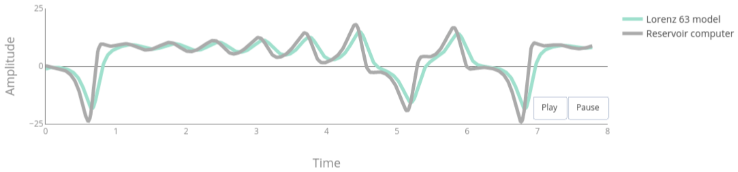
Learning chaotic dynamics via reservoir computing

Anton Pershin[1, 2]
anton.pershin@physics.ox.ac.uk
[1]Atmospheric, Oceanic and Planetary Physics, University of Oxford
[2]School of Mathematics, University of Leeds



Machine Learning Study Group, University of Leeds, Leeds, UK
10 March 2021

Origins

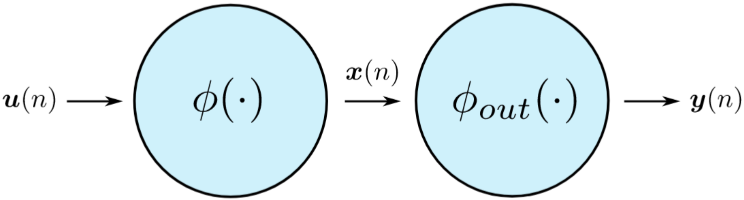
Machine-learning problem

Wish to learn the functional relationship between

→ Given input: $\boldsymbol{u}(n) \in \mathbb{R}^{N_u}$

→ Desired output: $\boldsymbol{y}_{target}(n) \in \mathbb{R}^{N_y}$

Dataset: $\{(\boldsymbol{u}(n), \boldsymbol{y}_{target}(n))\}_{n=1}^{T}$.

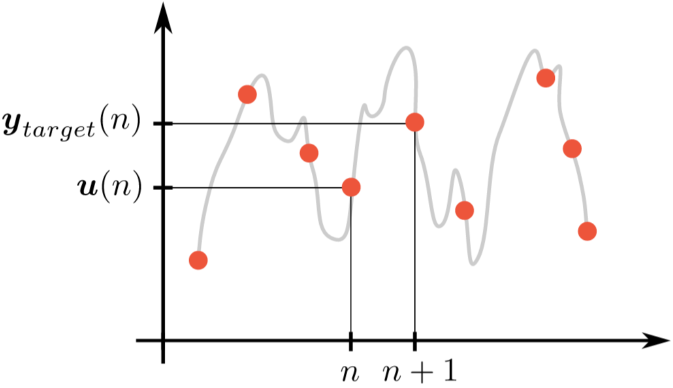


Time-series prediction

Suppose that n denotes the discrete time.

→ Feedforward Neural Networks: $\boldsymbol{x}(n) = \phi(\boldsymbol{u}(n), \boldsymbol{u}(n-1), \ldots)$

→ Recurrent Neural Networks: $\boldsymbol{x}(n) = \phi(\boldsymbol{x}(n-1), \boldsymbol{u}(n))$

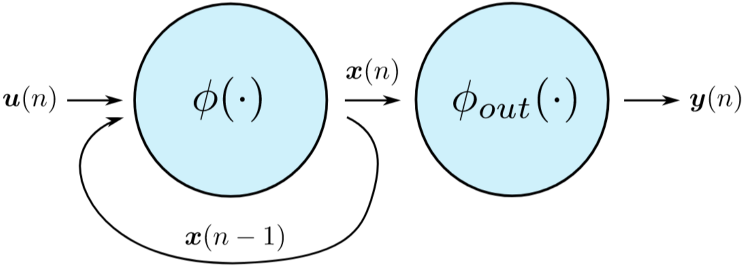


Time-series prediction

Suppose that n denotes the discrete time.

→ Feedforward Neural Networks: $\boldsymbol{x}(n) = \phi(\boldsymbol{u}(n), \boldsymbol{u}(n-1), \ldots)$

→ Recurrent Neural Networks: $\boldsymbol{x}(n) = \phi(\boldsymbol{x}(n-1), \boldsymbol{u}(n))$

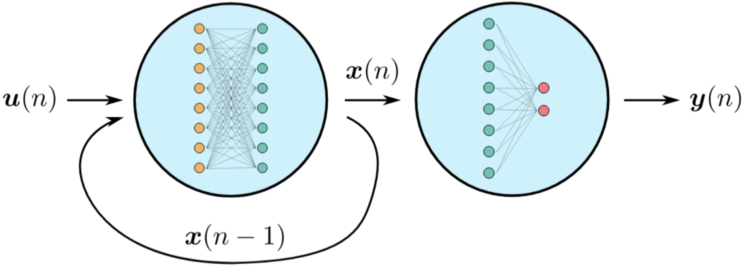


Recurrent neural networks (RNN)

Simple form of RNN:

$$\boldsymbol{x}(n) = f(\boldsymbol{W}_{in}\boldsymbol{u}(n) + \boldsymbol{W}\boldsymbol{x}(n-1)),$$
$$\boldsymbol{y}(n) = \boldsymbol{W}_{out}\boldsymbol{x}(n).$$

where $f(\cdot)$ is usually $\tanh(\cdot)$, $\boldsymbol{W}_{in} \in \mathbb{R}^{N_x \times N_u}$, $\boldsymbol{W} \in \mathbb{R}^{N_x \times N_x}$, $\boldsymbol{W}_{out} \in \mathbb{R}^{N_y \times N_x}$.
RNNs can be viewed as universal approximators of dynamical systems[1]



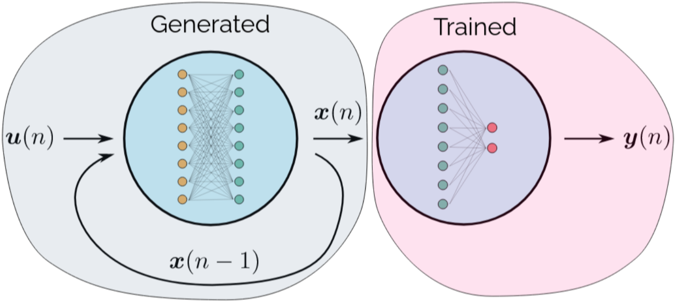
[1]Funahashi and Nakamura, *Neural Networks* **6**, 801–806 (1993)

Reservoir computing

Reservoir computing denotes the approach where the recurrent part of RNN is generated/trained separately from the recurrence-free readout.

Reservoir-computing methods:

- → Echo State Networks
- → Liquid State Machines
- → Backpropagation-Decorrelation training
- → Temporal Recurrent Networks

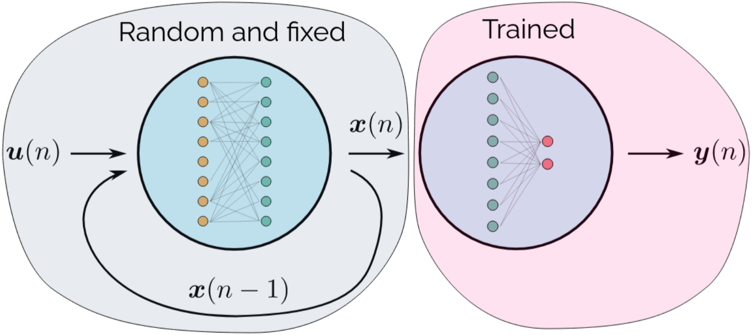


Echo State Networks

Echo State Network

$$\boldsymbol{x}(n) = f(\boldsymbol{W}_{in}\boldsymbol{u}(n) + \boldsymbol{W}\boldsymbol{x}(n-1)),$$
$$\boldsymbol{y}(n) = \boldsymbol{W}_{out}\boldsymbol{x}(n).$$

where $\boldsymbol{W}_{in}$ and $\boldsymbol{W}$ are random sparse matrices and $\boldsymbol{W}_{out}$ is to be trained.

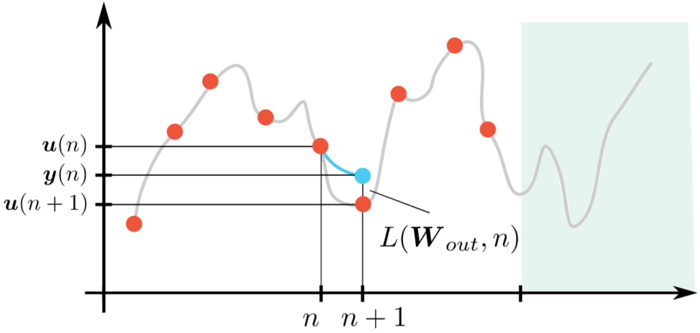


Training

Minimisation of the residual sum of squares (RSS):

$$\min_{\boldsymbol{W}_{out}} \sum_{n=1}^{T} ||\boldsymbol{y}(n) - \boldsymbol{y}_{target}(n)||_2^2 = \min_{\boldsymbol{W}_{out}} \sum_{n=1}^{T} ||\boldsymbol{W}_{out}\boldsymbol{x}(n) - \boldsymbol{u}(n+1)||_2^2,$$

where $\boldsymbol{x}(n) = f(\boldsymbol{W}_{in}\boldsymbol{u}(n) + \boldsymbol{W}\boldsymbol{x}(n-1))$.



Training

Solution via the normal equation:

$$\boldsymbol{W}_{out}^T = \left(\boldsymbol{X}^T \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{Y},$$

where matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ are given by

$$\boldsymbol{X} = \begin{bmatrix} \text{———} & \boldsymbol{x}(1) & \text{———} \\ \text{———} & \boldsymbol{x}(2) & \text{———} \\ & \cdots & \\ \text{———} & \boldsymbol{x}(T) & \text{———} \end{bmatrix}, \quad \boldsymbol{Y} = \begin{bmatrix} \text{———} & \boldsymbol{u}(1) & \text{———} \\ \text{———} & \boldsymbol{u}(2) & \text{———} \\ & \cdots & \\ \text{———} & \boldsymbol{u}(T) & \text{———} \end{bmatrix}$$

Here internal states $\boldsymbol{x}(n)$ are obtained via solving the explicit recurrent equation:

$$\boldsymbol{x}(n) = f(\boldsymbol{W}_{in}\boldsymbol{u}(n) + \boldsymbol{W}\boldsymbol{x}(n-1)),$$

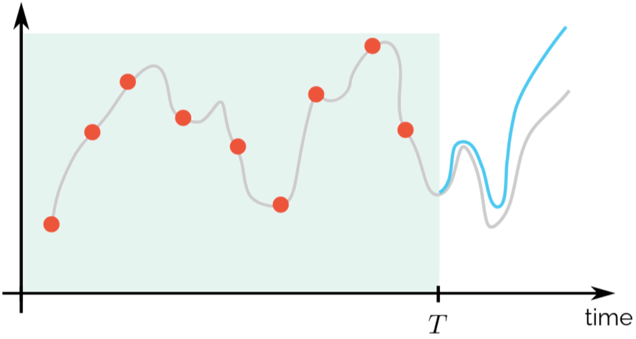
provided an initial condition $\boldsymbol{x}(0) = \boldsymbol{0}$.

Prediction

Generative mode:

$$\boldsymbol{x}(n) = f(\boldsymbol{W}_{in}\boldsymbol{y}(n-1) + \boldsymbol{W}\boldsymbol{x}(n-1)),$$
$$\boldsymbol{y}(n) = \boldsymbol{W}_{out}\boldsymbol{x}(n).$$

We still need to specify the initial condition $\boldsymbol{x}(0)$.

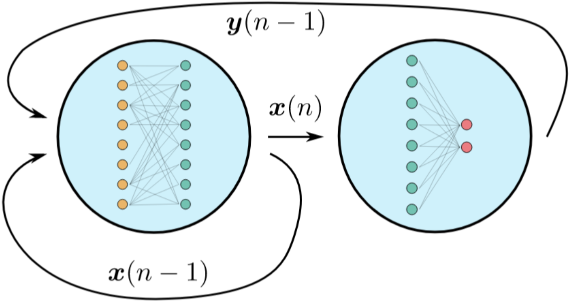


Prediction

Generative mode:

$$\boldsymbol{x}(n) = f(\boldsymbol{W}_{in}\boldsymbol{y}(n-1) + \boldsymbol{W}\boldsymbol{x}(n-1)),$$
$$\boldsymbol{y}(n) = \boldsymbol{W}_{out}\boldsymbol{x}(n).$$

We still need to specify the initial condition $\boldsymbol{x}(0)$.

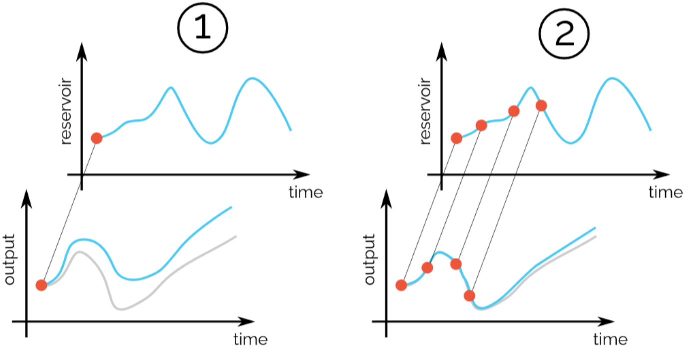


Prediction

Generative mode:

$$\boldsymbol{x}(n) = f(\boldsymbol{W}_{in}\boldsymbol{y}(n-1) + \boldsymbol{W}\boldsymbol{x}(n-1)),$$
$$\boldsymbol{y}(n) = \boldsymbol{W}_{out}\boldsymbol{x}(n).$$

We still need to specify the initial condition $\boldsymbol{x}(0)$.



Improvements

Echo State Property

This condition is guaranteed when $\alpha = \rho(\boldsymbol{W}) < 1$, so once $\boldsymbol{W}$ is randomly generated, it needs to be re-scaled:

$$\boldsymbol{W} := \frac{1}{\alpha}\boldsymbol{W}.$$

Tikhonov regularisation

Helps improve stability and overfitting.

$$\min_{\boldsymbol{W}_{out}} \sum_{n=1}^{T} ||\boldsymbol{W}_{out}\boldsymbol{x}(n) - \boldsymbol{u}(n+1)||_2^2 + \beta||\boldsymbol{W}_{out}||^2$$
$$\implies \quad \boldsymbol{W}_{out}^T = \left(\boldsymbol{X}^T\boldsymbol{X} + \beta\boldsymbol{I}\right)^{-1}\boldsymbol{X}^T\boldsymbol{Y}.$$

Noise immunisation

Helps improve stability and overfitting.

$$\boldsymbol{x}(n) = f(\boldsymbol{W}_{in}\boldsymbol{u}(n) + \boldsymbol{W}\boldsymbol{x}(n-1)) + \xi Z,$$

where $Z \sim \text{uniform}(-1, 1)$ and ξ is the noise strength.

Leaky integrator neurons

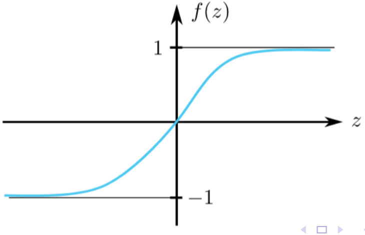
Echo State Networks can be viewed as Euler discretisation of the leaky-integrator-type ODE:

$$\frac{d\boldsymbol{x}}{dt} = -\boldsymbol{x} + f(\boldsymbol{W}_{in}\boldsymbol{u} + \boldsymbol{W}\boldsymbol{x})$$
$$\implies \boldsymbol{x}(n) = (1 - \triangle t)\boldsymbol{x}(n-1) + \triangle t f(\boldsymbol{W}_{in}\boldsymbol{u}(n) + \boldsymbol{W}\boldsymbol{x}(n-1)),$$

where $\triangle t \in [0; 1]$.

Input scaling

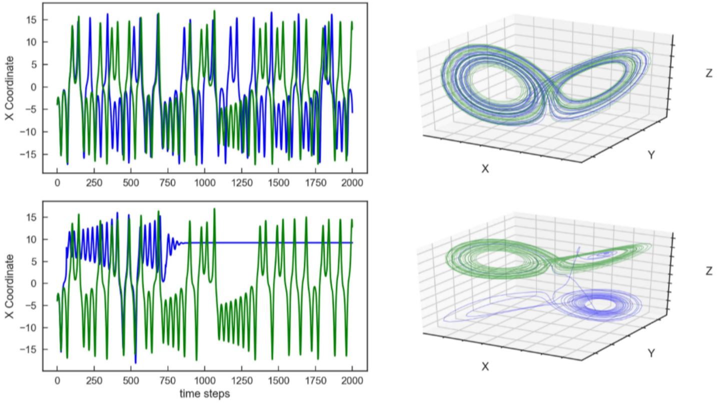
Scaling of the input weights $\boldsymbol{W}_{in}$ and shifting the input $\boldsymbol{u}(n)$ helps control the "amount of non-linearity".



Applications

Lorenz 63 model

Long-term prediction | Lyapunov exponents | Correlation dimension



Haluszczynski and Räth, *Chaos* **29**, 103143 (2019)

Lorenz 63 model

Long-term prediction | **Lyapunov exponents** | Correlation dimension

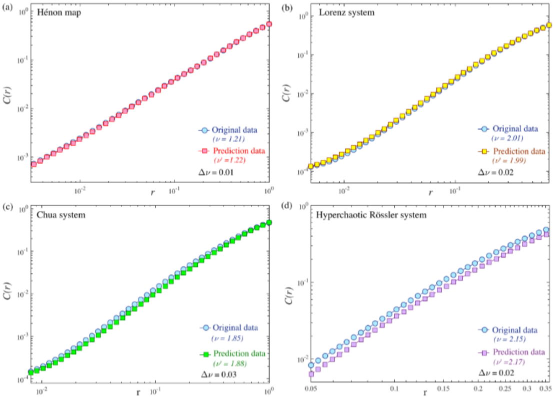
TABLE II. Three largest Lyapunov exponents $\Lambda_1 \geq \Lambda_2 \geq \Lambda_3$ for the Lorenz system [Eq. (5)], and for the reservoir set up in the configuration of Fig. 1(b) for R1 and R2. Since the reservoir system that we employ is a discrete time system, while the Lorenz system is a continuous system, for the purpose of comparison, Λ_1, Λ_2, and Λ_3 are taken to be per unit time; that is, their reservoir values (columns 2 and 3) are equal to the reservoir Lyapunov exponents calculated on a per iterate basis divided by Δt.

	Actual Lorenz system	R1 system	R2 system
Λ_1	0.91	0.90	0.01
Λ_2	0.00	0.00	–0.1
Λ_3	–14.6	–10.5	–9.9

Pathak et al., *Chaos* **27**, 121102 (2017)

Lorenz 63 model

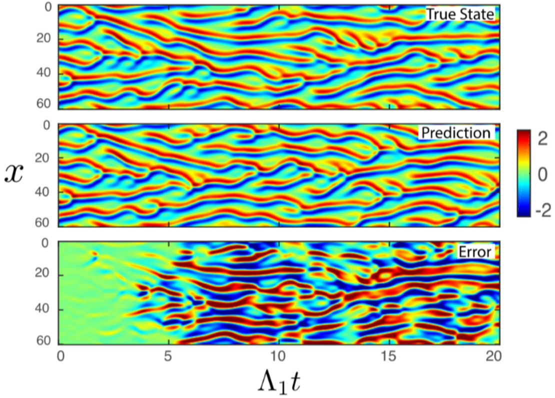
Long-term prediction | Lyapunov exponents | **Correlation dimension**



Chen et al., *Phys. Rev. E* **102**, 033314 (2020)

Kuramoto–Sivashinsky equation

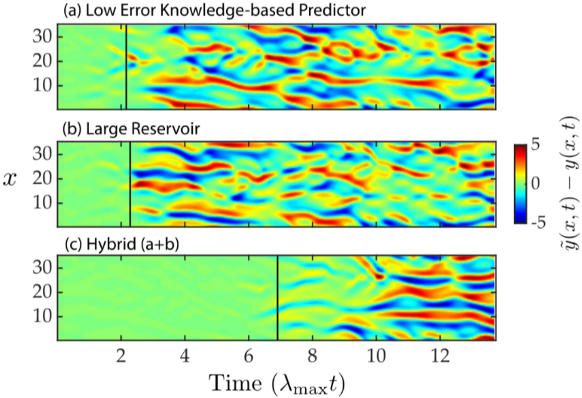
Short-term prediction | Failure | Improved prediction | Lyapunov exponents



Pathak et al., *Chaos* **27**, 121102 (2017)

Kuramoto–Sivashinsky equation

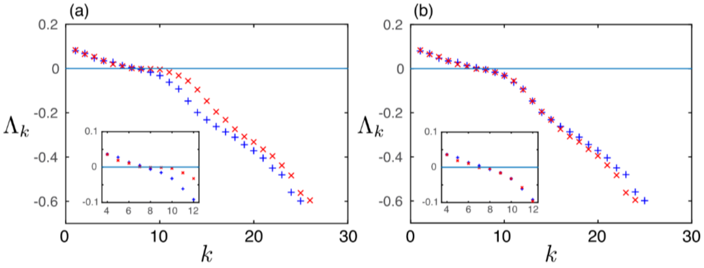
Short-term prediction | Failure | **Improved prediction** | Lyapunov exponents



Pathak et al., *Chaos* **28**, 041101 (2018)

Kuramoto–Sivashinsky equation

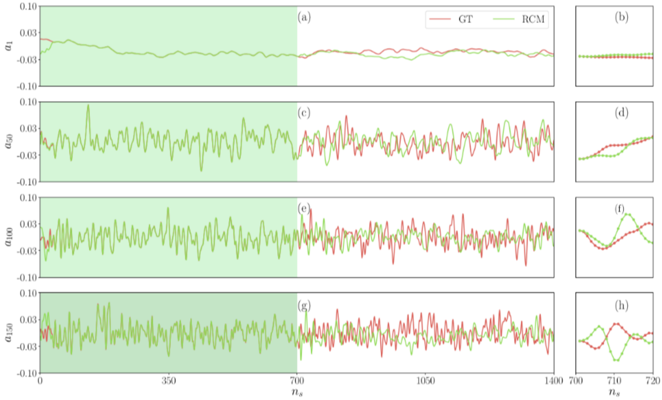
Short-term prediction | Failure | Improved prediction | **Lyapunov exponents**



Pathak et al., *Chaos* **27**, 121102 (2017)

Rayleigh–Bénard convection

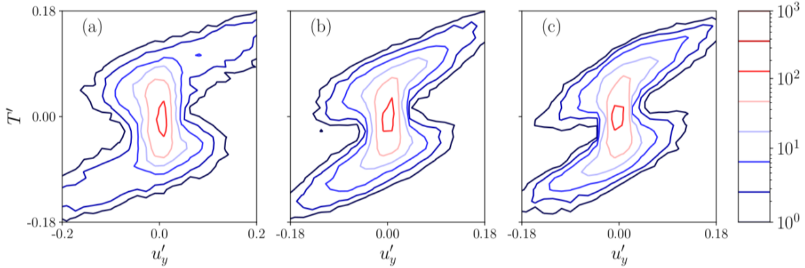
Long-term prediction | Snapshots | Statistics



Pandey et al., *Phys. Rev. Fluids* **5**, 113506 (2020)

Rayleigh–Bénard convection

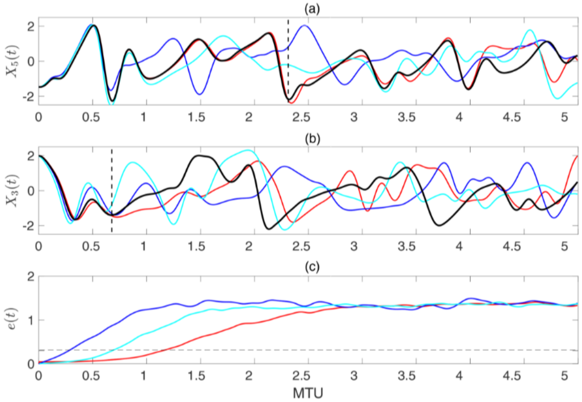
Long-term prediction | Snapshots | **Statistics**



Pandey et al., *Phys. Rev. Fluids* **5**, 113506 (2020)

Comparative studies: Lorenz 96

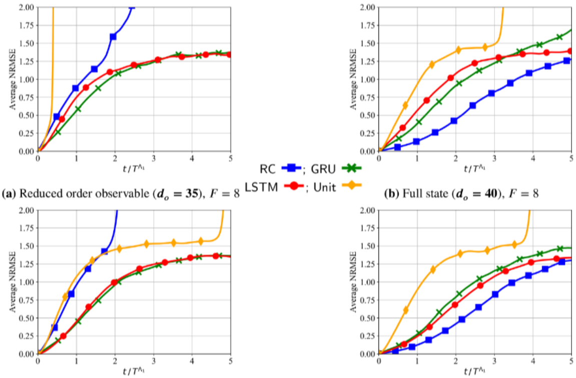
RC vs. ANN vs. LSTM | RC vs. GRU vs. LSTM vs. Unit



Chattopadhyay et al., *Nonlin. Processes Geophys.* **27**, 373–389 (2020)

Comparative studies: Lorenz 96

RC vs. ANN vs. LSTM | **RC vs. GRU vs. LSTM vs. Unit**



(a) Reduced order observable ($\boldsymbol{d_o} = \mathbf{35}$), $F = 8$

(b) Full state ($\boldsymbol{d_o} = \mathbf{40}$), $F = 8$

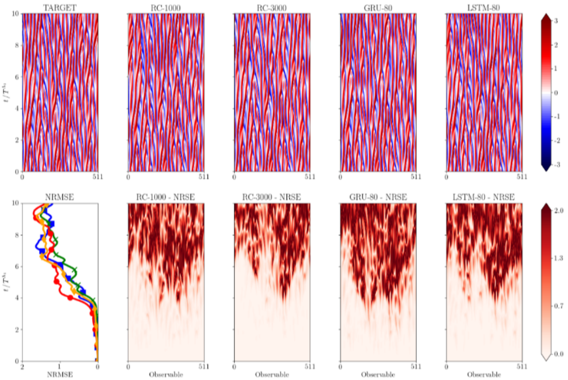
(c) Reduced order observable ($\boldsymbol{d_o} = \mathbf{35}$), $F = 10$

(d) Full state ($\boldsymbol{d_o} = \mathbf{40}$), $F = 10$

Vlachas et al., *arXiv:1910.05266* (2019)

Comparative studies: K.–S. equation

RMS error | Performance

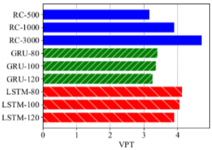


RC-1000 ; RC-12000 ; GRU-500 ; LSTM-100

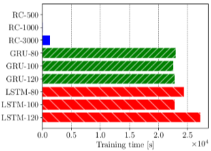
Vlachas et al., *arXiv:1910.05266* (2019)

Comparative studies: K.–S. equation

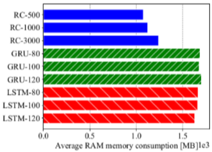
RMS error | **Performance**



(a) Valid prediction time in the test dataset



(b) Training time



(c) Average RAM memory requirement

Vlachas et al., *arXiv:1910.05266* (2019)

How can we use reservoir computing?

Building a dynamical system from observations only	Acceleration of numerical simulations
Super-parameterisation	Reduced-order modelling