Let us suppose we have an input signal $\mathbf{u}(t) \in R^M$ where $t = 1, \ldots, T$ is the discrete time and T the number of data points in the training set. In order to project this input onto the reservoir we will need an input to reservoir coupler, identified by the matrix $\mathbf{W}_{in} \in R^{N \times M}$. Usually this matrix is built in the same way the reservior is, and at the same time. For the implementation used in ReservoirComputing.jl we have followed the same construction proposed in [9] where the i-th of the M input signals is connected to N/M reservoir nodes with connection weights in the i-th column of $\mathbf{W}_{in}$. The non-zero elements are chosen randomly from a uniform distribution and then scaled in the range $[-\sigma, \sigma]$.

The reservoir is constitued by N neurons connected in a Erdős–Rényi graph configuration and it is represented by an adjacency matrix $\mathbf{W}$ of size $N \times N$ with values drawn from a uniform random distribution on the interval $[-1, 1]$ [5]. This is the most important aspect of the ESN, so in order to build one in an efficient manner we must first understand all of its components.

The size N is of course the single most important one: the more challenging the task, the bigger the size should be. Of course a bigger matrix will mean more computational time so the advice of Lukoševičius is to start small and then scale.

The sparsity of the reservoir. In most papers we see that each reservoir node is connected to a small number of other nodes, ranging from 5 to 12. The sparseness, beside theoretical implications, is also useful to speed up computations.

Spectral radius. After the generation of a random sparse reservoir matrix, its spectral radius $\rho(\mathbf{W})$ is computed and $\mathbf{W}$ is divided by it. This allows us to obtain a matrix with a unit spectral radius, that can be scaled to a more suited value. Altough there are exceptions (when the inputs $\mathbf{u}(t)$ are non-zero for example), a spectral radius smaller than unity $\rho(\mathbf{W}) < 1$ ensures the echo state property. More generally this parameter should be selected to maximize the performance, keeping the unitary value as a useful reference point.

After the construction of the input layer and the reservoir we can focus on harvesting the states. The update equations of the ESN are:

$$\mathbf{x}(t + \Delta t) = (1 - \alpha)\mathbf{x}(t) + \alpha f(\mathbf{W}\mathbf{x}(t) + \mathbf{W}_{in}\mathbf{u}(t))$$

$$\mathbf{v}(t + \Delta t) = g(\mathbf{W}_{out}\mathbf{x}(t))$$

where

$\mathbf{v}(t) \in R^M$ is the predicted output

$\mathbf{x}(t) \in R^N$ is the state vector

$\mathbf{W}_{out} \in R^{M \times N}$ is the output layer

f and g are two activation functions, most commonly the hyperbolic tangent and identity respectively

α is the leaking rate

The computation of $\mathbf{W}_{out}$ can be expressed in terms of solving a system of linear equations

$$\mathbf{Y}^{target} = \mathbf{W}_{out}\mathbf{X}$$

where $\mathbf{X}$ is the states matrix, built using the single states vector $\mathbf{x}(t)$ as column for every $t = 1, \ldots, T$, and $\mathbf{Y}^{target}$ is built in the same way only using $\mathbf{y}^{target}(t)$. The chosen solution for this problem is usually the Tikhonov regularization, also called ridge regression which has the following close form:

$$\mathbf{W}_{out} = \mathbf{Y}^{target}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \beta\mathbf{I})^{-1}$$

where $\mathbf{I}$ is the identity matrix and β is a regularization coefficient.

After the training of the ESN, the prediction phase uses the same update equations showed above, but the input $\mathbf{u}(t)$ is represented by the computed output $\mathbf{v}(t - \Delta t)$ of the preceding step.

In short this is the core theory behind the ESNs. In order to visualize how they work let's look at an example.