Thesis notes: Deep RON experimental analysis and comparison

Vincenzo Gargano

March 19, 2025

Contents

Notes

These notes shall serve as aid for the implementation and discussion of the thesis, all the useful information 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]

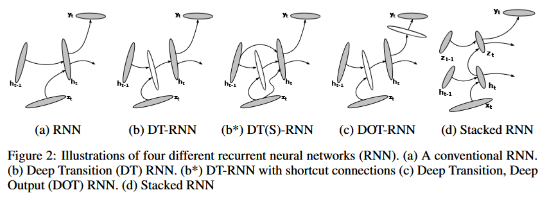

Figure 2: Illustrations of four different recurrent neural networks (RNN). (a) A conventional RNN. (b) Deep Transition (DT) RNN. (b*) DT-RNN with shortcut connections (c) Deep Transition, Deep Output (DOT) RNN. (d) Stacked RNN

Figure 1: Various ways to go deep in RNNs

We can refer to this type of deep architecture as *sRNN* or in the case of Reservoir, as *DeepESN*. Particularly the stacked version encourages the model to look at the data at different timescales, they can be used to learn complex patterns in sequential data, and the depth allows the network to learn hierarchical representations of the data. We want to analyse the behaviour of Recurrent Oscillator Network (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 develop 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 set up the $\mathbf{J}$ eigenspectrum to be stable, this is done by choosing the stiffness and dampening factors, γ, ϵ.

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

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 a clone of the same reservoir, hence having the same hyperparameters, or they could be different, this could help to better capture the patterns at different timescales. So let L be the number of layers 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 the 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.

2.4 Simple Cycle

The idea is that we want to pass the output of the last layer to the first layer, this could be done in different ways: for example, we can model the connection from the last state to the first as a sort of concatenation with the input such as a residual connection, and use a linear projection or even 1D convolution, we could even train it but since we're working with reservoir computing we will leave it untrained. Another way is to add the last state at last layer, let's call it: $y_k^{(L)}$ to the computation of the preactivation of the first layer in this way, let $\mathcal{F}$ be the new projection for the cycle part.

$$y_{k+1}^{(1)} = \tanh\big(\mathbf{W}_{rec}^{(1)} y_k^{(1)} + \mathbf{V}^{(1)} u_{k+1}^{(0)} + \mathcal{F} y_k^{(L)}\big) \tag{5}$$

This is a general form for the two models we're considering, theoretically by using the simple cycle we should improve the MC of the 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 network is good at generating a delayed version of a signal $u(n-k)$ with k being the delay, usually the signal is chosen to be sampled from a uniform distribution $\mathcal{U}(-0.8, 0.8)$. The Memory Capacity (MC) is defined in [3] as:

$$MC = \sum_{k=0}^{2*N_{units}} r^2(u(n-k), \bar{y}_k(n)), \tag{6}$$

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. Practically speaking the MC is computed by first applying the model forward pass to our data, collecting the hidden states and then for each lag we train a linear model such as Ridge Regression to predict the output for that timestep.

3.1.1 RON and Deep RON memory capacity

Here we show the results of the memory capacity of RON and Deep RON, for the hyperparameters search it's been conducted a Bayesian Search with the following parameters boundaries: $\gamma \in [0, 2.5]$, $\epsilon \in [0, 2.5]$, $\tau \in [0.001, 1]$. We will refer to the τ also as the dt.

Units	**Layers**	**Scaling**	ρ	**dt**	γ	ϵ	**MC Train**	**MC Test**
100	1	0.2	0.99	0.3	1.32	1	21.68± 2.61	16.75±4.04
100	1	0.2	0.99	0.95	1.14	0.88	26.70± 1.48	21.26±2.65
100	10	0.2	0.99	0.95	1.14	0.88	23.40± 9.45	17.54±12.54

Table 1: Experiment results on memory capacity shallow RON

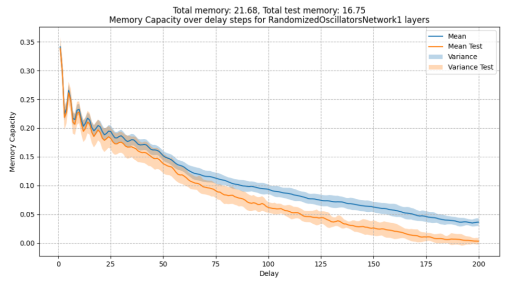

Figure 2: Memory Capacity on RON with low dt (0.3)

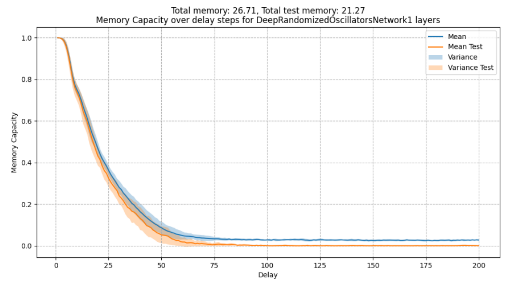

Figure 3: Memory Capacity on RON with high dt (0.95)

We reported in the hyperparameters search two different results, one with a low τ and one with a high τ, changing this hyperparameter makes the network behave differently, we can see that in the low τ the network presents an oscillatory component in the memory capacity over steps, on the other hand, high τ removes this behaviour and performs better. Running Deep RON with these same parameters yields high variance, in fact, the matrix results are Ill-conditioned in some trials, An explaination could be numerical instability of the model.

Forse è meglio fare anche un esperimento rimuovendo i trial dove la matrice è mal condizionata per vedere la differenza?

To avoid biasing towards several layers in the search, we ran another hyperparameters search for the deep RON and found that with the following configuration.

Units	**Layers**	**Scaling**	ρ	**dt**	γ	ϵ	**MC Train**	**MC Test**
100	10	0.2	0.99	0.5923	1.632	0.78	31.88± 4.69	26.29±7.95

Table 2: Experiment results on memory capacity on Deep RON

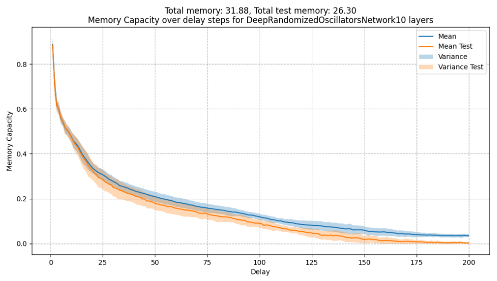

Figure 4: Memory Capacity on Deep RON

Aggiungere spiegazione perchè funziona meglio 10L rispetto a 1L, ma non pareggia ESN

Reproduce leaky esn with RON

Take only last state for MC both RON and ESN

3.2 RON as Leaky ESN

By setting the RON equation hyperarameters $\epsilon = 1/\tau$ and $\gamma = 1$, the z and y states are decoupled, thus we get the following equation:

$$y_{k+1} = \tau^2 \tanh(\mathbf{W} y_k + \mathbf{V} u_{k+1} + b) + (1 - \tau^2) y_k \tag{7}$$

Other than the parameter τ^2 if we take the square root and call it a or *leaky* of it we get the Leaky-ESN equation. The results we get from this models are equal both for Adiac and memory capacity. : 22+-4 for ron as leakyesn and 24+-1.75 for leakyesn. if we put like leaky and $\tau = \sqrt{leaky}$ at a value different than 1: eg for $leaky = 0.5$, by setting $\tau = 0.707$ and $\epsilon = 1.414$. For memory capacity we get different results if we set leaky or RON parameters according to what we said.

Ring topology: take concat states and last hidden layer state, both ESN and RON

try adiac for classification ts, hyperparameters on paper RON

report values

For Adiac with $leaky = 1$: 0.2473 accuracy for ron and 0.258 accuracy for esn, with $leaky = 0.5$ 0.344 for esn and 0.328 for DeepRON

3.3 Use only the last state

If we use only the last state without concatenating trought layer performance drops quite significantly

- Deep ESN with 10 layers using only last state gets 9.61 mc
- DeepRON with 10 layers using only last state gets 8.49 mc

3.4 Adiac Results

Results for Adiac with the same parameters from the paper and changing just the number of layers yields the following results

Maybe we don't scale the reservoir between layers, reservoir scaler is not used

Layers	**Model**	**Accuracy**
1	Shallow RON	0.7053
10	DeepRON	0.5583
1	Shallow ESN	0.5447
10	DeepESN	0.5066

3.4.1 ESN and DeepESN: Memory Capacity

The ESN model is well known, we will use the same parameters as in [2]. With the same settings as in the paper, we get the following results:

Use best parameters from the paper

3.5 Simple Cycle: Memory Capacity

How the simple cycle affects the memory capacity of our models? We will test the simple cycle we defined in 2.4 on the Deep RON and Deep ESN models. We conducted hyperparameters selection for both models' type with Random Search this time.

For DeepESN we use the following set of parameters for the simple cycle:

Qui l'inter scaling è sempre 0.1 cambiandolo peggiorano i risultati

Type	Layers	**Units**	**rho**	**MC Train**	**MC Test**
Standard	3	100	0.9	31.17	27.72
Simple Cycle	3	100	0.9	32.18	28.33
Standard	10	100	0.9	37.58	33.74
Simple Cycle	10	100	0.9	37.83	33.80

Table 3: Experiment Simple Cycle Deep ESN

For Deep RON we use the following set of parameters

Ho provato anche a cambiare inter scaling tra i layer ma peggiora

Type	**Layers**	**Units**	**Scaling**	ρ	**dt**	γ	ϵ	**MC Train**	**MC Test**
Standard	3	100	0.2	0.99	0.67	1.9	0.76	21.91	18.09
Simple Cycle	3	100	0.2	0.99	0.67	1.9	0.76	19.32	15.52
Standard	10	100	0.2	0.99	0.67	1.9	0.76	32.02	28.40
Simple Cycle	10	100	0.2	0.99	0.67	1.9	0.76	30.82	27.08

Table 4: Experiment Simple Cycle Deep RON

3.6 Distance between States

In this set of experiments, we will 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 are to keep the states across time as different as possible. For this task, we used the same *Artificial dataset* described in [2], we have a 10-hot encoded vector of an alphabet of 10 characters.

$$d(x_1, x_2) = \sqrt{\sum_{i=1}^{N} (x_{1,i} - x_{2,i})^2} \tag{8}$$

3.6.1 RON and Deep RON distance between states

The RON models behave quite differently by changing the parameters the τ is fundamental in this task. Now if we consider the Deep RON with the parameters that performed better in the MC task and compare it to the DeepESN model we get the following results:

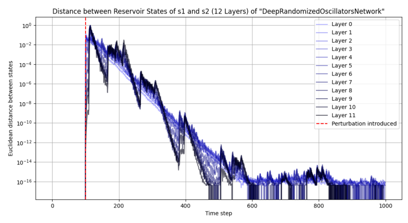

Figure 5: Distance between states on Deep RON

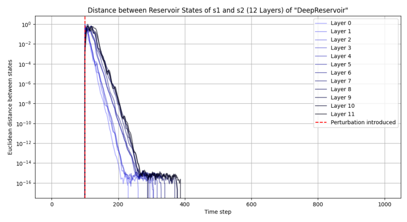

Figure 6: Distance between states on Deep ESN

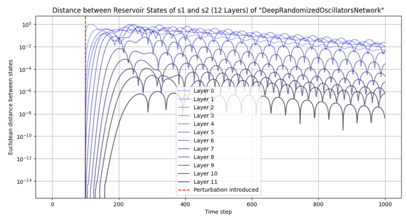

Figure 7: Distance between states on Deep ESN with low dt

Con seed diversi su DeepRON i risultati del plot cambiano molto

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).