Learning Timescales in MTRNNs

Fabien C. Y. Benureau (P)[1], and Jun Tani[1]

[1] Cognitive Neuro-Robotics Unit , Okinawa Institute of Science and Technology
E-mail: jun.tani@oist.jp

Abstract— We test the viability of having learnable timescales in multi-timescales recurrent neural networks.

Keywords— Machine Learning, Recurrent Neural Networks, Timescales

1 Motivation

This article is quite modest. It seeks to provide preliminary answers to the question: Can and should multi-timescales recurrent neural networks learn their own timescales? Multi-Timescales Recurrent Neural Networks (MTRNN) [1] are recurrent neural networks with several recurrent layers stacked onto one another. Each layer connects to the layer above and below, and each layer has its own timescale, which acts as a decay factor that controls how much the layer state over the past timesteps influence the current hidden state. Typically, inspired by the brain, the lower layers—which handle the raw sensory input—have short timescales while the higher layers, that deal with abstract features, have long timescales and react to broad changes in the dynamics of the network. In published MTRNN experiments, timescales values are usually set arbitrarily. A popular choice has been exponential timescales, such as setting the kth layer timescale to 2^k (in [2]) or 2^{k+1} (in [3]).

We can wonder, however, if those timescales could not be learned during training, like any other connection weight. And if doing so brings any benefits to the learning performance. In this article, we respond yes to the first question, and remain inconclusive about the second.

2 Method

2.1 Network

We employ a MTRNN [1]. The equations for the layer k are:

$$\boldsymbol{h}^k_{t+1} = \left(1 - \frac{1}{\tau_k}\right)\boldsymbol{h}^k_t + \frac{1}{\tau_k}\left(\boldsymbol{L}\boldsymbol{d}^{k-1}_t + \boldsymbol{S}\boldsymbol{d}^k_t + \boldsymbol{H}\boldsymbol{d}^{k+1}_t + \boldsymbol{b}\right)$$

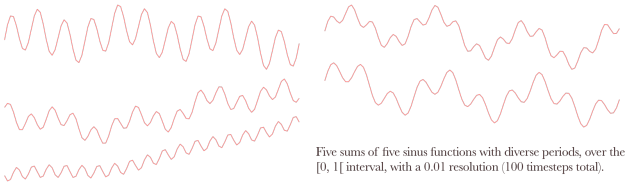
$$\boldsymbol{d}^k_{t+1} = \tanh(\boldsymbol{h}^k_{t+1})$$

with $\boldsymbol{h}^k_t$ the hidden state of the layer k at time t, $\boldsymbol{d}^k_t$ the activity of its units, and $\boldsymbol{L}$, $\boldsymbol{S}$, and $\boldsymbol{H}$ the weight matrices for the connection to the lower layer $k-1$, the layer itself, the upper layer $k+1$; and $\boldsymbol{b}$ the bias term. Here, we will consider a network with three layers of 25 units each.

The time constant τ_k is the focus of this paper here. Rather than a fixed scalar, we transform τ_k in a learnable weight in the network, modified through back-propagation through time. Moreover, based on our experience, while the learning rate of the rest of the network is 0.001, the specific learning rate for those τ_k weights is 0.01, ten times higher. Additionally, the value of τ_k is clamped to a minimum of 1, to avoid a degenerative behavior of the layer equation.

2.2 Dataset and Training

We consider five sums of one-dimensional sinus patterns of 100 timesteps each, as seen on the figure below.

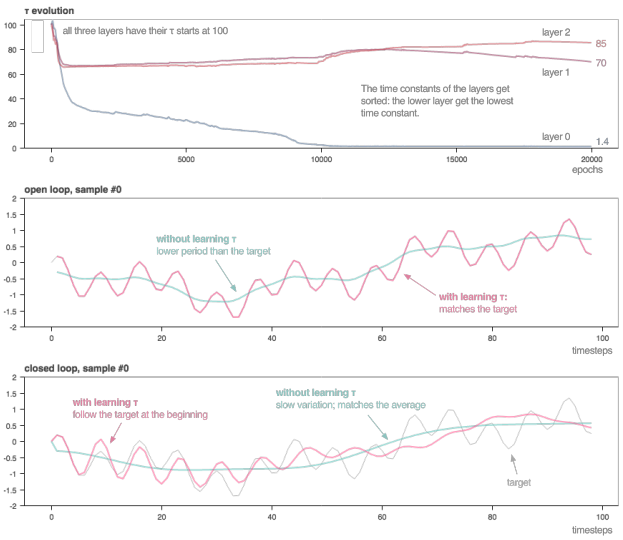


Five sums of five sinus functions with diverse periods, over the [0, 1[interval, with a 0.01 resolution (100 timesteps total).

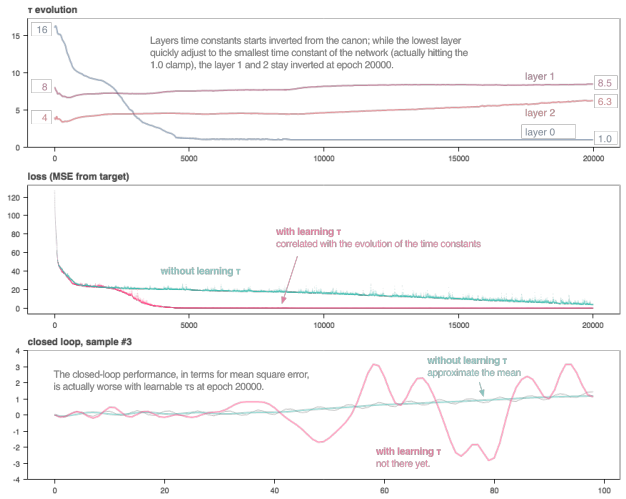
The training is done over 20000 epochs, with a 0.001 learning rate, and considering a mean-squared error loss over the dataset patterns. We evaluate the network by its capacity to predict the next timestep, and to reproduce the pattern in closed-loop: the output of a timestep is used as input for the next timestep.

3 Results

We consider a network initialized with all time constants at 100 (we pushed the learning rate of the τ to 0.1 for this network). Those time constant values are too high; they won't allow the network to generate the fast variations of the target patterns. As the figure below shows, learning τs during training allow the network to correct for this initialization. The lowest layer adapts to a low τ value, while the layer 1 and 2 remain fairly stable.

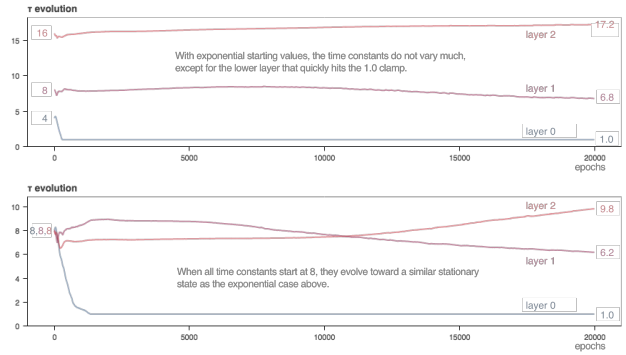


Note that while in this experiment, the time constants ended-up "in order", layers do not necessarily get all ordered when learning τ. If we consider initial τ values 16, 8 and 4 for layer 0, 1 and 2, layer 0 again gets a low τ value fast, but layer 2 keeps a lower τ value than layer 1, at least at epoch 20000, as the following figure shows.



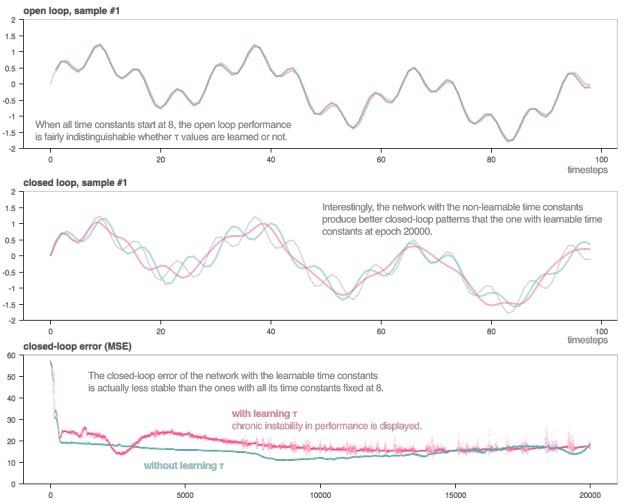
The loss of the network benefits from the adaptation of the τ values. However, its closed-loop error actually is worse: the network with fixed τ is mostly approximating the slow mean of the samples, while the network with learnable τ is starting to learn to reproduce faster features of the patterns, but is not yet good at it.

So: are exponential τ values a good way to initialize a MTRNN? The figure below suggests they might: τ values of layer 1 and 2 hardly change when the network is initialized with exponential τ values. And when initializing all layers at $\tau = 8$, the network evolves toward the similar state at the exponential initialization.



However, when looking at the output of the network with $\tau = 8$, those well-differentiated time constants do not necessarily help the network at producing the target patterns at epoch 20000, as shown in the figure that follows.

Contrary to our expectations, and while the network with learnable τ seems to have settled on good τ values, the network with a fixed $\tau = 8$ for all three layers produces better patterns in the closed-loop case. The graph of the closed-loop error shows that the former network is also less stable numerically.



4 Discussion

So let's review: learning τ in MTRNN works in some cases. It can allow a network to adapt to terrible parameter initialization. It is not clear, however, that the values that are derived from this learning process are the best ones for the target patterns. The learning process, for instance, consistently hits the hard clamp at 1.0 for the lowest layer. If we remove it, the network will adopt values of τ as low as 0.6, creating a dynamics that will degrade closed-loop learning and stability. Furthermore, it is not clear that the MTRNN performance is highly dependent on the precise value of its time constants, as the network with all layers with $\tau = 8$ testifies. One could posit that approximately good, fixed time constants might lead to faster and more stable performance than adaptive ones.

This article is exploratory. More complex dataset, simulations and analysis are needed to understand if and how MTRNN can benefit from learning their time constant in an autonomous manner.

Code Availability

The source code will be made available for review shortly after submission at `https://figshare.com/s/d90ec851f08af442627f`.

References

[1] Yamashita, Y., Tani, J. (2008). Emergence of Functional Hierarchy in a Multiple Timescale Neural Network Model: A Humanoid Robot Experiment. *PLoS Computational Biology*, 4(11), e1000220.

[2] Choi, M., Tani, J. (2018). Predictive Coding for Dynamic Visual Processing: Development of Functional Hierarchy in a Multiple Spatiotemporal Scales RNN Model. *Neural Computation*, 30(1), 237-270. doi: 10.1162/neco_a_01026

[3] Ahmadi, A., Tani, J. (2017). Bridging the Gap Between Probabilistic and Deterministic Models: A Simulation Study on a Variational Bayes Predictive Coding Recurrent Neural Network Model. *Neural Information Processing*, 760-769. doi: 10.1007/978-3-319-70090-8_77