Predicting nonlinear time series with reservoir computers

This document provides an example of how a reservoir computer can be implemented to predict a nonlinear time series. For a reference on the architecture and updating rules of reservoir computers, see pages 183 - 186 in [1]. The time series we wish to predict is given by the Lorenz equations

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{dt}}x_1 &= -\sigma x_1 + \sigma x_2, \\ \frac{\mathrm{d}}{\mathrm{dt}}x_2 &= -x_1 x_3 + r x_1 - x_2, \\ \frac{\mathrm{d}}{\mathrm{dt}}x_3 &= x_1 x_2 - b x_3, \end{aligned} \tag{1}$$

which is a model for atmospheric convection. Using the parameter values $\sigma = 10$, $r = 28$, and $b = 8/3$, the system displays chaotic dynamics, which is characterised by (among other things) high sensitivity to initial conditions. The sensitivity can be quantified by considering the evolution of a small perturbation $\delta(t) \equiv |\delta(t)|$ from an initial state in phase space. One can show [2] that for large times t, the perturbation evolves as

$$\delta(t) \sim e^{\lambda_1 t}\delta(0). \tag{2}$$

The quantity λ_1 is the maximal Lyapunov exponent of the system. For chaotic systems, $\lambda_1 > 0$, and hence nearby trajectories diverge exponentially fast. The inverse λ_1^{-1} is called the Lyapunov time and gives an approximation of how long it takes before two nearby trajectories diverge significantly. For the Lorenz system with the provided parameter values, $\lambda_1 \approx 0.906$ [3].

Reservoir computers have been successfully used to predicting nonlinear time series. In fig. 1, the dynamics of $x_2(t)$ is shown together with the prediction of a reservoir computer. As can be seen, the prediction remains accurate for more than twice the Lyapunov time. To obtain this result, eq. (1) is integrated numerically over $t = [0, 50]$ with time steps of 0.02, and initial conditions are set such that the trajectory starts on the chaotic attractor to avoid initial transients. The first 80% of the time series is then used for training, and the remaining 20% for testing. The input weights $w_{ij}^{(\mathrm{in})}$ of the reservoir computer are initialised such that each entry is drawn from a uniform distribution ranging between $[-0.1, 0.1]$. The weights connecting the reservoir neurons w_{ij} are drawn from a uniform distribution ranging between $[-1, 1]$. The reservoir matrix is then rescaled so that its maximal singular value is equal to unity[1]. Finally, the weights connecting the reservoir to the output $w_{ij}^{(\mathrm{out})}$ are found through ridge regression with ridge parameter $k = 0.1$, such that the difference between the output of the network $O_i(t)$ and the target values $y_i(t)$ are minimised. *tanh* is used as activation function for the reservoir neurons.

Training procedure

The updating rules for the employed reservoir computer is

$$r_i(t+1) = g\left(\sum_j w_{ij} r_j(t) + \sum_{k=1}^{N} w_{ik}^{(in)} x_k(t)\right) \tag{3}$$

$$O_i(t+1) = \sum_{j=1}^{M} w_{ij}^{(out)} r_j(t+1) \tag{4}$$

[1]Denoting by $\mathbb{W}$ the reservoir weight matrix, rescaling the matrix so that its maximal singular value is α is done by dividing $\mathbb{W}$ by its maximal singular value $\sigma_{\max}$ and multiplying it by α. That is,

$$\tilde{\mathbb{W}} = \alpha \frac{\mathbb{W}}{\sigma_{\max}},$$

where $\tilde{\mathbb{W}}$ is the rescaled weight matrix.

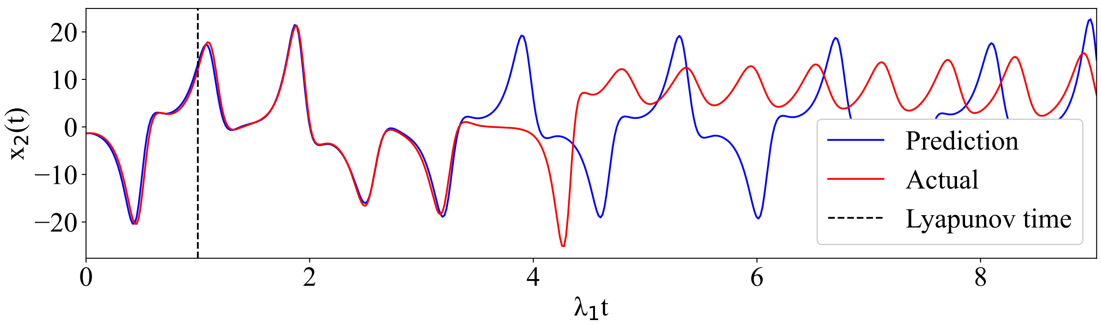

Figure 1: Time series prediction of x_2 component of the Lorenz equations using a reservoir computer.

where $r_i(t)$ is the state of the i:th reservoir neuron at time t ($r_i(0) = 0$), $x_i(t)$ is the i:th input to the network at time t, w_{ij} are the weights connecting the reservoir neurons, $w_{ij}^{(in)}$ are the weights connecting the input to the reservoir, $w_{ij}^{(out)}$ are the weights connecting the reservoir to the output, $O_i(t)$ is the output at time t, and $g(\cdot)$ is the activation function. To train a reservoir network, a time series is fed into the network and the states $r_i(0)$, $r_i(1), \ldots$ are recorded. Then, weights $w_{ij}^{(out)}$ are optimised (using e.g. least squares, gradient descent) such that the differences between the outputs $O_i(t)$ and target values $y_i(t)$ are minimised. In our case, this means that the outputs give the coordinates (x_1, x_2, x_3) for the next time step in the time series. Once the network has been trained, the input $x_i(t)$ is replaced by $O_i(t)$ so that the network loops on itself. The output is then recorded for each time step.

References

[1] B. Mehlig, *Artificial Neural Networks*, 2021, https://arxiv.org/pdf/1901.05639.pdf

[2] K. Gustafsson, *Chaos and Lyapunov exponents*, 2017, http://fy.chalmers.se/ f99krgu/dynsys/DynSysLecture10.pdf

[3] D. Viswanath, *Lyapunov exponents from random Fibonacci sequences to the Lorenz equations*, 1998, Cornell University