Echo State Networks

Hector Morlet

22247737

October 4, 2024

Introduction

This paper introduces Echo State Networks and broadly covers Reservoir Computing. We begin by outlining the limitations of traditional Recurrent Neural Networks (RNNs), followed by an explanation of the defining feature of Reservoir Computers: the Echo State Property. We will establish the formalisation of a reservoir computer, and explain the structure of an Echo State Network (ESN). We will discuss how we can find adequate values for the hyper-parameters of an ESN through heuristic methods and how we can 'train' our ESN using an input sequence to produce a predicted sequence.

Reservoir Computers: Background and Motivation

A recurrent neural network is designed to perform sequence-to-sequence prediction [2]. RNNs are often comprised of a single 'hidden layer' but can have more. The 'state' value of each hidden layer is updated through the equation:

$$\mathbf{s}(t+1) = \sigma(\mathbf{C_{in}}\mathbf{x}(t) + \mathbf{C_{rec}}\mathbf{s}(t) + \mathbf{b})$$

Due to its recurrence, an RNN can approximate any sequence with arbitrary accuracy, given enough hidden layers. Fitting the parameters of an RNN requires accounting

for errors propagating through time, often using the Backpropagation Through Time algorithm. However, the successive multiplication of extreme numbers (large and small) causes the gradients calculated through this algorithm to either 'vanish' or 'explode'[2]. While some RNNs such as Long Short Term Memory (LSTM) networks encode a memory buffer to mitigate this problem, reservoir computers offer another alternative.

Reservoir computers are a variation of RNNs for processing temporal data which consist of feeding input signals into a non-linear dynamical system named the 'reservoir'. The parameters of the reservoir are fixed, allowing us to only fit a readout vector and avoid fitting multiple layers of hidden node weights. Thus the training algorithm can simply become regularised least squares regression, and reservoir computers can take advantage of a large number and depth of parameters without suffering from the issues of vanishing gradients. While ESNs have the added benefit of requiring far less training computation than traditional RNNs, they require some care to be taken in establishing the properties of the reservoir.

The Echo State Property

The defining characteristic of reservoir computers is the *echo state property*. This property ensures that, given a sufficiently long input sequence, the current state of the reservoir is determined uniquely by the intervening input and is independent of its initial state[1]. In other words, the further past inputs are from the current state, the less they impact the current state. More formally, a network possesses echo states if its states are uniquely determined by the input sequence $\mathbf{x}(t)$ for some sufficiently large t, i.e., $\mathbf{s}(t) = \mathbf{s}'(t)$ for any initial states $\mathbf{s}(0)$ and $\mathbf{s}'(0)$, where $\mathbf{s}(t)$ represents the reservoir state at time t.

For a network to exhibit the echo state property, it must satisfy three key conditions:

- **Uniform State Contraction:** The network must be contractive, meaning that differences in initial states decay over time.
- **State Forgetting:** The network must 'forget' its initial state after a finite

amount of time, ensuring that only recent inputs affect the current state.

- **Fading Memory:** The influence of past inputs diminishes over time, making the network's state primarily a function of recent inputs. This can be seen in Figure 1a, where the accuracy of the remembered input compared to the actual input fades as the time delay grows.

Each state of the networks can be viewed as a function of a finite sequence of prior inputs:

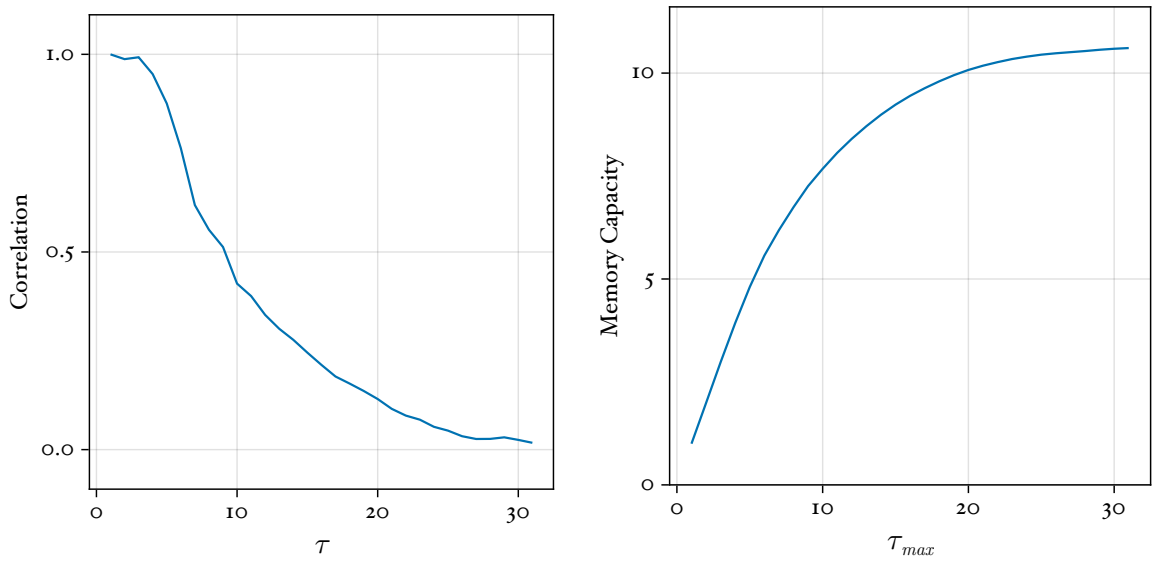
$$\mathbf{s}(t) = \mathcal{F}(\mathbf{s}(t-1), \mathbf{s}(t-2), ...\mathbf{s}(0)) \tag{1}$$

$$\approx \mathcal{F}(\mathbf{s}(t-1), \mathbf{s}(t-2), ...\mathbf{s}(t-\tau_{MC})) \tag{2}$$

Where the number of time steps τ_{MC} corresponds to some memory capacity of the network. The states that are beyond τ_{MC} steps in the past practically do not contribute to $\mathbf{s}(t)$. The memory capacity for a given time delay τ_{max} can be quantified by the following method :

$$MC(\tau_{max}) = \sum_{\tau=1}^{\tau_{max}} correlation\left(\{\mathbf{x}(n-\tau), \mathbf{y}_\tau(n)\}_{n=\tau}^{n=T}\right)$$

The variable $\mathbf{y}_\tau$ refers to the output of a reservoir computer trained to reproduce the input from τ time steps in the past. We are calculating the correlation of these 'remembered' values with the actual values over the whole sequence, where T is the final time step of the sequence. Figure 1a shows correlations measured for a reservoir computer over a range of τ values. To produce the memory capacity for a particular time delay τ_{max}, we then sum the correlations calculated for time steps 1 to τ_{max}. Figure 1b shows the relevant memory capacities for the measured correlations. The reader may notice that Figure 1b is a cumulative sum of Figure 1a. To measure the length of the reservoir computer's memory τ_{MC}, we should find the lowest value of τ_{max} for which the memory capacity has converged. This value will represent how far back a reservoir computer can recall patternless inputs. We can see an example of this convergence in



(a) Correlation over τ (b) Convergence of memory capacity $MC(\tau_{max})$

Figure 1: Measures of correlation and memory capacity over time delay τ for an example reservoir computer, an Echo State Network with 100 nodes and $\rho = 2$.

the memory capacity in Figure 1b. In this example, the memory capacity converges to around 11 after approximately $\tau_{max} = 27$, so we can say that $\tau_{MC} \approx 27$.

In most prediction tasks, the most recent inputs are more relevant than those received further back. This basic reasoning is the intuitive justification for why the echo state property is important to creating a useful model. With a large enough memory capacity, the echo state property ensures that the reservoir provides a rich, yet stable and decaying, representation of the input signal over time.

Defining a Reservoir Computer

Prediction using a reservoir computer can be split into two parts: firstly, the evolution of the reservoir states (also known as *activation states*) and secondly, reading out the activation states to produce an output. At each time step, a reservoir computer will update its state given the input at that time step, and an output may be calculated from this state. When the time steps of the reservoir computer are discrete, we can

abstractly define the state of the reservoir to be a function of the prior state and input using the evolution equation:

$$\mathbf{s}(t+1) = f_{RC}(\mathbf{s}(t), \mathbf{x}(t))$$

Here $\mathbf{s}(t)$ represents the state of the reservoir at time t, $\mathbf{x}(t)$ represents the input at time t and f_{RC} is the evolution operation of our reservoir.

The states $\mathbf{s}(t)$ over t can then be stored as a sequence and used to fit a readout matrix $\mathbf{C}_{out}$, perhaps using a regularised least squares regression. Unseen data can then be fed into the reservoir and we can use the readout vector to glean the desired prediction from the activation states using the linear output equation:

$$\mathbf{y}(t) = \mathbf{C}_{out}\mathbf{s}(t)$$

Echo State Networks

Echo State Networks (ESNs) are a form of RNN introduced by Herbert Jaeger in 2001 as a part of the broader family of reservoir computing methods [1]. Unlike traditional RNNs, ESNs avoid the computationally expensive process of training internal weights. Instead, the internal connections of the network (which form the 'reservoir') are chosen randomly and then fixed, while only the output weights are trained. This makes ESNs computationally efficient and - given the right choice of hyper-parameters - effective at processing time-dependent data.

In an ESN, the reservoir consists of a randomly generated recurrent neural network defined by the weights $\mathbf{W}_{in}$, $\mathbf{W}_{rec}$ and $\mathbf{W}_{bias}$ and activation function f_{act} that together govern the evolution operation. ESNs work in discrete time, so we can define the evolution function as:

$$\mathbf{s}(t+1) = f_{act}(\mathbf{W}_{in}\mathbf{x}(t) + \mathbf{W}_{rec}\mathbf{s}(t) + \mathbf{W}_{bias})$$

In Figure 2, the input weights $\mathbf{W}_{in}$ can be visualised as the connections between the

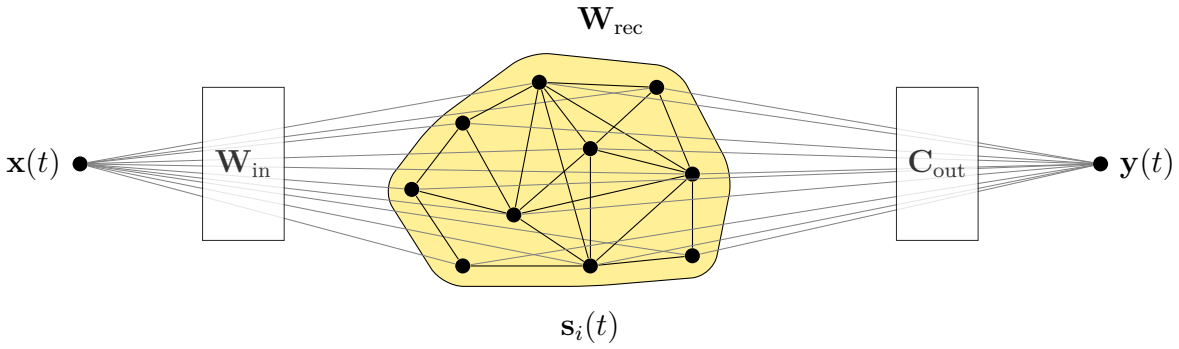


Figure 2: A diagram of an Echo State Network.

input, $\mathbf{x}(t)$, and the states of the reservoir (represented as the nodes within the yellow area). The adjacency matrix $\mathbf{W}_{rec}$ is visualised in Figure 2 as the connections between states in the reservoir (the yellow area). The activation states of the reservoir $\mathbf{s}(t)$ are visualised as the nodes in the reservoir, and the readout vector, $\mathbf{C}_{out}$ is visualised as connections between the states of the reservoir and the output, $\mathbf{y}(t)$. The bias matrix $\mathbf{W}_{bias}$ also acts on the activation states, but it is not shown in Figure 2.

Given a reservoir size of k and an input dimension of n, the weights and activation function of the reservoir can be described:

- $\mathbf{W}_{rec}$ is a $k \times k$ adjacency matrix that defines the step-wise effect of one reservoir state on another. It is typically generated as the adjacency matrix of an Erdős-Rényi random graph with a mean degree d, which is then scaled to have a maximum absolute Eigenvalue of ρ.

- $\mathbf{W}_{bias}$ is a bias vector of length k, typically generated by drawing from a uniform or normal distribution.

- $\mathbf{W}_{in}$ is an $n \times k$ input matrix that defines the step-wise effect of the input on the reservoir state. It is also typically generated by drawing from a uniform or normal distribution.

- f_{act} is an activation function for the states; tanh is a common choice of activation function.

The performance of the ESN is determined by the dynamics of the reservoir, which is affected by our selection of 'hyper-parameters' k, d and ρ. These hyper-parameters are typically chosen through heuristic methods. Here we will discuss each of these hyper-parameters and common heuristic methods for choosing their values as explained by Lukoševičius [3].

Reservoir Size (k)

Using a large number of nodes in a traditional RNN will result in significant computational load during training, but this is less so for ESNs because the vast majority of parameters are fixed. Performance will generally improve as we increase the number of nodes, thus we can set the number of nodes in the reservoir as high as our hardware setup and inference-time requirements reasonably allow. However ESNs are not immune to the nonlinear scaling of training and inference times. As can be seen in Table 1 as k increases, the training and inference times increase at a nonlinear rate. To summarise, a larger number of nodes will result in better performance and ESNs with k in the hundreds to thousands are common, but not without their own computation costs.

k	Mean training time (ms)	Mean inference time (ms)	τ_{MC}
10	1.825	0.498	4
20	3.801	0.639	11
40	5.576	0.958	14
80	13.197	1.821	17
160	95.047	5.317	22

Table 1: Memory capacities for various values of k and their training and inference times on a sequence of length 100,000.

Average Degree (d)

When the reservoir network is initiated using a method such as an Erdős-Rényi random graph, we can specify the average degree of the nodes. The average degree determines the sparsity of the connections within the reservoir. While it is important for d to be sufficiently small so that the reservoir is sparsely connected, the literature suggests that the exact setting of this hyper-parameter is not as important as the other hyper-parameters. A typical value for d is around 5% of the total network size k, ensuring that the network is sufficiently sparse while still capable of capturing the dynamics of the input.

Spectral Radius (ρ)

The spectral radius is the largest absolute eigenvalue of the reservoir's weight matrix W_{rec}. ρ controls the dynamics of the reservoir and therefore the memory capacity of the reservoir. If $\rho = 0$, the network is the zero matrix and so past states are not propagated through the network. In effect, the network forgets about past states and depends solely on the current input. When $0 < \rho < 1$, the past states are propagated through the network but the current input is still the dominant effect on the activation states. As ρ approaches 1, the reservoir begins to exhibit chaotic behavior. As we increase $\rho > 1$, the ESN approaches a tipping point (often referred to as the 'edge of chaos') beyond which the ESN memory dominates the input signal. With values $\rho >> 1$, the ESN becomes unstable and uninformative as the input signal has little effect on its dynamics. A commonly used heuristic is to set $\rho \approx 1$ to balance the memory and the stability of the ESN, however research suggests that setting ρ closer to the edge of chaos can improve performance of the reservoir. Figure 3 demonstrates the effect of ρ on the memory capacity of the ESN. As ρ is increased, memory capacity increases. Above $\rho = 1$, the spread of possible memory capacities increases considerably and the lower bound peaks at some value of ρ just above $\rho = 1$. We can consider this value to be the 'edge of chaos', beyond which the reservoir becomes increasingly unstable and so the memory capacity becomes increasingly unreliable.

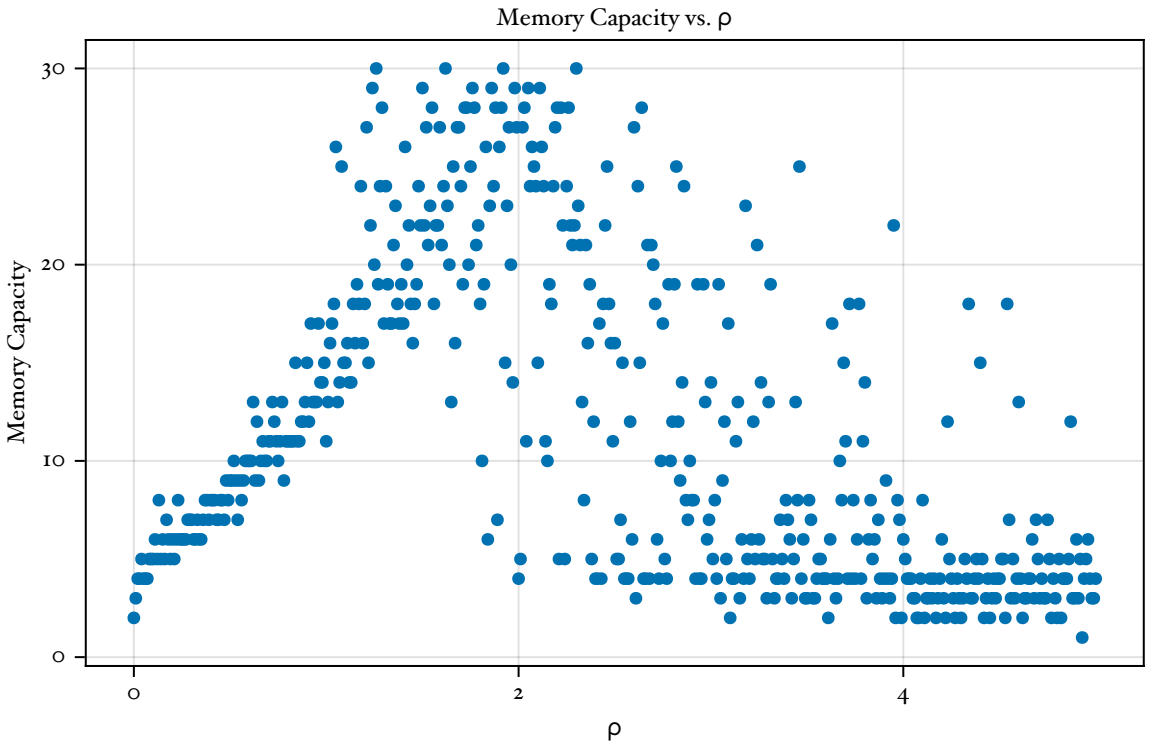


Figure 3: A plot of an ESN's memory capacity for various values of ρ.

Training the ESN

Since the internal reservoir weights remain fixed, the only part of the ESN that requires training is the output weight matrix $\mathbf{C}_{out}$. This is typically done using ridge regression, also known as Tikhonov regularisation. The goal of training is to minimise the following cost function:

$$\min_{\mathbf{C}_{out}} \sum_{t} \|\mathbf{y}(t) - \mathbf{C}_{out}\mathbf{s}(t)\|^2 + \lambda \|\mathbf{C}_{out}\|^2$$

where $\mathbf{s}(t)$ represents the reservoir states that were generated by feeding the training sequence into the reservoir and $\mathbf{y}(t)$ is the desired output at each time step t. The regularisation parameter λ controls the degree of penalty applied to the weights.

Conclusion

Echo State Networks have found applications in time series prediction, system modeling, and classification, but their main advantage is to handle complex, nonlinear dynamical systems with minimal computational cost. In this short paper, we have explained the background, properties and implementation of ESNs. We have discussed practical heuristics for establishing an ESN, and how hyper-parameter selection can affect the performance of the model. In my dissertation, I plan to demonstrate how novel changes to the architecture of ESNs will change their performance for tasks involving complex time series prediction.

References

[1] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks - with an erratum note. Technical Report 148, German National Research Center for Information Technology GMD, Bonn, Germany, 2001.

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

[3] Mantas Lukoševičius. A practical guide to applying echo state networks. In *Neural Networks: Tricks of the Trade: Second Edition*, pages 659–686. Springer, 2012.