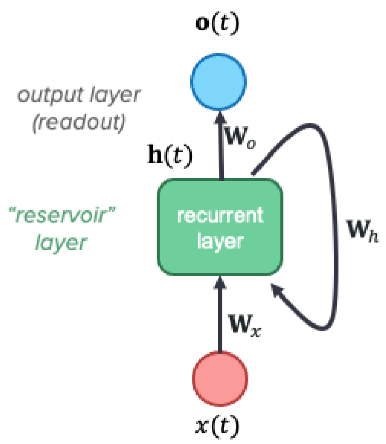
Additional Material (Lab3-2):

Echo State Networks



- Input $\mathbf{x}(t)$: column vector of size N_x
- Reservoir state $\mathbf{h}(t)$: column vector of size N_h
- Output $\mathbf{o}(t)$: column vector of size N_o

When both input and output are 1-dimensional (e.g., as in the assignments), we use $N_x = N_Y = 1$.

Reservoir

- It is a recurrent, non-linear layer with (typically) sparse connectivity among units.
- It computes the state transition function: how the state at time step t is obtained from the input at time step t and from the state at time step t-1:

$$\mathbf{h}(t) = \tanh\left(\mathbf{W_h}\mathbf{h}(t-1) + \mathbf{W_x}\mathbf{x}(t) + \mathbf{b}\right)$$

  - Note that the bias vector can be included in the input-weight matrix $\mathbf{W_x}$ for simplicity, in this case the input at each time-step should be augmented by a fixed input bias term equal to 1. In this case, the reservoir equation reads as follows:

$$\mathbf{h}(t) = \tanh\left(\mathbf{W_h}\mathbf{h}(t-1) + \mathbf{W_x}[\mathbf{x}(t); 1]\right),$$

    where [;] denotes vector concatenation
  - Input bias for the reservoir: concatenate the input at each time step with a constant input bias equal to 1;
  - Use a null state as initial state of the ESN: $\mathbf{h}(0) = \mathbf{0}$

Readout

It computes the output function: how the output of the network at time step t is obtained from the state at time step t. In the simplest form, it is implemented as a feed-forward, linear dense layer, as follows:

$$\mathbf{o}(t) = \mathbf{W}_y[\mathbf{o}(t); 1]$$

where the readout bias vector is already included in the output weight matrix $\mathbf{W}_y$.

Reservoir Initialization

Initialize the reservoir according to the necessary condition for the Echo State Property (ESP), i.e., $\rho(\mathbf{W}_h) < 1$. Recall that $\rho(\cdot)$ denotes the *spectral radius* (i.e., the maximum length of an eigenvalue) of its argument, namely $\rho(\mathbf{W}_h) = \max\left(abs\left(eig(\mathbf{W}_h)\right)\right)$.

To properly initialize the reservoir weights, proceed as follows:

1. Initialization of the input weight matrix:
   Choose the values of $\mathbf{W}_x$ from a random distribution scaled by a hyper-parameter ω_x. A typical choice is a uniform distribution on $(-\omega_x, \omega_x)$.
2. Initialization of the bias vector:
   Choose the values of $\mathbf{b}$ from a random distribution scaled by a hyper-parameter ω_b. A typical choice is a uniform distribution on $(-\omega_b, \omega_b)$.
3. Initialization of the recurrent weight matrix:
   a. Initialize randomly the values of $\mathbf{W}_h$ from a random distribution (e.g., from a uniform distribution in (-1, 1)).
   b. Re-scale $\mathbf{W}_h$ to a unitary spectral radius, i.e., $\mathbf{W}_h \leftarrow \frac{\mathbf{W}_h}{\rho(\mathbf{W}_h)}$
   c. Finally, re-scale $\mathbf{W}_h$ to the desired spectral radius ρ, i.e., $\mathbf{W}_h \leftarrow \rho\, \mathbf{W}_h$

Readout Training

Only the readout needs to be trained.

1. Discard an initial transient (run the network for some steps before starting collecting the states)

2. Collect all reservoir states and target values for each time step into matrices (after the initial transient)

$$\mathbf{H} = [\mathbf{h}(1), \dots, \mathbf{h}(T)] \qquad \mathbf{Y} = [\mathbf{y}(1), \dots, \mathbf{y}(T)]$$

where $\mathbf{y}(t)$ here denotes the target output at time-step t.

3. After having collected all the states, train the linear readout
   - Pseudo-inverse
     $\mathbf{W}_o = \mathbf{Y}\,\mathbf{H}^+$
   - Ridge regression (λ_r is a regularization coefficient)
     $\mathbf{W}_o = \mathbf{Y}\,\mathbf{H}^T(\mathbf{H}\,\mathbf{H}^T + \lambda_r\,\mathbf{I})^{-1}$

   note: the readout should not be trained after each time step, but only once after the collection of all the states

Implementation notes

- In NumPy, you can use the linear algebra functions in the numpy.linalg library (https://numpy.org/doc/stable/reference/routines.linalg.html) for computing the spectral radius and for training the readout (e.g., for inverse, pseudo inverse, etc)
- In Matlab, you can use the functions eig(.), pinv(.), and inv(.)

Reservoir Guesses

For every reservoir hyper-parametrization the performance (e.g. accuracy for classification task, MSE for regression task) should be averaged over a number of reservoir guesses (different random instantiations of networks with the same values of the hyper-parameters).

Model Selection!

To properly setup the network you need to use a proper model selection (e.g, by grid or random search), considering suitable values of the hyper-parameters. Relevant hyper-parameters include: number of reservoir units, spectral radius, input scaling, bias scaling, readout regularization