Thesis notes: Deep RON experimental analysis and comparison

Vincenzo Gargano

February 4, 2025

Contents

1 Introduction to Deep Reservoir Architectures **2**

2 Stacked Recurrent Oscillator Newtork **2**
2.1 RON equation . . . 2
2.2 Deep RON . . . 3
2.3 DeepESN . . . 3

3 Preliminary experiments **4**
3.1 Memory Capacity . . . 4
3.1.1 RON and Deep RON memory capacity . . . 4
3.1.2 ESN and DeepESN memory capacity . . . 4
3.2 Distance between States . . . 4

Notes

These notes shall serve as aid for the implementation and discussion about the thesis, all the useful informations will eventually be integrated into the final thesis;

1 Introduction to Deep Reservoir Architectures

If one wants to develop a deep recurrent network there are many ways to do so, what we will refer here as depth is the *StackedRNN* proposed in [4]

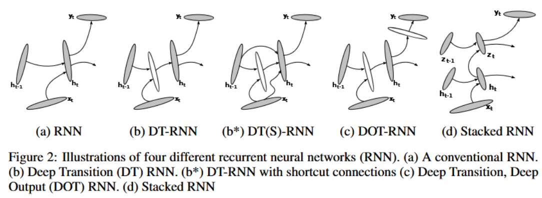

Figure 1: Various ways to go deep in RNNs

We can refer to this type of deep architecture as *sRNN* or in case of Reservoir, as *DeepESN*. Particularly the stacked version encourage the model to look at the data at different timescales, they can be used to learn complex patterns in sequential data, the depth allows the network to learn hierarchical representations of the data. We want to analize the behaviour of Recurrent Oscillator Newtwork (RON) presented in[1] with other well known Reservoir architectures like Echo State Networks and their deep counterparts. Another option that could be explored is what happens if we train such architectures when they are constructed in a "Deep" fashion, firstly we will develope the depth and structure by stacking our reservoir layers.

One fundamental property of stacking RNN in layers is that we can consider a sRNN as a dense RNN where some units are not connected.

...

2 Stacked Recurrent Oscillator Newtork

The RON model is a particular type of Reservoir Computing model, it is based on the idea of using a network of oscillators to perform computation on the input data. One could setup the **J** eigenspectrum to be stable, this is done by choosing the stiffness and dampening factors, γ, ϵ.

$$\gamma_{\min} \geq 0, \quad \gamma_{\max} \geq \frac{2}{\tau},$$
$$\epsilon_{\min} \geq 0, \quad \epsilon_{\max} \geq \frac{2}{\tau}$$

where τ is the time constant of the network.

2.1 RON equation

The RON is a discrete-time RNN model whose update reads as follows:

$$y_{k+1} = y_k + \tau z_{k+1}, \tag{1}$$

$$z_{k+1} = z_k + \tau\Big(\tanh\big(\mathbf{W}\, y_k + \mathbf{V}\, u_{k+1} + b\big) - \gamma \odot y_k - \epsilon \odot z_k\Big), \tag{2}$$

where z is the velocity of the oscillator, y is the position of the oscillator, $\mathbf{W}$ and $\mathbf{V}$ are the recurrent and input projections, respectively, and $\odot$ denotes the element-wise (Hadamard) product. k is the current time step and $\odot$ denotes the element-wise (Hadamard) product.

2.2 Deep RON

The stacked version of RON will develop the structure reservoir layer on top of each other, the layer could be clone of the same reservoir, hence having the same hyperaparameters, or they could be different, this could help to better capture the patterns at different timescales. So let L be the number of layer in the network:

Stacked RON: The update equations for the l-th layer of a Deep RON model are given by:

Position update: $y_{k+1}^{(l)} = y_k^{(l)} + \tau z_{k+1}^{(l)},$

Velocity update: $z_{k+1}^{(l)} = z_k^{(l)} + \tau \Big[\tanh\big(\mathbf{W}^{(l)}\, y_k^{(l)} + \mathbf{V}^{(l)}\, h_{k+1}^{(l-1)} + b^{(l)} \big) - \gamma^{(l)} \odot y_k^{(l)} - \epsilon^{(l)} \odot z_k^{(l)} \Big]$ (3)

In the model aforementioned, $h_{k+1}^{(l-1)}$ is the hidden state of the previous layer, $l-1$. The first layer will take the so $h_k^{(0)} = u_k$, the input data.
The output of the last layer will be the output of the network, $y_k^{(L)}$, it can be computed in two ways:

- We take only the last hidden state of the stack: $y_k^{(L)}$,
- We concatenated as last hidden state all the intermediate layers: $\mathbf{y_k^{concat}} = [y_k^{(1)}, y_k^{(2)}, \ldots, y_k^{(L)}]$

This last state will be multiplied by the trained output matrix $\mathbf{W_{out}}$ the readout

2.3 DeepESN

We will compare the RON with a stacked version of Echo State Networks, the deepESN presented in [2].

DeepESN: The update equations for the l-th layer of a DeepESN model are given by:

$$\mathbf{x}^{(l)}(t) = 1 - (a^{(l)})\mathbf{x}^{(l)}(t-1) + a^{(l)} \tanh\big(\mathbf{W}^{(l)} \mathbf{x}^{(l)}(t-1) + \mathbf{V}^{(l)} h^{(l-1)}(t) + \mathbf{b}^{(l)} \big), \tag{4}$$

where a is the leaking rate. And we treat the output of the layer as in the RON model.

3 Preliminary experiments

3.1 Memory Capacity

In this first experiment we will test the short-term memory capacity of the models thus how the trained newtwork is good at generating delayed version of a signal $u(n-k)$ with k being the delay, usually the signal is choosen to be sampled from a uniform distribution $\mathcal{U}(-0.8, 0.8)$. The Memory Capacity (MC) is defined in [3] as:

$$MC = \sum_{k=0}^{2*N_{units}} r^2(u(n-k), \bar{y}_k(n)), \tag{5}$$

With $\bar{y}_k(n)$ is reconstructed signal at time n by our net and r being the squared correlation coefficient between prediction and ground truth defined as: $r^2 = \frac{cov^2(u(n-k), \bar{y}_k(n))}{\sqrt{\sigma^2_{u(n-k)} \sigma^2_{\bar{y}_k(n)}}}$. We could compute infinite MC lags, however [3] showed we can stop after two times the amount of units in the network.

3.1.1 RON and Deep RON memory capacity

Testing the MC on the RON base model, so without adding depth, is straightforward, but we have to tweak some parameters, for example we should adjust the time constant τ to be able to capture the signal at the right timescale. We will set $\tau = 1$ since we are working with a signal of lenght 5000 and the ESN we use works with the same time constant.

Of course as in DeepESN we divide the number of parameters by number of layers. RON has more hyperparameters that change a lot the memory capacity, both stiffness and dampening factors.

3.1.2 ESN and DeepESN memory capacity

The ESN model is a well known model, we will use the same parameters as in [2].

With the same settings as in the paper we get the following results:

3.2 Distance between States

In this set of experiments we wil focus on the capacity of the network to distinguish between different but slightly similar signals. This is known as separation property and is defined simply as: The distance between two states x_1 and x_2 is defined as the euclidean distance between the two states, to do thisù we compute over two s_1, s_2 sequences of lenght $n = 500$ with one having a perturbation at timestep 100. And then we measure the distance between the two states at the end of the sequence. We expect over time the distance to decrease towards 0, the desiderata is to keep the states across time as different as possible.

$$d(x_1, x_2) = \sqrt{\sum_{i=1}^{N} (x_{1,i} - x_{2,i})^2} \tag{6}$$

Need to implement this experiment

References

[1] Andrea Ceni et al. "Random Oscillators Network for Time Series Processing". In: *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*. Ed. by Sanjoy Dasgupta, Stephan Mandt, and Yingzhen Li. Vol. 238. Proceedings of Machine Learning Research. PMLR, Feb. 2024, pp. 4807–4815. URL: https://proceedings.mlr.press/v238/ceni24a.html.

[2] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. "Deep reservoir computing: A critical experimental analysis". In: *Neurocomputing* 268 (2017). Advances in artificial neural networks, machine learning and computational intelligence, pp. 87–99. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2016.12.089. URL: https://www.sciencedirect.com/science/article/pii/S0925231217307567.

[3] Herbert Jaeger. "Short Term Memory in Echo State Networks". In: (Jan. 2002).

[4] Razvan Pascanu et al. "How to Construct Deep Recurrent Neural Networks". In: (Dec. 2013).