Echo-state network and its memory capcity

Jakub Bahyl[1]
Faculty of mathematics, physics and informatics, UK in Bratislava, Slovakia

In this work we present results of short-term memory capacity of a simple recurrent neural network - the *echo state network*. This network was trained using random data from a uniform distribution and tested against its ability to replicate the past. We examine suitability of various parameters, such as window depth, reservoir initialisation, activation functions and sparsity of synaptic weights.

Keywords: recurrent NN, memory capacity

"The human brain has 100 billion neurons, each neuron connected to 10 thousand other neurons. Sitting on your shoulders is the most complicated object in the known universe."
Michio Kaku

I. TECHNICAL NOTE

The code is written in a modern `Julia v0.5.1` - a dynamic programming language for numerical computing. The source code, executable with Jupyter environment using Julia kernel, can be found on this GitHub. A free online environment for of Jupyter with integrated Julia can be found on Juliabox.

II. ECHO-STATE NETWORK IN SHORT

Echo state network (ESN) is an architectural type of recurrent neural network suitable for a supervised learning. The main idea is based on the existence of a reservoir - a single hidden layer which creating its own copy after each input. This copy is later used with the next input to produce the output layer. Such network effectively uses its own memory capacity and consequently, modules the input signal.

In this work, we worked with a highly restricted ESN:

- no directed connections between an input and output
- connections of reservoir and the output are one-way (without feedback)
- one-level memory (ESN remembers only the very last reservoir state)
- the output weight matrix is constructed using pseudo-inverse

In the picture (FIG.1), you can see the depiction of the network:

III. INITIALISATION OF ESN

During each epoch, a single input number is chosen $x_i \in \mathrm{Uni}(-1,1)$ and given to the network. This input

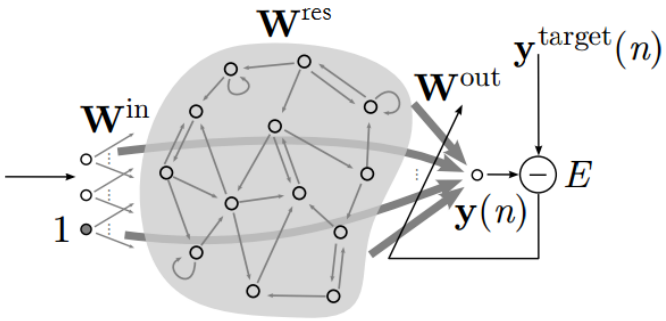

FIG. 1. A schematic of a simple ESN

is moved to the reservoir $\mathbf{r}_i$ of 100 neurons via weight matrix $\mathbf{W}_{in}$. The reservoir recalls its past state $\mathbf{r}_{i-1}$ and activates itself into the new state:

$$\mathbf{r}_i = \tanh(\mathbf{W}_{\mathrm{in}} x_i + \mathbf{W}_{\mathrm{res}} \mathbf{r}_{i-1})$$

Matrices $\mathbf{W}_{\mathrm{in}}$, $\mathbf{W}_{\mathrm{res}}$ are initialised as $\mathbf{W}_{\mathrm{in}} \in 10^{-2} \cdot \mathrm{Uni}(-1,1)$ and $\mathbf{W}_{\mathrm{res}} \in \mathrm{Norm}(0,1)$. Moreover, $\mathbf{W}_{\mathrm{res}}$ is additionally scaled by a number C so that we could ensure the magnitude of the highest eigenvalue.

The output vector $\mathbf{y}_i$ is expected to be of size L, which should reconstruct the past *true* outputs $d_{i-1}, ..., d_{i-L}$. According to this the macroscopic squared error is calculated as:

$$\mathrm{Err} = \frac{1}{L\#} \sum_{l=1}^{L} \sum_{i=1}^{\#} (d_{i-l} - y_i(l))^2$$

where # is the number of used data for testing.

IV. EXPERIMENT

We used the first 100 random inputs $\mathbf{x}_{100}$ for a *network cleansing*. Using the resultant $\mathbf{r}_{100}$, we started the training part with 500 random inputs $\mathbf{x}_{500}$. After each input, we externally saved the state of reservoir $\mathbf{r}_i$, which resulted in reservoir matrix $\mathbf{R}^{500\times100}$. This state history is used as a generator of a new output matrix $\mathbf{W}_{\mathrm{out}}$, which would map such states into the correct results $\mathbf{D}$:

$$\mathbf{W}_{\mathrm{out}} = \mathbf{D}\mathbf{R}^{+}$$

The testing part was performed using another 500 pieces of data. Again, we collected the history of states $\mathbf{R}$ and constructed the predictions as:

$$\mathbf{Y} = \mathbf{W}_{\text{out}}\mathbf{R}$$

Total memory capacity

We repeated the experiment for various L and then calculated the total memory capacity (TMC) as:

$$\text{TMC} = \sum_{l=1}^{L} \frac{\text{cov}^2(\mathbf{Y}_l, \mathbf{D}_l)}{\text{var}(\mathbf{Y}_l)\text{var}(\mathbf{D}_l)}$$

The typical graph of TMC against spectral radius (*spectral radius* = the highest eigenvalue of square matrix) of $\mathbf{W}_{\text{res}}$ is depicted (FIG.2) below:

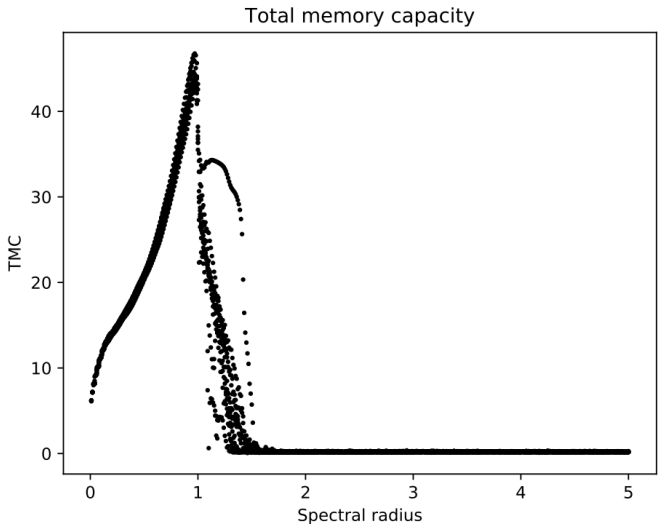

FIG. 2. TMC behaviour for fixed $L = 80$.

Apparently, unless a critical value of the spectral radius is reached, ESN performs a non-zero memory capacity. After this critical value, the ability to remember the past is negligibly small.

The ideal window length L

It might seem that the higher L implicates the higher TMC. However, this doesn't have to be true. Below (FIG.3) is depicted the evolution of TMC using various window depths:

If L is too low, the TMC maximum is bounded and cannot be exceeded. In case of higher L, the TMC maximum occurs exactly when spectral r. $\to 1$. After this achievement, the TMC curve falls down hitting the zero level.

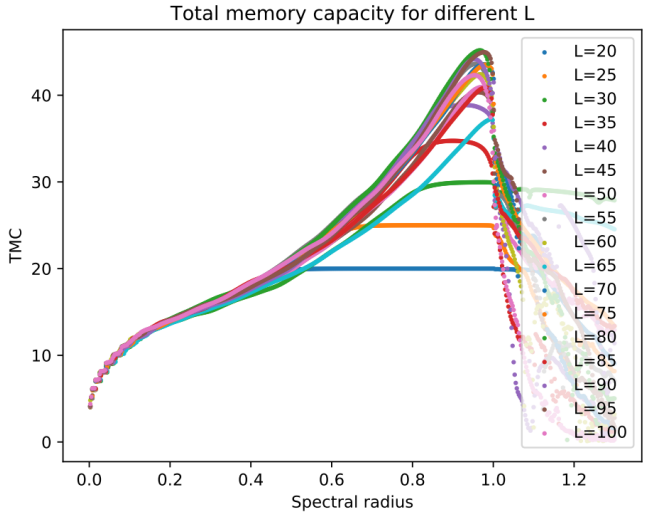

FIG. 3. Dependency of TMC for $L \in \{20, 25, ..., 95, 100\}$

We chose a fixed the best $L = 95$, which maximises the TMC value. Then we performed the same evolution 500-hundred times, to confirm the hypothesis about the critical value:

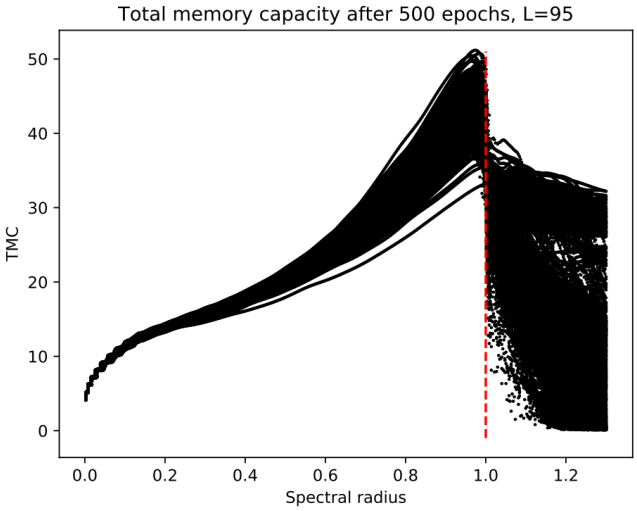

FIG. 4. Dispersion of TMC for $L = 95$.

Such drop is related with the *vanishing problem*, which regularly occurs when using recurrent NNs, especially with symmetric activation function (such as $\tanh(x)$).

V. ADDITIONAL EXPERIMENTS

Logistic activation

When we replaced the $\tanh(x)$ function with the *logit* function $1/(1+\exp(x))$, the TMC evolution curves reach the maximum approximately near spec rad ≈ 7. Although, the position of maximum is not so clear and the *vanishing problem* appears in its weaker form.

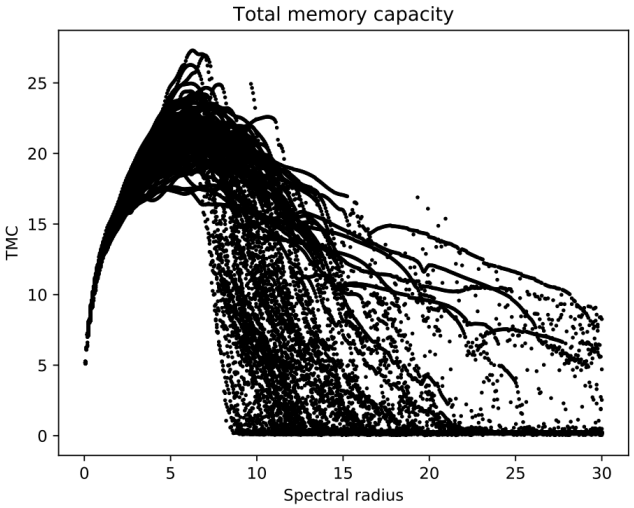

FIG. 5. Dispersion of TMC(ρ) using logit activation.

Reservoir sparsity

One of the common methods to achieve the network improvement is the implementation of the weight matrix sparsity (percentage of non-zero elements) $\mathbf{W}_{\text{res}}$. The example evolution of TMC for extreme sparsiity cases:

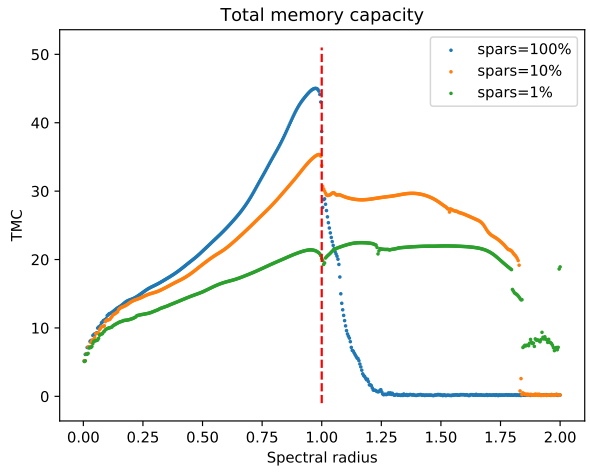

FIG. 6. TMC with spectral radius for 100%, 10% and 1% sparsity.

The global trend seems to be decreasing with the sparsity loss. Therefore, we made a statistics on TMC maximum (believed that the maximum always lied at spec rad $= 1$) for various sparsity between 80% and 100%:

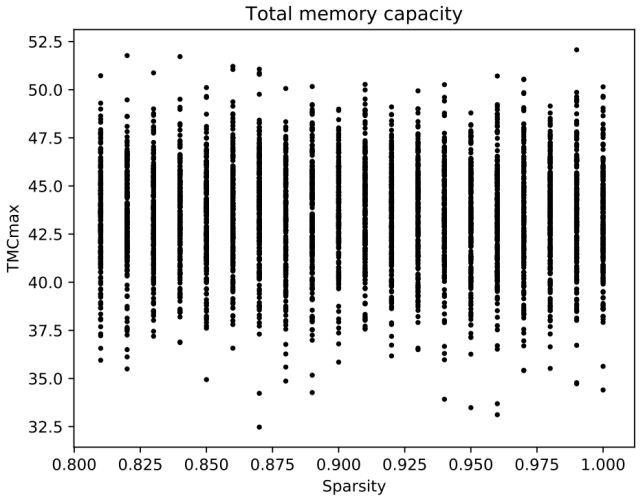

FIG. 7. The statistics of TMC maximum for various values of $\mathbf{W}_{\text{res}}$ sparsity.

MC contributions

Since TMC is composed of single MC_l contributions for $l \in \{1, 2, .., L\}$:

$$\text{TMC} = \sum_{l=1}^{L} \text{MC}\,,$$

it is worth to ask, which MC contributions are the biggest ones. Below (FIG.8) we plotted the contribution evolution against the window depth:

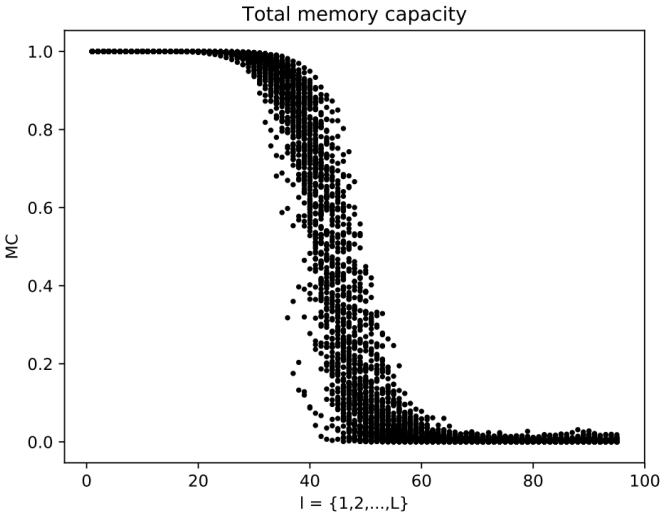

FIG. 8. Contributions to TMC for various l.

VI. CONCLUSIONS

There emerged a few observations from our analysis:

- ESN behaves as expected, although our implementation is much simpler than the architecture of original authors.
- Due to the *vanishing* effect, the TMC reaches maximum near the point, where spectral radius takes the value 1 (from the left side). The dispersion of TMC value is quite big (around 20%) due to too high L and low number of used neurons.
- The most reliable method for choosing optimal L is probably from the MC plot. The plot provides the visualisation of the best tradeoff between the highest TMC (choosing the highest L) and the computational resources.
- The logistic activation produces much more uncertain localisation of TMC maximum and in general, more chaotic *vanishing* process. What is event more important, the TMC values are generally smaller than those when using hyperbolic tangent activation.
- It seems that the reservoir matrix sparsity doesn't take a big role in network improvement. Again, this can be caused by the small number of used neurons.

I would to thank to my NN lecturer prof. Igor Farkaš and my consultant Tomas Kuzma for all fruitful discussions, which led me through these experiments and their interpretations.