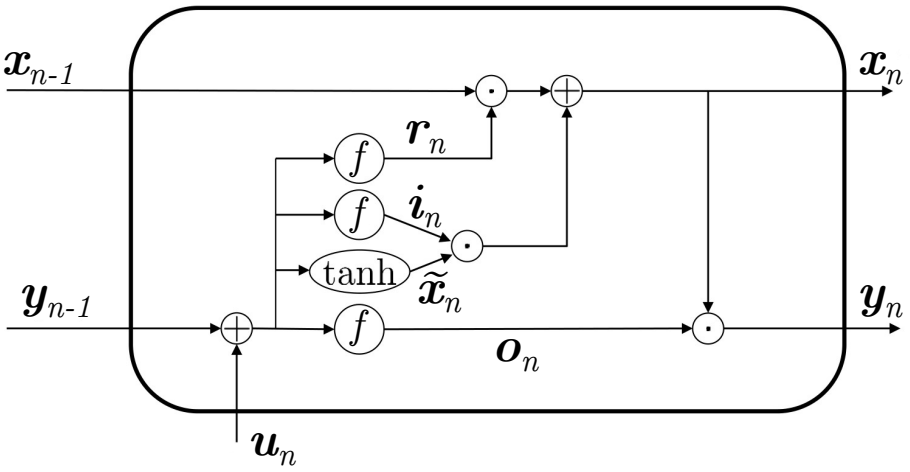

Figure 2: Representation of an LSTM network.

- What to write (input gate):

$$\boldsymbol{i}_n = f\left(\boldsymbol{W}_i \boldsymbol{y}_{n-1} + \boldsymbol{U}_i \boldsymbol{u}_n + \boldsymbol{b}_i\right) \tag{9}$$

- What to read (output gate):

$$\boldsymbol{o}_n = f\left(\boldsymbol{W}_o \boldsymbol{y}_{n-1} + \boldsymbol{U}_o \boldsymbol{u}_n + \boldsymbol{b}_o\right) \tag{10}$$

The update of the LSTM state is then made via:

$$\boldsymbol{x}_n = \boldsymbol{r}_n \odot \boldsymbol{x}_{n-1} + \boldsymbol{i}_n \odot \tanh\left(\boldsymbol{W}_y \boldsymbol{y}_{n-1} + \boldsymbol{U}_y \boldsymbol{u}_n + \boldsymbol{b}_y\right) \tag{11}$$

where $\odot$ is the Hadamard product.
And the update of the output is finally:

$$\boldsymbol{y}_n = \boldsymbol{o}_n \odot \boldsymbol{x}_n \tag{12}$$

In the equations (8) to (12), the $\boldsymbol{W}.$, $\boldsymbol{U}.$ and $\boldsymbol{b}.$ are the weights and biases to be optimized during the training of the LSTM.

3.2 Echo State Network

Echo State Network (ESN) is a form of Reservoir Computing. The basis of reservoir computing were introduced and developed independently by Wolfgang Mass with Liquid State Machine [18] and Herbert Jaeger with Echo State Networks. Both approaches were motivated by several aspects which inhibited the development and widespread use of conventional RNNs, such as the LSTM [19]. The first reason was that conventional RNNs are particularly difficult to train using typical gradient-descent with backpropagation algorithms. The second is that it was observed that RNNs could work well without perfect adaptation of all the network weights and that the dominant changes in a network performance were coming mainly from the output weights [20]. As a result of these considerations, the principal idea of reservoir computing was to use a fixed, random, large recurrent neural network, called the "reservoir", which would be driven by the inputs and whose outputs would come from a linear combination of the reservoir neurons. In what follows, we will only be focusing on the ESN and not on the Liquid State Machine.
A typical representation of an ESN is shown in Fig. 3. In its simplest form, an ESN is composed of 3 parts: the input layer, the dynamical reservoir and the output layer. In contrast to conventional recurrent neural networks, the weights of the input layer and the adjacency matrix in the reservoir are fixed and only the output layer is trained. More specifically:

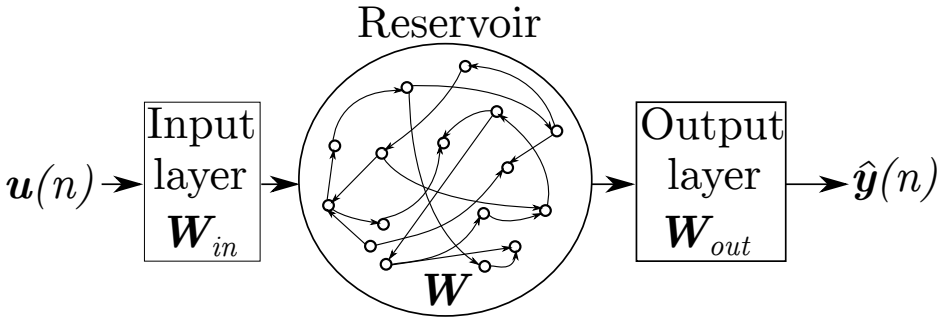

Figure 3: Typical Echo State Network architecture.

1. The input layer, $\boldsymbol{W}_{in}$, is a matrix of dimension $N_x \times N_u$ where the elements are sampled from a uniform distribution over the range $[-\sigma, \sigma]$ (or using a normal distribution with standard deviation σ). The values of σ (called the input scaling) indicates the sensitivity of the reservoir neurons to the input excitation and will tune the amount of nonlinearity (through the saturation of the activation function) in the reservoir. This is typically a sparse matrix where each neuron is connected to a small number inputs or even only one. For tasks where there are extremely different sensitivities to the inputs, each column of $\boldsymbol{W}_{in}$ can be scaled differently resulting in N_u different input scalings.

2. The dynamical reservoir is composed of N_x neurons which are connected through an adjacency (or recurrent) matrix $\boldsymbol{W}$ of dimension $N_x \times N_x$. In general, the reservoir is (i) big, (ii) sparsely and (iii) randomly connected. This aims at obtaining many reservoir activation signals which are only loosely coupled and different. The neuron states, $\boldsymbol{x}$, also called echoes (of the input history), evolve according to:

$$\boldsymbol{x}(n+1) = f\left(\boldsymbol{W}_{in}\boldsymbol{u}(n+1) + \boldsymbol{W}\boldsymbol{x}(n)\right) \tag{13}$$

where f is the neuron activation function, typically a sigmoid function such as the tanh. This equation (13) is the central equation governing the dynamics of the reservoir. As can be seen from that expression, it is a recursive equation. To highlight this, we can rewrite that equation as:

$$\boldsymbol{x}(n+1) = f\left(\boldsymbol{W}_{in}\boldsymbol{u}(n+1) + \boldsymbol{W}\left[f\left(\boldsymbol{W}_{in}\boldsymbol{u}(n) + \boldsymbol{W}\boldsymbol{x}(n-1)\right)\right]\right) \tag{14}$$

This shows that the future state of the reservoir is depending on its history, but also on the history of the inputs and that the reservoir is indeed a dynamical system.

The most important feature of $\boldsymbol{W}$ is that $\boldsymbol{W}$ must respect the echo state property [19]. This property, in essence, states that the effect of a previous state $\boldsymbol{x}(n)$ and a previous input $\boldsymbol{u}(n)$ on a future state $\boldsymbol{x}(n+k)$ vanishes gradually as time passes (i.e., for $k \to \infty$). A sufficient (but not necessary) condition to achieve that, in reservoir with tanh activation and zero input, is to ensure that the spectral radius of $\boldsymbol{W}$ is smaller than 1. Despite this condition, for some tasks, some reservoirs with spectral radius larger than 1 have been observed to also be able to perform well [19].

Typically, the adjacency matrix $\boldsymbol{W}$ is initialised as a sparse matrix with an average connectivity $\langle d \rangle$ and whose non-zero elements are sampled from a uniform distribution between $[-1, 1]$. The matrix $\boldsymbol{W}$ is then rescaled to ensure that its spectral radius is equal to a given value $\Lambda < 1$. This value Λ governs the amount of memory and nonlinearity in the reservoir. Observations have shown that for systems with long memory, values of Λ close to 1 are preferable. Additionally, regarding the sparsity of $\boldsymbol{W}$, previous studies showed that sparse matrices generally performed slightly better than fully connected ones but that this parameter $\langle d \rangle$ only has a minor impact on the performance of the ESN. Nonetheless, having a sparse matrix improves the scalability of the ESN in terms of computation time as the cost of the network estimation for sparse network scales with N_x and not N_x^2 as in dense network [19].

3. The readout layer, $\boldsymbol{W}_{out}$, is a matrix of dimension $N_y \times N_x$. This provides the prediction from the reservoir state as a simple linear combination:

$$\widehat{\boldsymbol{y}}(n) = \boldsymbol{W}_{out}\boldsymbol{x}(n) \tag{15}$$

From this, the learning of the output weight $\boldsymbol{W}_{out}$ consists in the optimization of the weights of $\boldsymbol{W}_{out}$ to minimize the MSE between the prediction and the target values. In theory, the optimal $\boldsymbol{W}_{out}$ matrix can be obtained directly using a Moore-Penrose pseudoinverse, i.e. $\boldsymbol{W}_{out} = \boldsymbol{Y}\boldsymbol{X}^{+}$ where $^{+}$ is the pseudo-inverse, $\boldsymbol{X}$ is a column-concatenation of all $\boldsymbol{x}(n)$ produced by exciting the reservoir with all the $\boldsymbol{u}(n)$ and $\boldsymbol{Y}$ are all the associated target $\boldsymbol{y}(n)$ column-concatenated. However, this approach generally only works for small reservoir as the Moore-Penrose pseudoinverse is computationally and memory-wise expensive to perform. Furthermore, this approach can run the risks of overfitting when the reservoir is large compared to the amount of training data, N. This overfitting generally translates into large values in $\boldsymbol{W}_{out}$. Indeed, extremely large values in $\boldsymbol{W}_{out}$ indicates that the ESN exploits very small differences in the reservoir states to obtain the desired outputs. However, exploiting this can lead to unstable reservoir when running the reservoir in a generative (also called natural) set-up (where the output of the reservoir is looped back as its input) or when the states are slightly perturbed. Thus, conventionally, a ridge or Tikhonov regularization is rather used which aims at preventing large values in $\boldsymbol{W}_{out}$ by limiting its norm. This corresponds to minimizing the following loss function:

$$L_r = \frac{1}{N}\sum_{i}^{N} ||\boldsymbol{y_i} - \widehat{\boldsymbol{y}}_i||^2 + \gamma||\boldsymbol{w}_{out,i}||^2 \tag{16}$$

where the first part is the MSE between the data available and the ESN prediction, $\boldsymbol{w}_{out,i}$ is the i-th row of $\boldsymbol{W}_{out}$ and $||\cdot||$ is the L2-norm. This latter term, $\gamma||\boldsymbol{w}_{out,i}||^2$, penalizes large values in $\boldsymbol{W}_{out}$ which prevents the risk of overfitting and allow for better numerical stability in generative mode. Furthermore, the γ factor acts as a balancing factor between fitting the data and avoiding large values in $\boldsymbol{W}_{out}$. In general, this term should be as small as possible while ensuring that the reservoir remains stable and that there is no overfitting.

The optimal $\boldsymbol{W}_{out}$ which minimizes L_r can be obtained analytically using the ridge regression:

$$\boldsymbol{W}_{out} = \boldsymbol{Y}\boldsymbol{X}^T\left(\boldsymbol{X}\boldsymbol{X}^T + \gamma\boldsymbol{I}\right)^{-1} \tag{17}$$

where $\boldsymbol{I}$ is the identity matrix of size N_x. Algebraically, it can be seen that γ improves the regularity of the $\boldsymbol{W}_{out}$ matrix by improving the conditioning before the matrix-inversion operation. The value of γ should be adjusted for concrete instantiation of the ESN and can vary by several orders of magnitude.

Additional details on reservoir computing and ESN and how to design them efficiently can be found in [19, 21].

3.2.1 Architecture variants

Small variants on the base architecture presented above includes the possibility of having a feedback matrix where the output of the ESN is fedback to the reservoir, shown with the red parts in Fig. 4. The evolution equation for the reservoir neurons is then:

$$\boldsymbol{x}(n+1) = f\left(\boldsymbol{W}_{in}\boldsymbol{u}(n+1) + \boldsymbol{W}\boldsymbol{x}(n) + \boldsymbol{W}_{FB}\boldsymbol{y}(n)\right) \tag{18}$$

In addition, a bias, shown in blue in Fig. 4, can also be added to the ESN at the input and output layers and a direct link between the input and output, shown in green in Fig. 4, can be added. In that case, the reservoir dynamics is governed by:

$$\boldsymbol{x}(n+1) = f\left(\boldsymbol{W}_{in}[1;\boldsymbol{u}(n+1)] + \boldsymbol{W}\boldsymbol{x}(n)\right) \tag{19}$$

where ; indicates a vertical concatenation. The output is obtained as:

$$\boldsymbol{y}(n) = \boldsymbol{W}_{out}[1;\boldsymbol{u}(n);\boldsymbol{x}(n)] \tag{20}$$

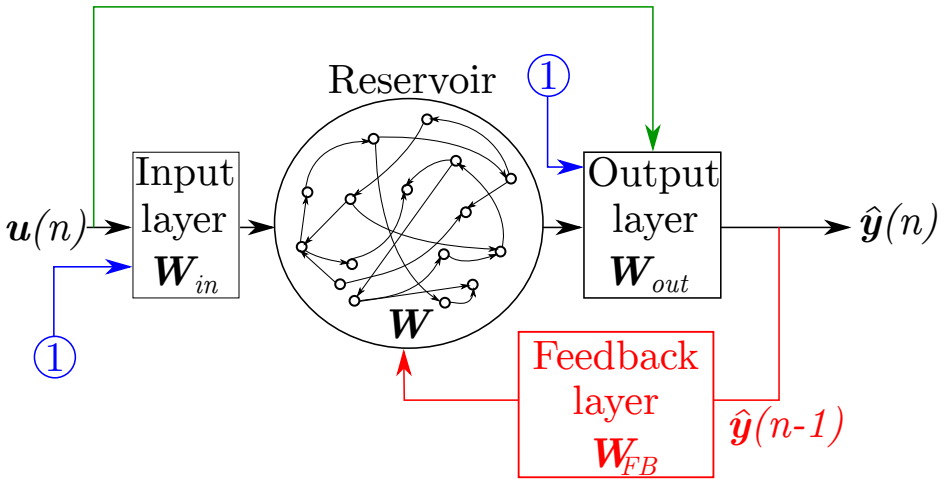

Figure 4: Variation on the typical ESN architecture. The addition of biases is indicated in blue with the 1 value, the feedback loop is in red and the direct input/output link is in green.

Finally, another commonly used variant of the standard ESN is the leaky ESN where a "leaky" integration of the previous reservoir states is performed. In that architecture, the reservoir dynamics is governed by:

$$\boldsymbol{x}(n+1) = \alpha f\left(\boldsymbol{W}_{in}\boldsymbol{u}(n+1) + \boldsymbol{W}\boldsymbol{x}(n)\right) + (1-\alpha)\boldsymbol{x}(n) \tag{21}$$

where $\alpha \in]0,1]$ is the leakage parameter. This approach allows to control the "speed" (or inertia) of the reservoir dynamics with small values of α inducing reservoir which react slowly to the input as their updated values is close to the previous one. Indeed, if $\alpha = 0$, the reservoir does not evolve at all and the reservoir states are maintained indefinitely. Conversely, if $\alpha = 1$, the new reservoir states $\boldsymbol{x}(n+1)$ are entirely replaced by the new term coming from the combination of the excitation and the other reservoir neurons.

3.2.2 Generative/Prediction mode and physical constraints

The objective of the present section was to try to model dynamical systems. So to actually compare the capability of the ESN to model an existing system, it is the generative (or prediction) response of the ESN which has to be compared. This is obtained by looping back the output of the ESN as its input so that the input becomes the output at the previous time step i.e. $\boldsymbol{u}(n+1) = \hat{\boldsymbol{y}}(n)$. This configuration is illustrated in Fig. 5. Then, given the same initial conditions, one can analyse how close to the exact system the ESN is by comparing the natural evolution of the ESN with the evolution of the exact system. It is also possible to embed/constrain physical knowledge in the training of the ESN using a extension to the time domain of the method presented in section 2.2 and presented in [22]. Assuming that the system under study is also governed by Eq. (4), one can also consider collocation points for the ESN which are for time $t > (N-1)\Delta t$ after the training. Then, if one consider N_p collocation points, the prediction

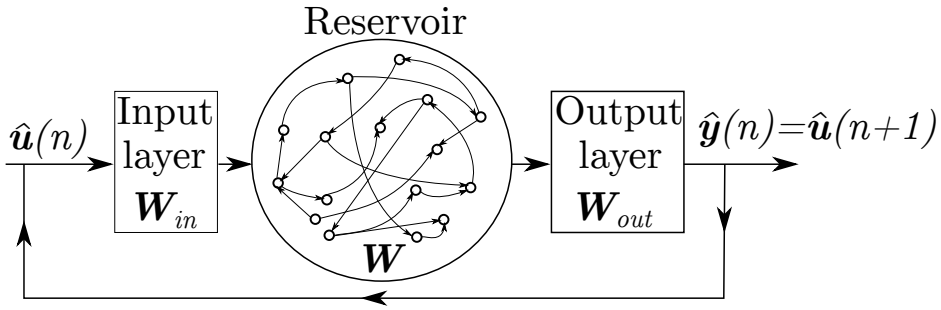

Figure 5: ESN in prediction (or generative) mode where the output is looped back as the input of the ESN.

from the ESN over a time period $t \in [N\Delta t, (N + N_p - 1)\Delta t]$, noted $\{\widehat{\boldsymbol{y}}(n_p)\}_{n_p=1,...,N_p}$, can be collected and the physical residual estimated as:

$$L_p = \frac{1}{N_p} \sum_{n_p=1}^{N_p} ||\mathcal{F}(\widehat{\boldsymbol{y}}(n_p))||^2 \tag{22}$$

By combining the MSE and this physical residual, a new loss function can be used for the training of the ESN which does not rely on regularizing $\boldsymbol{W}_{out}$ by considering its norm but by using the physical residual at these additional collocation points:

$$L_{phys} = \frac{1}{N} \sum_{i=1}^{N} ||\boldsymbol{y_i} - \widehat{\boldsymbol{y}}_i||^2 + \frac{1}{N_p} \sum_{n_p=1}^{N_p} ||\mathcal{F}(\widehat{\boldsymbol{y}}(n_p))||^2 \tag{23}$$

This latter term in Eq. 23 acts as a physics-based regularisation factor compared to the original regularisation which is done just on the norm of $\boldsymbol{W}_{out}$. However, while this new loss function can improve the resulting trained ESN, its main drawback is that there is no quick method to compute the optimal $\boldsymbol{W}_{out}$ and one has to use a gradient-based optimization to obtain the optimal $\boldsymbol{W}_{out}$.

References

[1] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. Machine Learning for Fluid Mechanics. *Annu. Rev. Fluid Mech.*, 52(1):477–508, 2020.

[2] J. Nathan Kutz. Deep learning in fluid dynamics. *J. Fluid Mech.*, 814:1–4, 2017.

[3] M P Brenner. Perspective on machine learning for advancing fluid mechanics. *Phys. Rev. Fluids*, 4(10):100501, 2019.

[4] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. Turbulence Modeling in the Age of Data. *Annu. Rev. Fluid Mech.*, 51:357–377, 2019.

[5] Michele Milano and Petros Koumoutsakos. Neural network modeling for near wall turbulent flow. *J. Comput. Phys.*, 182(1):1–26, 2002.

[6] Brendan Colvert, Mohamad Alsalman, and Eva Kanso. Classifying vortex wakes using neural networks. *Bioinspiration and Biomimetics*, 13(2), 2018.

[7] Kai Fukami, Koji Fukagata, and Kunihiko Taira. Super-resolution reconstruction of turbulent flows with machine learning. *J. Fluid Mech.*, 870:106–120, 2019.

[8] Romit Maulik, Omer San, Jamey D. Jacob, and Christopher Crick. Sub-grid scale model classification and blending through deep learning. *J. Fluid Mech.*, 870:784–812, 2019.

[9] Julia Ling, Andrew Kurzawski, and Jeremy Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *J. Fluid Mech.*, 807:155–166, 2016.

[10] J. Zhang, J. Mi, P. Li, F. Wang, and B. B. Dally. Moderate or intense low-oxygen dilution combustion of methane diluted by CO<inf>2</inf> and N<inf>2</inf>. *Energy and Fuels*, 29(7):4576–4585, 2015.

[11] P A Srinivasan, L Guastoni, H Azizpour, P Schlatter, and R Vinuesa. Predictions of turbulent shear flows using deep neural networks. *Phys. Rev. Fluids*, page 054603, 2019.

[12] Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nat. Commun.*, 9:4950, 2018.

[13] Kurt Hornik. Approximation Capabilities of Multilayer Neural Network. *Neural Networks*, 4:251–257, 1991.

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

[15] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations. *J. Comput. Phys.*, 378:686–707, 2019.

[16] A. G. Baydin, Barak A Pearlmutter, A. A. Radul, and Jeffrey Mark Siskind. Automatic Differentiation in Machine Learning: a Survey. *J. Mach. Learn. Res.*, 18:1–43, 2018.

[17] Sepp Hochreiter. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[18] Wolfgang Maass, T. Natschläger, and H. Markram. Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. *Neural Comput.*, 14(11):2531–2560, 2002.

[19] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Comput. Sci. Rev.*, 3(3):127–149, 2009.

[20] Ulf D. Schiller and Jochen J. Steil. Analyzing the weight dynamics of recurrent learning algorithms. *Neurocomputing*, 63:5–23, 2005.

[21] Mantas Lukoševičius. A Practical Guide to Applying Echo State Networks. In G. Montavon, G. B. Orr, and K.-R. Muller, editors, *Neural Networks: Tricks of the Trade*. Springer, 2012.

[22] N. A. K. Doan, W. Polifke, and L. Magri. Physics-Informed Echo State Networks for Chaotic Systems Forecasting. In J. M. F. Rodrigues, P. J. S. Cardoso, J. Monteiro, R. Lam, V. V. Krzhizhanovskaya, M. H. Lees, J. J. Dongarra, and P. M.A. Sloot, editors, *Comput. Sci. – ICCS 2019*, pages 192–198. Springer International Publishing, 2019.