Neural Networks 166 (2023) 645–669

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Warming up recurrent neural networks to maximise reachable multistability greatly improves learning

Gaspard Lambrechts [a,*,1], Florent De Geeter [a,1], Nicolas Vecoven [a,1], Damien Ernst [a,b], Guillaume Drion [a]

[a] Montefiore Institute, University of Liège, 10 allée de la découverte, Liège, 4000, Belgium
[b] LTCI, Telecom Paris, Institut Polytechnique de Paris, 19 place Marguerite Perey, Palaiseau, 91120, France

ARTICLE INFO

Article history:
Received 27 July 2022
Received in revised form 12 June 2023
Accepted 14 July 2023
Available online 7 August 2023

Keywords:
Recurrent neural network
Multistability
Initialisation procedure
Long-term memory
Warmup
Long time dependencies

ABSTRACT

Training recurrent neural networks is known to be difficult when time dependencies become long. In this work, we show that most standard cells only have one stable equilibrium at initialisation, and that learning on tasks with long time dependencies generally occurs once the number of network stable equilibria increases; a property known as multistability. Multistability is often not easily attained by initially monostable networks, making learning of long time dependencies between inputs and outputs difficult. This insight leads to the design of a novel way to initialise any recurrent cell connectivity through a procedure called "warmup" to improve its capability to learn arbitrarily long time dependencies. This initialisation procedure is designed to maximise network reachable multistability, i.e., the number of equilibria within the network that can be reached through relevant input trajectories, in few gradient steps. We show on several information restitution, sequence classification, and reinforcement learning benchmarks that warming up greatly improves learning speed and performance, for multiple recurrent cells, but sometimes impedes precision. We therefore introduce a double-layer architecture initialised with a partial warmup that is shown to greatly improve learning of long time dependencies while maintaining high levels of precision. This approach provides a general framework for improving learning abilities of any recurrent cell when long time dependencies are present. We also show empirically that other initialisation and pretraining procedures from the literature implicitly foster reachable multistability of recurrent cells.

1. Introduction

Despite their performances and widespread use, recurrent neural networks (RNNs) are known to be blackbox models with extremely complex internal dynamics. A growing body of work has focused on understanding the internal dynamics of trained RNNs (Ceni, Ashwin, & Livi, 2020; Maheswaranathan, Williams, Golub, Ganguli, & Sussillo, 2019; Sussillo & Barak, 2013), providing invaluable intuition into the RNN prediction process. This viewpoint has already been used to understand the difficulties for RNNs to capture longer time dependencies (Bengio, Frasconi, & Simard, 1993; Doya, 1993). In particular, recent work has highlighted the important role played by fixed points in RNN state spaces, that are defined as hidden states that updates to themself for a given input (Katz & Reggia, 2017; Sussillo & Barak, 2013). This line of work has argued that locating such fixed points efficiently could provide insights into RNN dynamics and input–output properties. Here, we build upon this line of work by studying the impact of the number of reachable fixed points in an RNN on the ability to learn long time dependencies. Moreover, we highlight how maximising the number of reachable fixed points at initialisation can improve RNN learning, in particular in the presence of arbitrarily long dependencies.

More precisely, we introduce a fast-to-compute measure of the multistability of a network called variability amongst attractors (VAA). This measure gives the number of reachable attractors for a set of initial states. We show that loss decrease during learning in tasks with long time dependencies is highly correlated with an increase in VAA, highlighting both the relevance of the measure and the importance of multistability for efficient learning. Second, we use stochastic gradient ascent on a differentiable proxy of the VAA, called VAA*, as a way of maximising the number of reachable attractors within the network at initialisation. We show that this technique strongly improves performance on long time dependencies benchmarks, at the cost of precision, the latter relying on the richness of network transient dynamics.

* Corresponding author.
E-mail addresses: gaspard.lambrechts@uliege.be (G. Lambrechts), florent.degeeter@uliege.be (F. De Geeter), dernst@uliege.be (D. Ernst), gdrion@uliege.be (G. Drion).

[1] These authors contributed equally.

https://doi.org/10.1016/j.neunet.2023.07.023
0893-6080/

Third, we propose a parallel recurrent network structure with a partial warmup that enables one to combine long-term memory through multistability with precision through rich transient dynamics. Finally, we show empirically that other methods from the literature such as the chrono initialisation and the bistable recurrent cells implicitly achieve the same goal of maximising the number of reachable attractors. Another pretraining procedure, the auxiliary losses proposed by Trinh, Dai, Luong, and Le (2018), are also shown to foster multistability and to achieve good results on benchmarks with long time dependencies, using a much heavier procedure. For the sake of clarity, those results are only reported in Appendix H.

In Section 2, related works on training RNNs in the presence of long time dependencies are presented. In Section 3, RNNs are introduced as dynamical systems and the concept of multistability is introduced for those systems. In Section 4, the supervised learning and reinforcement learning benchmarks are given. In Section 5, the VAA is introduced along with the estimation procedure of the multistability of an RNN for a set of initial states. The correlation between multistability and learning is shown empirically on the benchmarks with long time dependencies. In Section 6, the VAA* is introduced along with the warmup procedure that fosters multistability at initialisation. The benefits of warmup are shown empirically on benchmarks with long time dependencies. In addition, the double-layer architecture with partial warmup is introduced and shown to achieve a better performance on all benchmarks. Finally, Section 7 concludes and proposes several future works.

2. Related works

Training RNNs is known to be difficult when time dependencies become too long (Pascanu, Mikolov, & Bengio, 2013). Indeed, the most used algorithm to train RNNs is the backpropagation through time (BPTT) algorithm (Werbos, 1990), which unrolls the RNN to see it as a feedforward neural network with shared weights before applying the backpropagation. However, the longer the sequence, the deeper the corresponding feedforward neural network is. Backpropagating through such deep networks often leads to vanishing or exploding gradients, and different methods have been proposed to tackle this issue. These methods usually act on one of three different levels: the training, the initialisation/pretraining and the network architecture.

Training. These methods modify the training of RNNs. For instance, clipping the gradients (Pascanu et al., 2013) prevents the gradients from exploding. Another example is the truncated variant of BPTT (Williams & Zipser, 1995), which does not propagate gradients through the whole sequences, but rather through parts of these sequences, leading to gradients that vanish or explode less often. It is likely that truncating the BPTT prevents from learning long time dependencies efficiently. Finally, Trinh et al. (2018) propose adding auxiliary losses at some timesteps, to avoid having only one loss computed at the end of the sequences. These losses are computed in an unsupervised fashion: either a decoder has to reconstruct a part of the sequence (*reconstruction* loss), or a network has to predict the next input (*prediction* loss). This method can also be used as a pretraining to first train the RNN to encode correctly the sequences. This work achieved good results on very long sequences, which motivated the aforementioned comparison with our work in Appendix H.

Initialisation/pretraining. The goal of these methods is to bring the network weights to a better place in the parameters space where the learning will be better and faster. Notably, the chrono-initialisation (Tallec & Ollivier, 2018; Van Der Westhuizen & Lasenby, 2018) changes the initial biases parameters to improve the learning of long time dependencies. Some pretraining methods rely on autoencoders: Pasa and Sperduti (2014) use the parameters of a linear encoder as initial weights for the RNN, Sagheer and Kotb (2019) train a LSTM-based stacked autoencoder layer-wise before adding a output layer and fine-tuning on the dataset and Ong, Sugiura, and Zettsu (2014) introduce a dynamic pretraining of AE specifically made for time-series. Pasa, Testolin, and Sperduti (2015) pretrain the RNN on a smoothed version of the dataset produced by a first-order hidden Markov model and then fine-tunes on the original dataset. Tang, Wang, and Zhang (2016) first train a DNN before using it as a teacher to train the RNN. Ienco, Interdonato, and Gaetano (2019) focus on multi-class sequences classification. A trained RNN is used to rank the classes by decreasing order of complexity, then a new RNN is pretrained to predict the most complex class, then the second one, etc. All these pretraining methods have improved the performance of RNNs either on classification or on time-series prediction tasks. While making the final training of the network easier and better, none of them seems to directly promote the learning of long time dependencies.

Network architectures. The most notable improvement made in the RNN architectures is the introduction of the gates, which are used to control the flow of information in the network and to help the gradients to propagate through the time. These gates have led to the development of the long-short term memory (LSTM) (Hochreiter & Schmidhuber, 1997) and the gated recurrent unit (GRU) (Cho, Van Merriënboer, Bahdanau, & Bengio, 2014), which are now the most used RNNs in practice. In the experiments, we also consider the minimal gated unit (MGU) (Zhou, Wu, Zhang, & Zhou, 2016), a minimal design among gated recurrent units that only has one gate. Other approaches include the introduction of different time-scales inside the RNN. The segmented-memory RNN (Chen & Chaudhari, 2009) splits the sequences into segments and uses a two-layers RNN, where the first layer is reset at the end of each segment, while the second one is updated when a new segment begins. The hierarchical RNN (Hihi & Bengio, 1995), the hierarchical multiscale RNN (Chung, Ahn, & Bengio, 2017) and the clockwork RNN (CW-RNN) (Koutnik, Greff, Gomez, & Schmidhuber, 2014) stack recurrent layers that are updated at different frequencies. The structurally constrained recurrent network (SCRN) (Mikolov, Joulin, Chopra, Mathieu, & Ranzato, 2015) imposes some constraints on a subset of the recurrent weights, forcing some neurons hidden states to be slowly updated. The nonlinear autoregressive with exogenous inputs (NARX) RNN (Lin, Horne, Tino, & Giles, 1996; Menezes & Barreto, 2008) uses the n previous hidden states as inputs, making it a nth-order RNN. Likewise, novel recurrent cell dynamics, such as the bistable recurrent cell (BRC) and the neuromodulated BRC (NBRC) (Vecoven, Ernst, & Drion, 2021), have been introduced to help tackle long time dependencies benchmarks. NBRCs were specifically designed to maximise reachability of cellular bistability, providing a way to create never-fading memory at the cellular level. These results highlighted how dynamics of untrained RNNs, i.e., at initialisation, can strongly impact learning performance of RNNs. In this work, we extend this approach at the network level by maximising multistability of any recurrent cell type prior to learning. To this end, we propose a novel RNN pretraining procedure called "warmup" that is designed to maximise the number of RNN attractors that can be reached from hidden states resulting from input sequences. Compared to pretraining methods, this method is very efficient since it only requires a few gradient steps before reaching a multistable regime for the RNN.

3. Background

In this section, RNNs are formalised as dynamical systems. The fixed points of these systems are defined, and the notions of attractors, reachable attractors and multistability are introduced.

3.1. Recurrent neural networks

RNNs are parametric function approximators that are often used to tackle problems with temporal structure. Indeed, RNNs process the inputs sequentially, exhibiting memory through hidden states that are outputted after each timestep, and processed at the next timestep along with the following input. These connections allow RNNs to memorise relevant information that should be captured over multiple timesteps. More formally, an RNN architecture is defined by its update function f, its output function g and its initialisation function h that are parameterised by a parameter vector $\theta \in \mathbb{R}^d$. Let $\mathbf{u}_{1:T} = [\mathbf{u}_1, \dots, \mathbf{u}_T]$, with $T \in \mathbb{N}$ and $\mathbf{u}_t \in \mathbb{R}^n$, an input sequence. RNNs maintain an internal memory state $\mathbf{x}_t$ through an update rule $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t; \theta)$ and output a value $\mathbf{o}_t = g(\mathbf{x}_t; \theta)$, where the initial hidden state $\mathbf{x}_0 = h(\theta)$ is often chosen to be zero. We note that often, the output of the RNN is simply its hidden state $\mathbf{x}_t$, i.e. g is the identity function. RNNs can be composed of only one recurrent layer, or they can be built with L layers that are linked sequentially through $\mathbf{u}_t^i = \mathbf{o}_t^{i-1}$ with $\mathbf{u}_t^1 = \mathbf{u}_t$ and $\mathbf{o}_t = \mathbf{o}_t^L$, where $\mathbf{o}_t^i$ denotes the output of layer i and $\mathbf{u}_t^i$ its input. In this case, each layer i has its own update function f^i, output function g^i and initialisation function h^i. Backpropagation through time is used to train these networks where gradients are computed through the complete sequence via the hidden states (Werbos, 1990). The following recurrent architectures are considered in the experiments: LSTM, GRU, BRC, NBRC, MGU. The specific update functions of those RNNs can be found in Appendix A. In addition, we consider the chrono-initialised LSTM.

3.2. Fixed points in recurrent neural networks

Fixed points in $\mathbf{u}$. In dynamical systems, fixed points are defined as points in the state space that map to themselves through the update function, for a given input $\mathbf{u}$. For a system f, we say that a state $\mathbf{x}^*$ is a fixed point in $\mathbf{u}$ if and only if

$$\mathbf{x}^* = f(\mathbf{x}^*, \mathbf{u}). \tag{1}$$

Attractors in $\mathbf{u}$. Fixed points can either be fully attractive (attractors), fully repulsive (repellors), or combine attractive and repulsive manifolds (saddle points). For a constant input $\mathbf{u}$, the set of starting states for which the system converges to the fixed point $\mathbf{x}^*$ is called basin of attraction of $\mathbf{x}^*$ in $\mathbf{u}$ and is written as

$$\mathcal{B}^{\mathbf{u}}_{\mathbf{x}^*} = \left\{ \mathbf{x} \;\middle|\; \lim_{n\to\infty} f^n(\mathbf{x}, \mathbf{u}) = \mathbf{x}^* \right\} \tag{2}$$

$$\text{with } f^n(\mathbf{x}, \mathbf{u}) = \begin{cases} f\left(f^{n-1}(\mathbf{x}, \mathbf{u}), \mathbf{u}\right) & \text{if } n > 1, \\ f(\mathbf{x}, \mathbf{u}) \text{ if } n = 1. \end{cases} \tag{3}$$

If the limit is not defined for some point $\mathbf{x}$, then this point does not belong to any basin of attraction in $\mathbf{u}$. Mathematically, $\mathbf{x}^*$ is an attractor in $\mathbf{u}$ if its basin of attraction in $\mathbf{u}$, $\mathcal{B}^{\mathbf{u}}_{\mathbf{x}^*}$, has a positive measure.

Reachable attractors in $\mathbf{u}$. In particular, we say that an attractor $\mathbf{x}^*$ in $\mathbf{u}$ is reachable from some state $\mathbf{x}$ if, and only if $\mathbf{x} \in \mathcal{B}^{\mathbf{u}}_{\mathbf{x}^*}$.

Monostability and multistability in $\mathbf{u}$. Given a set of states $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, a system that has a unique reachable attractor in $\mathbf{u}$ for all states is said to be monostable in $\mathbf{u}$ for this set, whereas a system that has multiple reachable attractors in $\mathbf{u}$ is said to be multistable in $\mathbf{u}$ for this set. More formally, f is said to be monostable in $\mathbf{u}$ for $\mathcal{X}$ if, and only if, there exists a unique attractor $\mathbf{x}^*$, such that $\forall \mathbf{x} \in \mathcal{X},\ \mathbf{x} \in \mathcal{B}^{\mathbf{u}}_{\mathbf{x}^*}$. On the contrary, f is said to be multistable in $\mathbf{u}$ is, and only if, there exists at least two attractors $\mathbf{x}_1^*$ and $\mathbf{x}_2^*$ such that $\mathbf{x}_1^* \neq \mathbf{x}_2^*$ and $\exists \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X},\ \mathbf{x}_1 \in \mathcal{B}^{\mathbf{u}}_{\mathbf{x}_1^*},\ \mathbf{x}_2 \in \mathcal{B}^{\mathbf{u}}_{\mathbf{x}_2^*}$.

Recurrent neural networks. Due to their temporal nature and update rules, RNNs can be seen as discrete-time non-linear dynamical systems. Formally, given a parameter vector θ, the system f is given by the update function of the RNN, such that $f(\mathbf{x}, \mathbf{u}) = f(\mathbf{x}, \mathbf{u}; \theta)$. Since attractors correspond to network steady states, they are thought to be the allowing factor for RNNs to retain information over a long period of time (Maheswaranathan et al., 2019; Pascanu et al., 2013; Sussillo & Barak, 2013).

4. Benchmarks

In this section, the different benchmarks are introduced. First, the supervised learning tasks are introduced, including long-term information restitution benchmarks in Section 4.1 and sequence classification benchmarks in Section 4.2. In Section 4.3, a reinforcement learning benchmark with partially observable environment is introduced. This environment contains long time dependencies.

4.1. Long-term information restitution benchmarks

The benchmarks introduced in this subsection contain long time dependencies, and therefore require networks able to remember relevant information for a long period. For those benchmarks, RNNs are trained on a dataset of 40 000 sample sequences and evaluated on a dataset of 40 000 sample sequences. During training, 20% of the training set is used as a validation set.

Copy first input benchmark. In this benchmark, the network is presented with a one-dimensional sequence of T timesteps $\mathbf{u}_{1:T}$, where $\mathbf{u}_t \sim \mathcal{N}(0, 1),\ t = 1, \dots, T$, and is tasked at approximating the target $\mathbf{y}_T = \mathbf{u}_1$. This benchmark thus consists of memorising the initial input for T timesteps. It allows one to measure the ability of recurrent architectures to bridge long time dependencies when the length T is large. Given the output $\mathbf{o}_T$ of the network, we seek to minimise the squared error $\mathcal{L}(\mathbf{o}_T, \mathbf{y}_T) = (\mathbf{o}_T - \mathbf{y}_T)^2$.

Denoising benchmark. In this benchmark, the network is presented with a two-dimensional sequence of T timesteps. The first dimension is a noised input stream $\mathbf{u}^1_{1:T}$, where $\mathbf{u}^1_t \sim \mathcal{N}(0, 1),\ t = 1, \dots, T$. Five timesteps of this stream should be remembered and outputted one by one by the network at timesteps $\{T - 4, \dots, T\}$. These five timesteps $\mathcal{S} = \{t_1, t_2, t_3, t_4, t_5\}$, with $t_1 < t_2 < t_3 < t_4 < t_5$, are sampled without replacement in $\{1, \dots, T - N\}$ with $N \geq 5$. N is a hyperparameter that allows one to tune how long the network should be able to retain the information at a minimum. The five timesteps are communicated to the network through the second dimension of the input $\mathbf{u}^2_{1:T}$, where $\mathbf{u}^2_t = 1$ if $t \in \mathcal{S}$, and $\mathbf{u}^2_t = 0$ otherwise, for $t = 1, \dots, T$. The target is thus given by $\mathbf{y}_{T-4:T} = [\mathbf{u}_{t_1}, \mathbf{u}_{t_2}, \mathbf{u}_{t_3}, \mathbf{u}_{t_4}, \mathbf{u}_{t_5}]$. Given the output sequence $\mathbf{o}_{T-4:T}$ of the network, we seek to minimise the mean squared error $\mathcal{L}(\mathbf{o}_{T-4:T}, \mathbf{y}_{T-4:T}) = \sum_{t=T-4}^{T} (\mathbf{o}_t - \mathbf{y}_t)^2$.

4.2. Sequence classification benchmarks

The benchmarks introduced in this subsection are sequence classification problems and therefore require networks able to use the information received in the sequence in order to infer the class. For those benchmarks, RNNs are trained on datasets derived from the usual train and test sets of the original MNIST dataset. During training, 20% of the training set is used as a validation set.

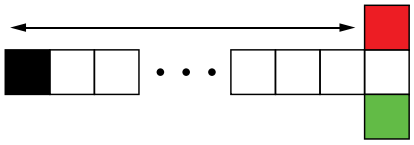


Fig. 1. T-Maze layout example, with the initial position of the agent in black, the treasure in green and the cell to avoid in red.

Permuted sequential MNIST. In this benchmark, the network is presented with the MNIST images, where pixels are presented to the network one by one as a sequence of length $T = 28 \times 28 = 784$. It differs from the regular sequential MNIST in that pixels are shuffled in a random order. Note that all images are shuffled according to the same random order.[2] The network is tasked at outputting a probability for each possible digit that could be represented in the initial image. This benchmark is known to be a more complex challenge than the regular one. Given the output $\mathbf{o}_T \in \mathbb{R}^{10}$ of the network and the true digit index $\mathbf{y}_T \in \{1, \dots, 10\}$, we seek to minimise the negative log likelihood loss $\mathcal{L}(\mathbf{o}_T, \mathbf{y}_T) = -\log(\mathbf{o}_T^{\mathbf{y}_T})$.

Permuted line-sequential MNIST. This benchmark is the same as the permuted sequential MNIST benchmark, except that the pixels are fed 28 by 28, which corresponds to one line of the permuted image.[4,2] The input dimension is thus 28 instead of one. N black lines are added at the end of the sequence such that the total length of the sequence is $T = 28 + N$. This has the effect of a forgetting period, such that any relevant information for predicting the class probabilities will be farther from the prediction timestep T.

4.3. Reinforcement learning benchmark

In reinforcement learning, the function approximators also process sequences as input when considering partially observable Markov decision processes (POMDPs). Indeed, in such environments, the optimal policies, as well as the value functions, are functions of the complete sequence of observations and past actions, called the history. In this work, we focus on the approximation of the history-action value function, or $\mathcal{Q}$-function, in order to derive a near-optimal policy in the considered POMDP. The deep recurrent Q-network (DRQN) algorithm is used to approximate this $\mathcal{Q}$-function with an RNN. From this approximation, we derive the fully greedy policy by taking the action that maximises the $\mathcal{Q}$-function for any given history. See Appendix B for the formal definition of POMDPs and their $\mathcal{Q}$-functions, and see Appendix C for the detailed DRQN algorithm.

The partially observable environment that is considered is the T-Maze environment (Bakker, 2001). The T-Maze is a POMDP where the agent is tasked with finding the treasure in a T-shaped maze (see Fig. 1). The state is given by the position of the agent in the maze and the maze layout that indicates whether the goal lies up or down after the crossroads. The initial state determines the maze layout, and it never changes afterwards. The initial observation made by the agent indicates the layout. Navigating in the maze provides zero reward, except when bouncing onto a wall, in which case a reward of -0.1 is received. While travelling along the maze, the agent only receives the information that it has not yet reached the junction. Once the junction reached, the agent is notified: it must now choose a direction depending on the past information it remembers. Finding the treasure provides a reward of 4. Passed the crossroads, the states are always terminal. The optimal policy thus consists of going through the maze, while remembering the initial observation in order to take the correct direction at the crossroads. This POMDP is parameterised by the corridor length $L \in \mathbb{N}$ that determines the number of timesteps for which the agent should remember the initial observation. The discount factor is $\gamma = 0.98$. This POMDP is formally defined in Appendix D.

[2] The permutation is given by: `np.random.seed(42); np.random.permutation(28*28);` (NumPy 1.23.2).

5. Correlating multistability and learning

This section aims at showing the correlation that exists between multistability properties of RNNs and their ability to learn long time dependencies. To this end, in Section 5.1 we first introduce the VAA, a measure of the number of basins of attraction that are spanned by a set of states. In Section 5.2, we show how to estimate the multistability of an RNN using VAA by estimating the number of reachable attractors for a set of states resulting from the input sequences. We then carry out a number of experiments in Section 5.3 to show the correlation between multistability and learning with different types of RNN on the benchmarks previously introduced.

5.1. Variability amongst attractors

One way to quantify the multistability in $\mathbf{u}$ of a system for a set of states $\mathcal{X}$ is to count the number of different attractors that can be reached starting from those states. The VAA of a system f for a set of initial states $\mathcal{X}$ and an input $\mathbf{u}$ is defined as

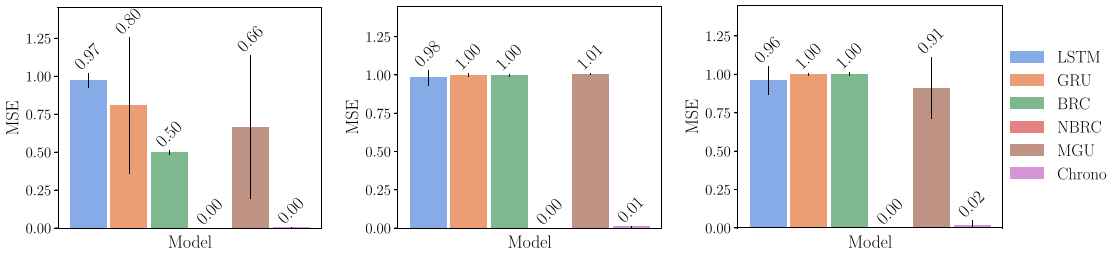
$$\mathrm{VAA}(f, \mathcal{X}, \mathbf{u}) = \frac{1}{|\mathcal{X}|} \sum_{i=1}^{|\mathcal{X}|} \frac{1}{\sum_{j=1}^{|\mathcal{X}|} \delta\left(\limsup_{n\to\infty} \|f^n(\mathbf{x}_i, \mathbf{u}) - f^n(\mathbf{x}_j, \mathbf{u})\| = 0\right)} \tag{4}$$

where $\delta(x)$ is the Kronecker delta function that returns 1 when condition x is met, and 0 otherwise. It can be noted that this definition does not exclude limit cycles and considers states that are on the same limit cycle but far from each others as different attractors. This is a limitation that we discuss in our conclusion. In the following, we make the hypothesis that such limit cycles are not encountered in practice.

The denominator of Eq. (4) gives the number of states in $\mathcal{X}$ that converge towards the same attractor as $\mathbf{x}_i$. The sum of this fraction over all states that converge towards a given attractor is thus equal to one, such that the sum of this fraction over all states gives the number of different attractors. $\mathrm{VAA}(f, \mathcal{X}, \mathbf{u})$ is thus equal to the number of different attractors in $\mathbf{u}$ reached from the initial states contained in $\mathcal{X}$ divided by the number of initial states $|\mathcal{X}|$. Its maximal value is thus 1, when all reached attractors are different, and its minimal value is $\frac{1}{|\mathcal{X}|}$, when all the states have converged towards the same attractor (i.e., the system is monostable).

In practice, since it is impossible to evaluate the limits to infinity in the VAA, we fix a finite number of timesteps M for state convergence, called the stabilisation period. As a consequence, the system may not have completely converged towards the attractor after this period. We thus define a tolerance ε below which two final states are considered to correspond to the same attractor. This truncated VAA is written as

$$\mathrm{VAA}_{M,\varepsilon}(f, \mathcal{X}, \mathbf{u}) = \frac{1}{|\mathcal{X}|} \sum_{i=1}^{|\mathcal{X}|} \frac{1}{\sum_{j=1}^{|\mathcal{X}|} \delta\left(\left\|f^M(\mathbf{x}_i, \mathbf{u}) - f^M(\mathbf{x}_j, \mathbf{u})\right\| \le \varepsilon\right)}. \tag{5}$$



(a) $T = 50$ (b) $T = 300$ (c) $T = 600$

Fig. 2. Test MSE loss for the copy first input benchmark with different sequence lengths T. Mean and standard deviation are reported after 50 epochs.

5.2. Estimating the multistability of an RNN for a set of input sequences

RNNs can exhibit a long-lasting memory through multistability in their hidden states (Vecoven et al., 2021). Indeed, having multiple attractors that are reachable from different input sequences probably allows one to encode information about these sequences over the long term. We propose estimating the multistability of an RNN for a set of input sequences by computing the number of different reachable attractors for hidden states resulting from different input sequences. More precisely, we propose to compute $\text{VAA}(f, \mathcal{X}, \mathbf{u})$ for hidden states $\mathcal{X}$ sampled from different input sequences. In practice, it is not feasible to estimate the VAA for all hidden states resulting from the set of input sequences. Indeed, computing the VAA is quadratic in the number of hidden states because of the pairwise distances. We thus propose to estimate the VAA by averaging its value over several small batches of hidden states sampled at random time steps in different sequences sampled from the set of input sequences. Moreover, we still have to choose the stable input $\mathbf{u}$ according to which we want to measure the multistability in $\mathbf{u}$. In order to measure the multistability of the network for a wide range of stable inputs, we propose to measure the multistability on average for several inputs sampled according to a standard normal distribution. Note that for each batch of hidden states, a unique $\mathbf{u} \sim \mathcal{N}(0, \mathbf{1})$ is sampled and kept constant during the convergence period of M timesteps. The resulting procedure for estimating the multistability of an RNN for a set of input sequences is given in Algorithm 1.

Algorithm 1: Estimating the proportion of reachable attractors of an RNN for a set of input sequences

Parameters: $I \in \mathbb{N}$ the number of iterations to compute the mean of the VAA.
$M \in \mathbb{N}$ the stabilisation period.
$\varepsilon \in \mathbb{R}^+$ tolerance when considering state similarity.
$\theta \in \mathbb{R}^{d_\theta}$ the parameters of the network.
Data: $\mathcal{D} = \{\mathbf{u}^1_{1:T_1}, \dots, \mathbf{u}^N_{1:T_n}\}$ a set of N input sequences.

Let $f \leftarrow f(\cdot, \cdot;\ \theta)$ the dynamical system.
Initialise mean value $\overline{\text{VAA}} \leftarrow 0$.
for $i = 1, \dots, I$ **do**
  Sample a batch of input sequences $\mathcal{B} \sim \mathcal{D}$.
  Sample a random hidden state in each input sequence $\mathcal{X} \leftarrow \texttt{RandomHiddenStates}(\mathcal{B}, \theta)$.
  Sample $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$
  $\overline{\text{VAA}} \leftarrow \overline{\text{VAA}} + \frac{1}{I}\,\text{VAA}_{M,\varepsilon}(f, \mathcal{X}, \mathbf{u})$
return $\overline{VAA}$

Algorithm 2: Random Hidden States

Parameters: $\theta \in \mathbb{R}^{d_\theta}$ the parameters of the network.
Data: $\mathcal{B} = \{\mathbf{u}^1_{1:T_1}, \dots, \mathbf{u}^n_{1:T_n}\}$ a batch of n input sequence sampled in the training set.

$\mathcal{X} \leftarrow \{\}$
foreach $\mathbf{u}^i_{1:T_i} \in \mathcal{B}$ **do**
  Sample a timestep $t \sim \mathcal{U}(\{1, \dots, T_i\})$
  Set $\mathbf{x}^i_0 = h(\theta)$ where h is the RNN's initialisation function
  for $k = 1, \dots, t$ **do**
    Set $\mathbf{x}^i_k = f(\mathbf{x}^i_{k-1}, \mathbf{u}^i_k;\ \theta)$ where f is the RNN's update function.
  Update $\mathcal{X} \leftarrow \mathcal{X} \bigcup \{\mathbf{x}^i_t\}$
return $\mathcal{X}$

5.3. Experiments

In this subsection, we observe how the multistability of RNNs evolves when they are trained on the long-term information restitution and reinforcement learning benchmarks introduced in Section 4. The multistability of these networks is estimated throughout the training procedure, using Algorithm 1. For the copy first input benchmark, networks are made up of one 128 neurons recurrent layer. For the other benchmarks, networks are made up of two recurrent layers, each of 256 neurons. All averages and standard deviations reported were computed over five different training sessions. Training was done using the Adam optimiser (Kingma & Ba, 2014) with a learning rate of 1×10^{-3} and a batch size of 32. All hyperparameters have been chosen a priori to standard values and are kept fixed. The goal here is not to measure the best performance of each architecture but rather to study, for a given architecture and optimisation procedure, whether there is a link between learning and multistability for different benchmarks. In Appendix E.1, we show that those results also hold with other hyperparameters for the copy first input benchmark. In all experiments, the multistability is estimated with $M = 10\,000$, $\varepsilon = 1 \times 10^4$, and $I = 10$.

Copy first input benchmark. Fig. 2 shows the performance of the different cells on this benchmark for different sequence lengths $T \in \{50, 300, 600\}$. The best-performing cell is the NBRC, whose performance is not affected by the length of the sequences. In comparison, the classical cells, MGU, LSTM and GRU, struggle to decrease their losses. Surprisingly, the BRC, a bistable cell, does not succeed in decreasing its loss. Generally speaking, the longer the sequences are, the worse their performances are. The last cell, the chrono-initialised LSTM, competes with the NBRC with its hyperparameter T_{max} chosen to 600. Fig. 3 illustrates the correlation between the VAA and the validation loss for the LSTM and CHRONO cells. The LSTM cell, whose VAA increases late and little, fails to learn. On the other hand, the chrono initialised LSTM cell

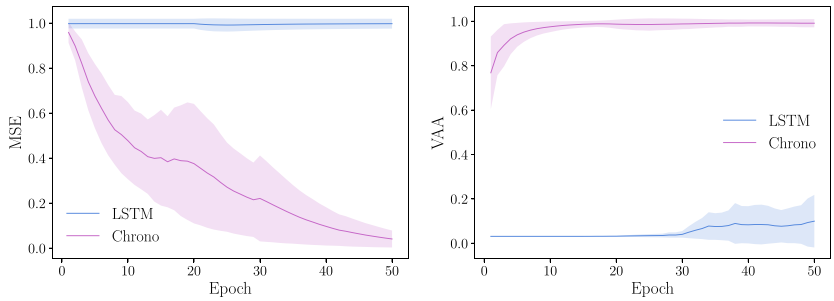


Fig. 3. Evolution of the validation loss (left) and of the VAA (right) of LSTM networks, with and without chrono initialisation, for the copy first input benchmark with $T = 50$. Mean and standard deviation are reported after 50 epochs.

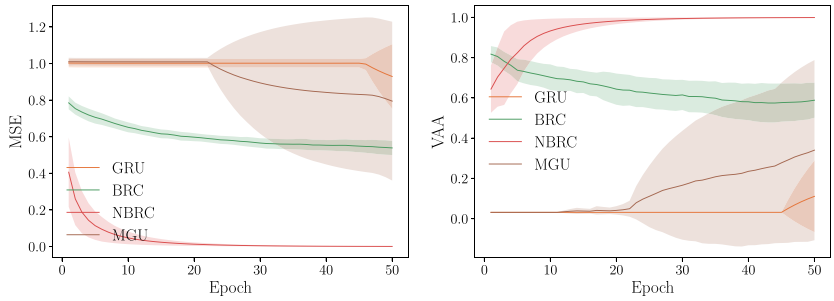
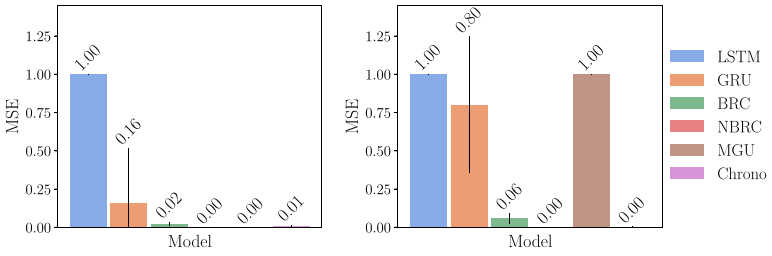


Fig. 4. Evolution of the validation loss (left) and of the VAA (right) of GRU, MGU, BRC and NBRC networks, for the copy first input benchmark with $T = 50$. Mean and standard deviation are reported after 50 epochs.



(a) $N = 5$ (b) $N = 100$

Fig. 5. Test MSE loss for the denoising benchmark with different forgetting periods N and $T = 200$. Mean and standard deviation are reported after 50 epochs.

sees its loss decreasing while its VAA increases. This figure also shows that the chrono initialisation promotes the learning of long time dependencies through multistability. Fig. 4 illustrates the correlation between the VAA and the validation loss for the other cells. It is clear from this figure that the bistability mechanism introduced in the BRC and NBRC cells also promote multistability. Moreover, as for the LSTMs and chrono-initialised LSTMs, the loss only starts decreasing when the VAA increases.

Denoising benchmark. Fig. 5 shows the performance of the different cells on this benchmark for different forgetting periods $N \in \{5, 100\}$. Once again, the NBRC has the best performance, closely followed by the chrono-initialised LSTM. On this benchmark, the BRC also reaches a very low loss. Once again, we can see that all classical cells (LSTM, GRU, and MGU) generally fail in learning when longer time dependencies are present ($N = 100$). Fig. 6 shows the evolution of the VAA and the validation loss of multiple LSTM cells, with and without chrono initialisation, during the training on this benchmark. As for the previous benchmark, only the chrono-initialised LSTMs have a high VAA and efficiently decrease their loss. It can be noted that classically initialised LSTMs have a VAA close to zero throughout the learning on this harder benchmark. Fig. 7 shows these results for the GRU, BRC, NBRC and MGU cells. It is observed that the GRU network has a very low VAA, and learning does not start before its VAA increases. The MGU network does not manage to learn on this benchmark while its VAA only slowly increases at the end of the training procedure. As far as the two bistable networks are concerned, their VAA is directly maximised and learning starts directly, indicating that those indeed promote the learning of long time dependencies through multistability. Finally, Fig. 8 shows the validation loss and the VAA of five different trainings of the GRU cell on the denoising benchmark with $N = 5$. It is clear that the GRU cell only starts decreasing its loss when its VAA has started increasing. This proves once more the correlation between the VAA and the learning on long-term information restitution benchmarks.

T-Maze benchmark. In this reinforcement learning setting, a policy is derived from the approximation of the $\mathcal{Q}$-function. The hyperparameters of the DRQN algorithm used for approximating the $\mathcal{Q}$-function are given in Appendix C. On the left in Fig. 9, we can see the mean non-discounted cumulative reward obtained by the policies derived from GRU cells approximating the $\mathcal{Q}$-function. On the right in Fig. 9, we can see the VAA of these cells estimated with Algorithm 1 using the histories of the replay buffer as input sequences. Those value are clearly correlated. Indeed, the better the agent plays, the higher its VAA is.

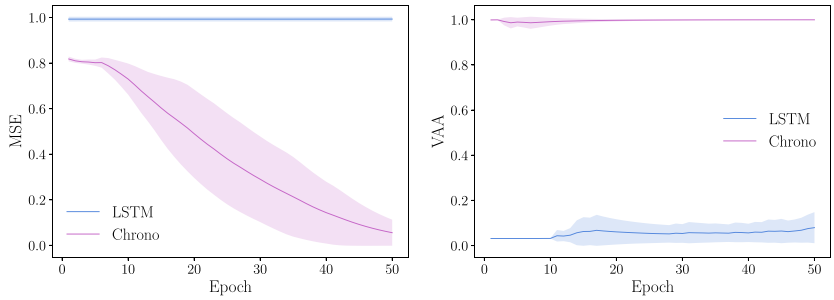


Fig. 6. Evolution of the validation loss (left) and of the VAA (right) of LSTM networks, with and without chrono initialisation, for the denoising benchmark with $N = 100$ and $T = 200$. Mean and standard deviation are reported after 50 epochs.

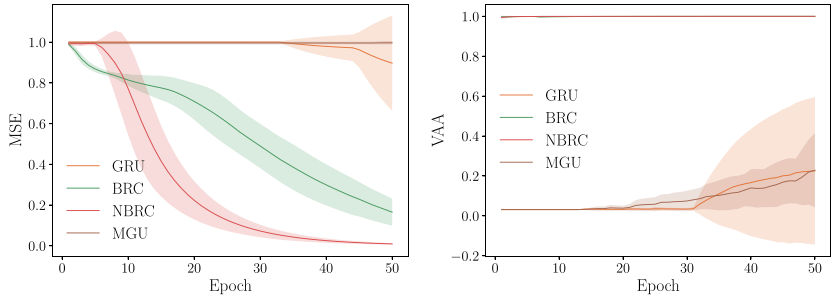


Fig. 7. Evolution of the validation loss (left) and of the VAA (right) of GRU, MGU, BRC and NBRC networks, for the denoising benchmark with $N = 100$ and $T = 200$. Mean and standard deviation are reported after 50 epochs.

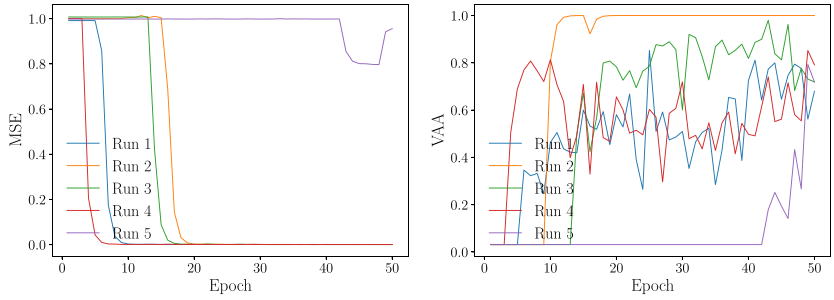


Fig. 8. Evolution of the validation loss (left) and of the VAA (right) of multiple GRU networks, for the denoising benchmark with $N = 5$ and $T = 200$. Mean and standard deviation are reported after 50 epochs. Loss decrease only start when the network becomes multistable (VAA greater than $\frac{1}{|\mathcal{X}|}$).

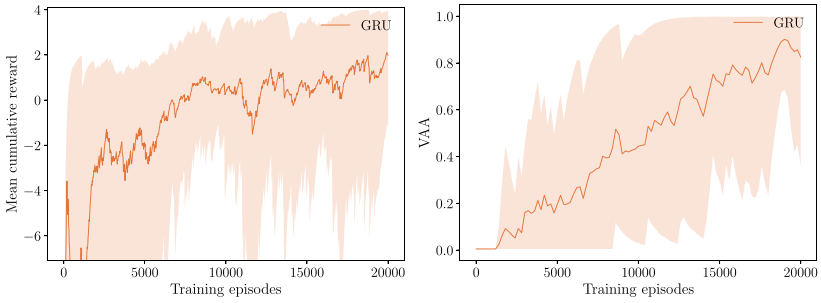


Fig. 9. Evolution of the mean cumulative reward (left) and their VAA (right) obtained by GRU agents during DRQN training on a T-Maze of length 200. Mean and standard deviation are estimated over 3 training sessions.

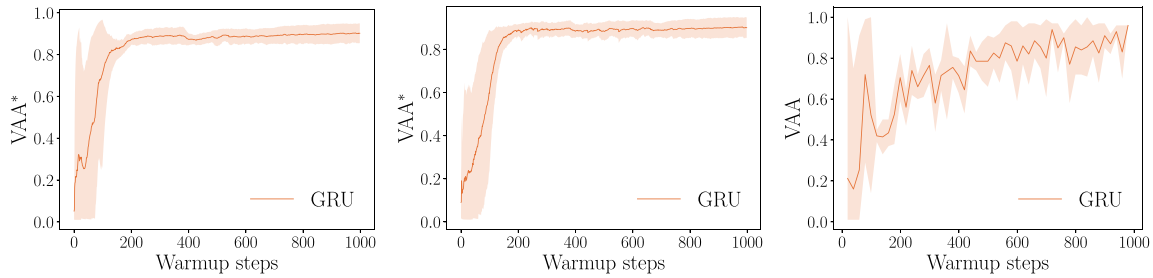


Fig. 10. Evolution of the VAA* for a two-layer GRU (left and middle) and of the VAA of the network (right) during warmup. This network is warmed up on the denoising dataset and results were averaged over three runs.

6. Fostering multistability at initialisation

In Section 6.1, we describe the warmup initialisation procedure that allows one to maximise the estimated multistability of a network for a dataset of input sequences. Then, in Section 6.2, we compare classic cells to warmed-up cells on information restitution, sequence classification, and RL benchmarks and show the benefits of the warmup in tasks with long time dependencies, when considering the same standard hyperparameters as in the previous section. However, we also show that the warmup procedure does not improve the results in the sequence classification tasks. In Section 6.3, we introduce the double-layer architecture, that has both multistable and transient dynamics. We show that this architecture reaches a better performance both on information restitution and sequence classification benchmarks. Finally, in Section 6.4, we show that the advantage of the warmup and the double-layer architecture, shown for standard hyperparameters in Section 6.2 and Section 6.3, also holds when optimising the hyperparameters for each cell version (number of recurrent layers L, number of hidden units H, batch size B, learning rate α).

6.1. Warming up RNNs

The previous observations, that show a correlation between the multistability of a network and its ability to learn long time dependencies, suggest that fostering multistability could ease learning in this case. In order to promote the multistability of a network, we propose maximising the number of reachable attractors for hidden states resulting from the set of input sequences. As for the estimation of the multistability, computing the VAA for all hidden states is not feasible because of its quadratic complexity. In practice, we propose using stochastic gradient descend (SGD) to maximise the number of reachable attractors for batches of hidden states from different input sequences. As for the estimation of the multistability, we sample a different stable input $\mathbf{u} \sim \mathcal{N}(0, \mathbf{1})$ for each batch of hidden states. We note however that SGD cannot be used directly on the estimation of the proportion of reachable attractors detailed in Algorithm 1, for two different reasons. First, the VAA and the $\text{VAA}_{M,\varepsilon}$ are not differentiable because of the Kronecker delta, which prevents from computing the gradient. Second, it is likely that hidden states convergence is slow when several RNNs are stacked. Indeed, the first layers must have reached stability for the following one to receive a stable input.

In order to solve the first problem, we introduce a differentiable proxy $\text{VAA}^*_{M,\varepsilon}$ of the $\text{VAA}_{M,\varepsilon}$. Instead of the denominator

$$C_{i,j} = \delta\left(\left\|f^M(\mathbf{x}_i, \mathbf{u}) - f^M(\mathbf{x}_j, \mathbf{u})\right\| \leq \varepsilon\right), \tag{6}$$

that is equal to 1 when the final states after truncated convergence are close enough, we use

$$C^*_{i,j} = 1 - \frac{\max(0, \left\|\tanh f^M(\mathbf{x}_i, \mathbf{u}) - \tanh f^M(\mathbf{x}_j, \mathbf{u})\right\| - \varepsilon)}{\left\|\tanh f^M(\mathbf{x}_i, \mathbf{u}) - \tanh f^M(\mathbf{x}_j, \mathbf{u})\right\|}. \tag{7}$$

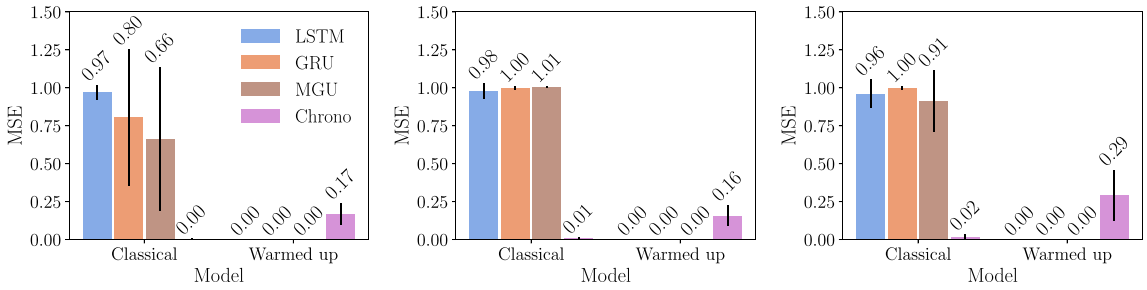
We note that the value of $C^*_{i,j}$ is strictly equal to 1 if $f^M(\mathbf{x}_i, u)$ is close enough in Euclidean distance to $f^M(\mathbf{x}_j, u)$. On the other hand, $C^*_{i,j}$ will be close to 0 when they are far away. We also note that $C^*_{i,j}$ will never be strictly equal to 0, but will get closer as the distance increases, since the fraction tends towards 1. It can be noted that we are not interested in states being far apart from each other, but just in them being different. However, we noticed in the experiments that this small bias provides a good direction for the gradient in order to reach multistability. For this same reason, we need to apply a saturating function (hyperbolic tangent in this case) to the states in order to avoid extreme states when maximising VAA*. The resulting differentiable proxy of the VAA is given by

$$\text{VAA}^*_{M,\varepsilon}(f, \mathcal{X}, \mathbf{u}) = \frac{1}{|\mathcal{X}|}\sum_{i=1}^{|\mathcal{X}|}\frac{1}{\sum_{j=1}^{|\mathcal{X}|} 1 - \frac{\max(0, \|\tanh f^M(\mathbf{x}_i,\mathbf{u}) - \tanh f^M(\mathbf{x}_j,\mathbf{u})\| - \varepsilon)}{\|\tanh f^M(\mathbf{x}_i,\mathbf{u}) - \tanh f^M(\mathbf{x}_j,\mathbf{u})\|}}. \tag{8}$$

For maximising the multistability of an RNN for a given dataset of input sequences, we thus propose to maximise by SGD the VAA* of batches of hidden states resulting from different input sequences, at random time steps. For each batch of hidden states, a constant input perturbation is randomly sampled from $\mathbf{u} \sim \mathcal{N}(0, \mathbf{1})$ in order to stabilise the RNN hidden states over M time steps. However, as can be seen from Eq. (8), maximising the VAA* only occurs when all hidden states are infinitely distant, which is not desirable for learning efficiently. In practice, we thus use SGD to get the VAA* of each layer as close as possible to $k = 0.95$, as this proved empirically to maximise the number of attractors (see Fig. 10) while avoiding too extreme states that could arise from the approximation of the VAA with C^*. In Appendix E.3, we show on the copy first input benchmark with $T \in \{50, 300, 600\}$ that the warmup procedure improves learning for a wide range of k. It shows the robustness of our findings with respect to some hyperparameter variation. The loss used is thus given by

$$\mathcal{L}(\mathbf{v}, k) = \frac{1}{L}\sum_{i=1}^{L}(v_i - k)^2 \tag{9}$$

where $v_i = \text{VAA}^*_{M,\varepsilon}(f^i, \mathcal{X}, \mathbf{u})$ is the estimated multistability of layer i and L is the number of layers in the RNN. Maximising the VAA* of each layer separately allows one to tackle the problem of layer convergence as identified above. To avoid over-fitting problems, M is sampled uniformly in $\{1, \ldots, M_{\max}(s)\}$ at gradient step s, where $M_{\max}(s) = \min(M^*, 1+c\cdot s)$ with M^* the maximum stabilisation period and c the stabilisation period increment. This progressive increase is required for reaching multistability smoothly, avoiding gradients problems. For the supervised learning tasks, the batches of input sequences are sampled in the training set. For the reinforcement learning tasks, batches of input sequences are sampled from the exploration policy. Algorithm details the whole warmup procedure for a dataset $\mathcal{D}$ of input sequences.



(a) $T = 50$ (b) $T = 300$ (c) $T = 600$

Fig. 11. Test MSE loss for the copy first input benchmark with different sequence lengths T. Mean and standard deviation are reported after 50 epochs.

Algorithm 3: Warming up an RNN

Parameters: $S \in \mathbb{N}$ the number of gradient steps.
$n \in \mathbb{N}$ the batch size.
$\alpha \in \mathbb{R}^+$ the learning rate.
$k \in [0, 1]$ the target average $\text{VAA}^*_{M,\varepsilon}$.
$M^* \in \mathbb{N}$ the maximum stabilisation period.
$c \in \mathbb{N}$ the stabilisation period increment.
$\varepsilon \in \mathbb{R}^+$ tolerance when considering state similarity.
$\theta \in \mathbb{R}^{d_\theta}$ the parameters of the network.
L the number of layers in the RNN.
Data: $\mathcal{D} = \{\mathbf{u}^1_{1:T_1}, \dots, \mathbf{u}^N_{1:T_N}\}$ a training set of N input sequences.

for $s = 1, \dots, S$ **do**
  Sample a batch $\mathcal{B}$ of n sequences in $\mathcal{D}$ without replacement $\mathcal{B} \sim \mathcal{U}^n(\mathcal{D})$.
  Sample a random hidden state in each input sequence $\mathcal{X} \leftarrow \texttt{RandomHiddenStates}(\mathcal{B}, \theta)$.
  Sample $M \sim \mathcal{U}(\{1, \dots, \min(M, 1 + s \cdot c)\})$.
  for $i = 1, \dots, L$ **do**
    Sample $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$.
    Set $v_i = \text{VAA}^*_{M,\varepsilon}(f^i, \mathcal{X}, \mathbf{u})$ where f^i is the update function of the i^{th} RNN layer.
  Compute loss $L \leftarrow \mathcal{L}(\mathbf{v}, k)$ where $\mathbf{v} = \begin{pmatrix} v_1 & \cdots & v_L \end{pmatrix}$.
  Compute gradient $g \leftarrow \nabla_\theta L$ with BPTT (over stabilisation period and input sequence).
  Update parameters $\theta \leftarrow \theta - \alpha g$.
  Update maximum stabilisation period $M^* \leftarrow M^* + c$.

We show in Fig. 10 that the warmup procedure effectively increases the VAA* of each layer in an RNN. Furthermore, we can also see on the right in Fig. 10 that as the warmup procedure is carried out, the true VAA measure of the RNN is increasing as well, even reaching 1 as the warmup procedure ends.

6.2. Experiments

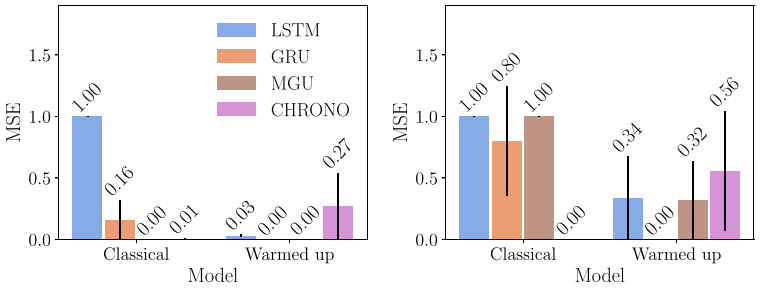
To demonstrate the impact of warming up RNNs on information restitution tasks, sequence classification tasks, and in partially observable RL environment, we tackle all benchmarks introduced in Section 4. We train the LSTM, GRU and MGU cells with and without warmup and show that their performance is greatly improved with warmup. As chrono-initialised LSTMs are known to work well, we also compare our results to such cells, with and without warmup. The hyperparameters have been chosen to the same values as in previous section. The goal here is not to measure the best performance of each architecture with or without warmup but rather to measure, for a given architecture and optimisation procedure with fixed hyperparameters, whether the warmup initialisation procedure provides a better learning for different benchmarks. In Appendix E.2, we show that those results also hold for other hyperparameters for the permuted row sequential MNIST benchmark. In addition, in Section 6.4, we compare the performance of all cells with and without warmup with optimised hyperparameters. All averages and standard deviations reported were computed over three different training sessions. The optimal parameters for warming up can vary depending on architectures and needs, but we found $\alpha = 1e^{-2}$, $c = 10$, $S = 100$, $n = 200$ and $M^* = 200$ to be a good choice.

Copy first input benchmark. As can be seen from Fig. 11, warming up RNNs greatly improves performances in the copy first input benchmark, for any sequence length $T \in \{50, 300, 600\}$. Indeed, classically initialised RNNs have an average loss above 0.500 after 50 epochs, while all warmed-up RNNs have an average loss below 0.001 after 50 epochs. On the other hand, the chrono-initialised LSTM performs better when it is not warmed up. Even if the chrono-initialisation and the warmup both promote the learning of long-term dependencies, combining them seems to have the opposite effect, leading to less performant model.

Denoising benchmark. As far as the denoising benchmark is concerned, Fig. 12 shows that warmed-up cells always perform better than classically initialised ones, on sequences of length $T = 200$. However, it can be noted that the average loss is still quite significant after 50 epochs for the LSTM and MGU cells, in the case of a forgetting period of $N = 100$. As for the copy first input benchmark, the chrono-initialised cells perform worse when warmed-up which suggests once again that the chrono initialisation interacts disadvantageously with the warmup procedure.

T-Maze benchmark. On the left in Fig. 13, we can see the evolution of the expected cumulative reward of the DRQN policy for the T-Maze environment as a function of the number of episodes of interaction. It is more than clear that all warmed-up cells and bistable cells (i.e., BRC and NBRC), are better than the classically initialised ones on this RL benchmark. As for the other benchmarks, the chrono-initialised LSTMs seem to interact disadvantageously with the warmup procedure. In any case, it can be noted that the chrono-initialised LSTMs are always among the worse cells for this benchmark, with and without warmup. Furthermore, we can see that warming up cells improves their performance even more as the length of the T-Maze increases, suggesting that the warmup procedure and the multistability of an RNN indeed help to tackle tasks with long time dependencies. On the right in Fig. 13, we can see the number of episodes required to reach the optimal policy for each cell. It is clear that warming up a cell speeds up the convergence towards the optimal policy when time dependencies become large. Indeed, for $L = 200$, all warmed-up cells reach the optimal policy before any classically initialised cell, except for the chrono-initialised LSTM.

Permuted sequential MNIST. In Fig. 14, we can see the test accuracies after 70 epochs on the permuted sequential MNIST benchmark. It is clear that the warmup initialisation does not help in this task. For the LSTM and GRU, the warmed-up cells are even worse than the classic cells. This confirms that some tasks such as this sequence classification benchmark needs more transient dynamics instead of multistable ones.



(a) $N = 5$ (b) $N = 100$

Fig. 12. Test MSE loss for the denoising benchmark with different forgetting periods N and $T = 200$. Mean and standard deviation are reported after 50 epochs.

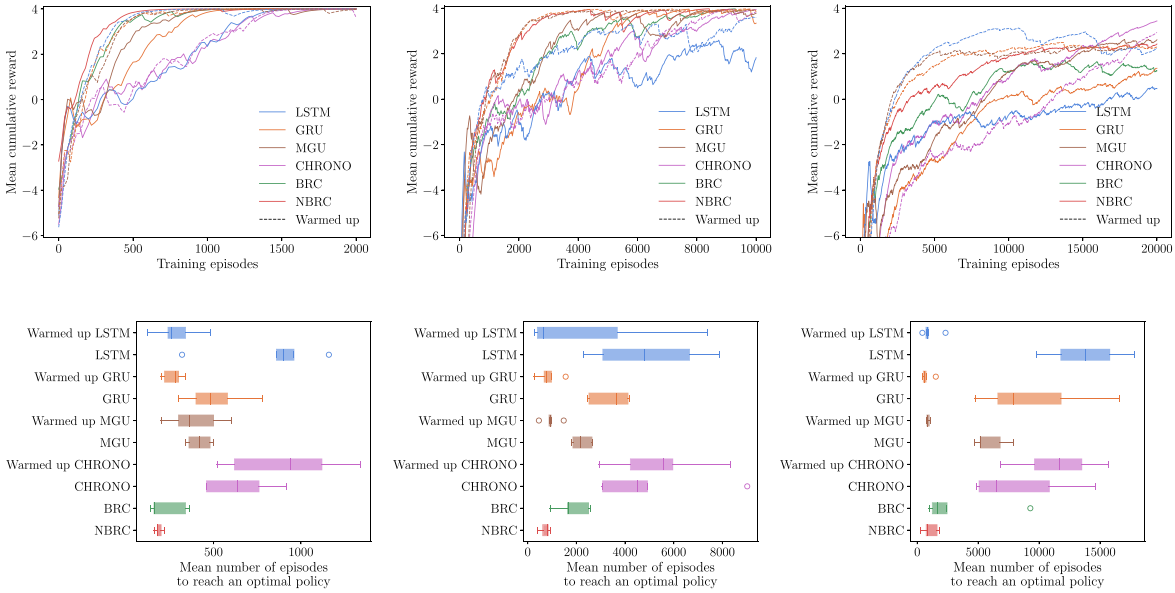


Fig. 13. Evolution of the mean cumulative reward obtained by warmed-up and classic agents during their training (up) and mean number of episodes required to reach the optimal policy (down) on T-Mazes of length 20 (left), 100 (centre) and 200 (right).

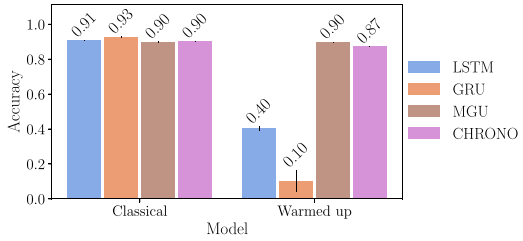


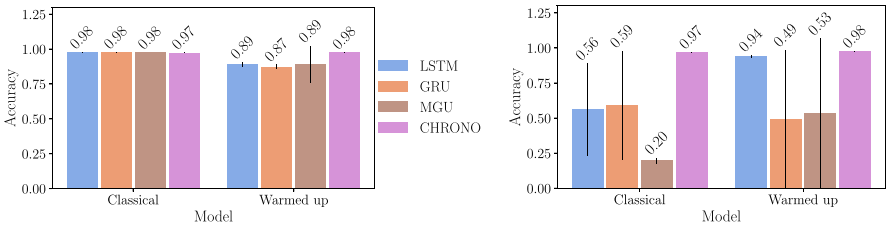
Fig. 14. Test accuracy for the permuted sequential MNIST benchmark. Mean and standard deviation are reported after 70 epochs.

Permuted line-sequential MNIST. In Fig. 15, we can see the accuracies of each cell after 70 epochs on the test set of the permuted line-sequential MNIST benchmark. For a sequence length of 100 (i.e., $N = 72$), it is clear that the classically initialised cells are better at this task. As for the permuted sequential MNIST, this shows that transient dynamics are important for those sequence classification tasks, as opposed to information restitution tasks.

6.3. Recurrent double-layers

As shown in the previous section and mentioned in the literature (Sussillo & Barak, 2013), the importance of the transient dynamics of RNNs should not be neglected for prediction. Indeed, it is easy to see why transient dynamics can be of importance when trying to tackle a regression task. If information is only stored in the form of attractors, then there can only be a limited number of states the network can take, making it very hard to get precise predictions. We observe that when warming up neural networks they tend to lose predictive accuracy, at the benefit of easier training on longer sequences. This leads one to think that RNNs should be built to have both rich transient and multistable dynamics. We thus propose using a double-layer architecture that allows one to get precise predictions while maintaining the benefits of warmup. We simply split each recurrent layer in two equal parts and only warmup one of them. In this double architecture, the hidden states sizes are divided by two compared to the simple architecture, for a fairer comparison. This allows to endow some part of each layer with multistability, while the other remains monostable with richer transient dynamics. A double-layer structure is depicted in Fig. 16.

As can be seen from Fig. 17, Fig. 18, Fig. 19 and Fig. 20, the double-layer architecture is always among the best performing architecture, for all four supervised learning benchmarks and for the LSTM, GRU and MGU cells, when using the same standard hyperparameters of the previous sections. Even the chrono-initialised LSTMs perform well with the double-layer architecture except on the copy first input benchmark. It shows that the



(a) $N = 72$ (b) $N = 472$

Fig. 15. Test accuracy for the permuted line-sequential MNIST benchmark for different forgetting periods N. Mean and standard deviation are reported after 70 epochs. We note that when N equals 72 (472) the resulting image has 100 (500) lines.

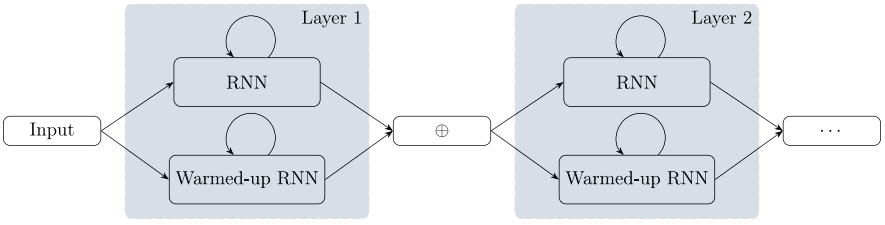
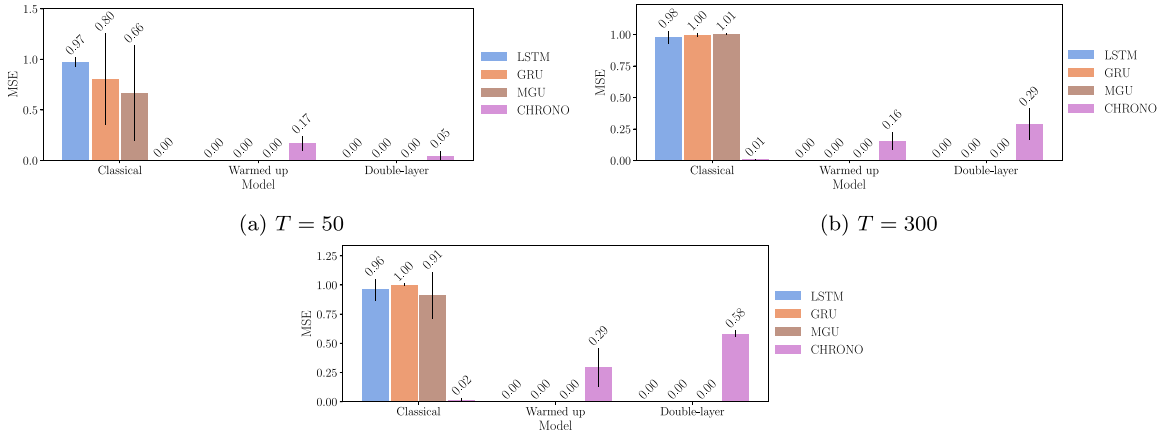


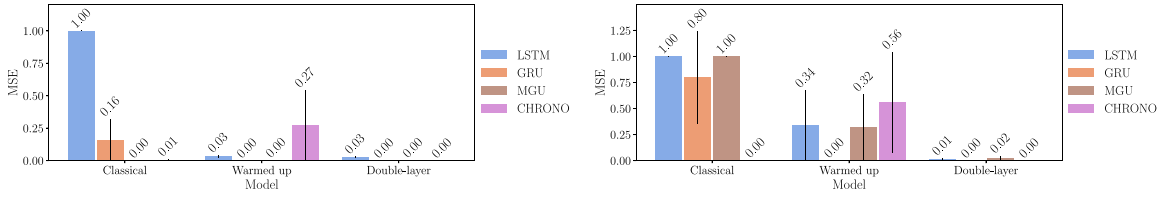
Fig. 16. Double layer architecture.



(a) $T = 50$ (b) $T = 300$

(c) $T = 600$

Fig. 17. Test MSE loss for the copy first input benchmark with different sequence lengths T. Mean and standard deviation are reported after 50 epochs.



(a) $N = 5$ (b) $N = 100$

Fig. 18. Test MSE loss for the denoising benchmark with different forgetting periods N and $T = 200$. Mean and standard deviation are reported after 50 epochs.

double-layer architecture combines both the transient and multistable features of an RNN. In addition, we can see in Fig. 20 that the double-layer architecture is significantly better than the other architecture, for all types of cell, on the permuted line-sequential MNIST benchmark with a forgetting length of $N = 472$, a problem that requires both transient and multistable dynamics. In addition, we show in Appendix G that the double-layer architectures without partial warmup generally perform worse that the classic architectures. This ensures that the partial warmup is the most important factor for the performance of the double-layer architecture. In Fig. 21, we can visualise the evolution of the validation loss averaged over 5 training sessions on the denoising benchmark for the LSTM, GRU and MGU cells, with the three architectures (i.e., classic, warmed up and double). It is clear that the warmed-up and double-layer architectures are better. Additionally, we can see that the double-layer architecture is significantly faster at learning this task for the GRU and MGU cells.

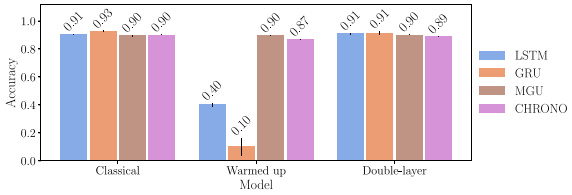


Fig. 19. Test accuracy for the permuted sequential MNIST benchmark. Mean and standard deviation are reported after 70 epochs.

6.4. Hyperparameter optimisation

In this section, we study the performance of the different cells in their different versions (i.e., classic, warmed up and double), when the hyperparameters are optimised. In Section 6 and Appendix E, we have shown that the warmup procedure and double-layer architecture provides a nice improvement in performance for a wide range of hyperparameters. Here, we consider a more practical setting in which the hyperparameters of a considered cell version can be optimised according to the learning set. We consider a standard hyperparameter selection procedure where the hyperparameters are selected according to the loss on a selection set, averaged over 5 training sessions (see Appendix F for details). Those hyperparameters are then selected for 5 training sessions according to the standard procedure, and the average loss on the test set is reported. Due to the computational cost of such an optimisation procedure, we only consider the most challenging benchmarks of each category, that is the denoising benchmark with $N = 100$ and the permuted line-sequential MNIST benchmark with $N = 472$.

The best hyperparameters are reported in Appendix F for both benchmarks. The test losses obtained using those hyperparameters are given in Fig. 22. As can be seen by putting Fig. 22(a) in perspective with Fig. 18(b), the hyperparameter selection allows all cell versions to reach a lower test MSE for the denoising benchmark. Similarly, by putting Fig. 22(b) in perspective with Fig. 20(b), it can be seen that all cell versions reach a higher test accuracy for the MNIST benchmark, when the hyperparameters have been optimised. Fig. 23 shows the evolution of the validation losses throughout the training procedure for the best hyperparameters of each cell version, averaged over the 5 training sessions, for the denoising benchmark with $N = 100$. It can be seen that the warmup procedure and the double cell architecture still provide a significant advantage in term of convergence speed and final performance. Fig. 24 shows the evolution of the validation losses throughout the training procedure using the best hyperparameters, for the line-sequential MNIST benchmark with $N = 472$. As for the denoising benchmark, the warmup and the double layer architecture still provide a very significant improvement in term of convergence speed.

7. Conclusion

In this work, we introduced a new initialisation procedure, called warmup, that improve the ability of recurrent neural networks to learn long time dependencies. This procedure is motivated by recent work that showed the importance of fixed points and attractors for the prediction process of trained RNNs. More precisely, we introduced a lightweight measure called VAA, that can be optimised at initialisation in few gradient steps to endow RNNs with multistable dynamics. Warmup can be used with any type of recurrent cell and we show that it vastly improves their performance on problems with long time dependencies. In addition, we introduced a new architecture that combines transient and multistable dynamics through partial warmup. This architecture was shown to reach a better performance than both classic and warmed-up cells on several tasks, including information restitution and sequence classifications tasks.

This work also motivates several future works. First, it can be noted that the double-layer architecture might be worth exploring with different types of cell. We showed here that there are benefits of using different types of initialisation for the same type of cell. This might hint at the possibility of having similar benefits when combining different types of cell that have different dynamical properties in a single recurrent neural network. Furthermore, in this paper we have aimed at maximising the number of attractors through warmup before training. We noticed however that in some rare cases, networks loose multistability properties when training. Using VAA as a regularisation loss to avoid this could be interesting. For online reinforcement learning too, a regularisation loss throughout the learning procedure might make more sense than warming up a priori on random trajectories. Moreover, we note that not all benchmarks would benefit from warming up. In fact, it is likely that for several benchmarks, having only a few attractors could be better. In this regard, it would be interesting to try to warm up in order to reach a specific number of attractors, rather than for maximising them. Finally, the warmup procedure maximises reachable multistability for a particular dataset of input sequences. Warming up on totally random input sequences would result in a simpler procedure that might still provide a good initialisation for reaching multistability.

This work also present some limitations. First, the VAA is not discriminating limit cycles from fixed point attractors. In addition, states that are on the same limit cycles but far from each others are not considered in the basin. Moreover, the warmup maximises the number of attractors present in all hidden states, while we might want the hidden states from a same input sequence to belong to a single basin of attraction. Finally, the stability of the RNN is measured for a stable input, an assumption that is unrealistic in our experiments and in general. It might be worth exploring those problems in future works.

Declaration of competing interest

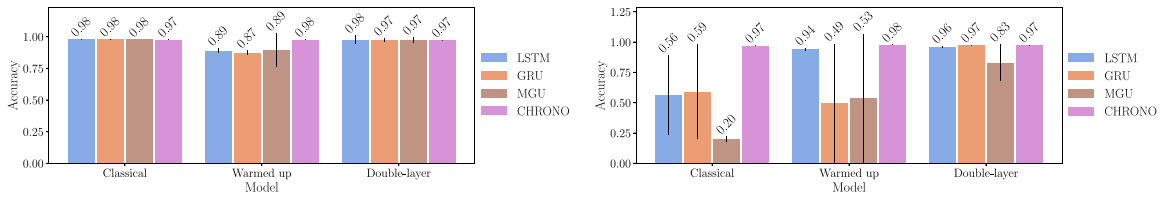
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request

Acknowledgements

Gaspard Lambrechts and Nicolas Vecoven gratefully acknowledge the financial support of the Wallonia-Brussels Federation for their FRIA grant. Florent De Geeter and Gaspard Lambrechts gratefully acknowledge the financial support of the Walloon Region for Grant No. 2010235 – ARIAC by DW4AI. Computational resources have been provided by the *Consortium des Équipements de Calcul Intensif* (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2502011 and by the Walloon Region, including the Tier-1 supercomputer of the *Fédération Wallonie-Bruxelles*, infrastructure funded by the *Walloon Region* under Grant No. 1117545.



(a) $N = 72$ (b) $N = 472$

Fig. 20. Test accuracy for the permuted line-sequential MNIST benchmark for different forgetting periods N. Mean and standard deviation are reported after 70 epochs. We note that when N equals 72 (472) the resulting image has 100 (500) lines.

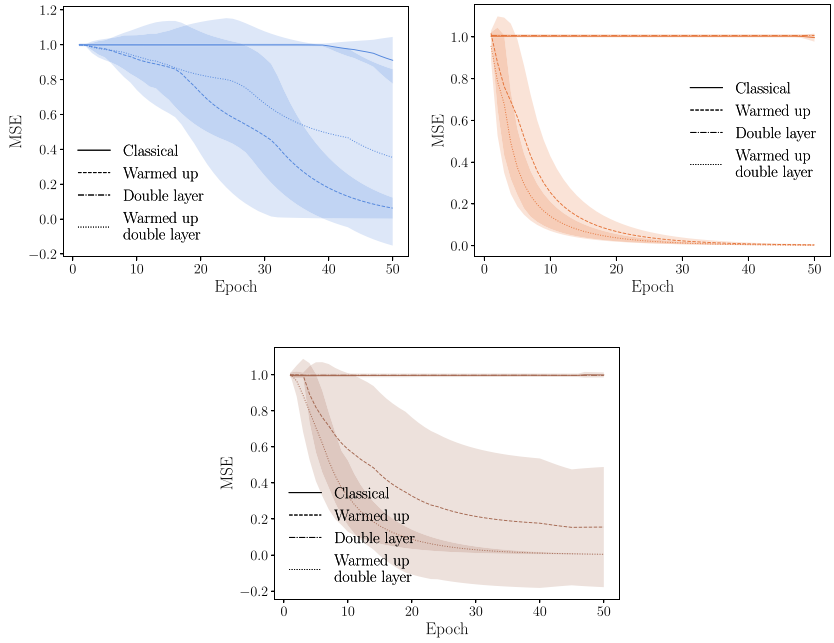
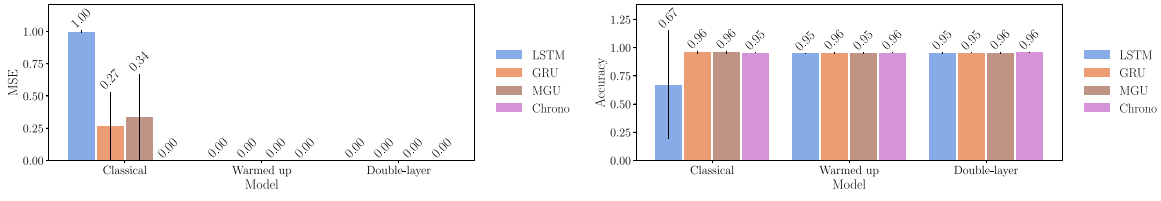


Fig. 21. Evolution of the validation loss on the denoising benchmark for LSTM, GRU and MGU networks, with $N = 100$ and $T = 200$. For each cell, four versions are considered: the classical one, the warmed-up one and the double-layer one, with and without partial warmup.



(a) Denoising benchmark with $T = 200$ and $N = 100$. (b) Permuted line-sequential MNIST with $N = 472$.

Fig. 22. Test accuracy for the denoising benchmark and the permuted line-sequential MNIST benchmark with hyperparameter selection on the learning set. Mean and standard deviation are reported after 50 epochs.

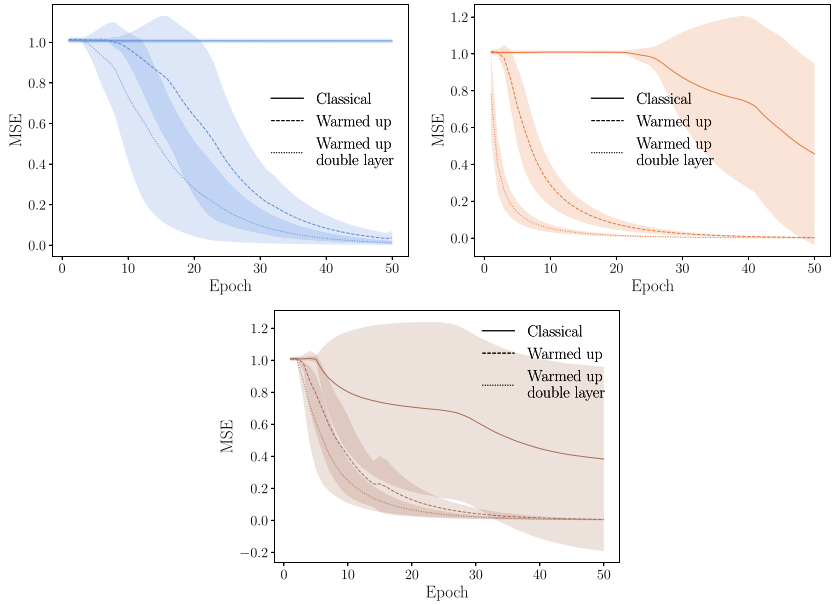


Fig. 23. Evolution of the validation loss on the denoising benchmark for LSTM, GRU and MGU networks, with $N = 100$ and $T = 200$. For each cell, three versions are considered: the classical one, the warmed-up one and the double-layer one with partial warmup. The hyperparameters of each cell version were optimised on the learning set.

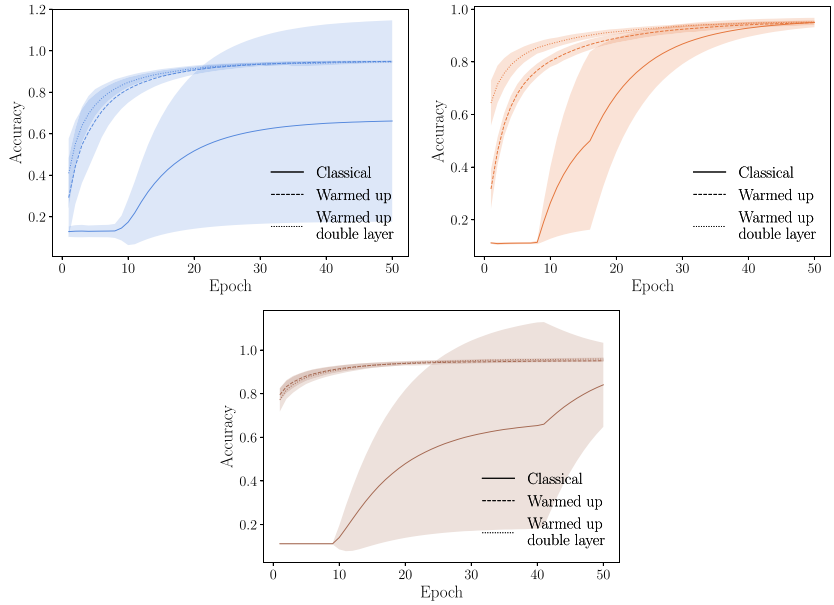


Fig. 24. Evolution of the validation loss on the line-sequential MNIST benchmark for LSTM, GRU and MGU networks, with $N = 100$ and $T = 200$. For each cell, three versions are considered: the classical one, the warmed-up one and the double-layer one with partial warmup. The hyperparameters of each cell version were optimised on the learning set.

Appendix A. Recurrent neural network architectures

Formally, an RNN architecture is defined by its update function f, its output function g and its initialisation function h that are parameterised by a vector $\theta \in \mathbb{R}^d$. Given a sequence of inputs $\mathbf{u}_{1:T} = [\mathbf{u}_1, \dots, \mathbf{u}_T]$, with $T \in \mathbb{N}$ and $\mathbf{u}_t \in \mathbb{R}^n$, the RNN maintains a hidden state $\mathbf{x}_t$ and output a prediction $\mathbf{o}_t$ according to

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t; \theta), \ t = 1, \dots, T, \tag{A.1}$$

$$\mathbf{o}_t = g(\mathbf{x}_t; \theta), \ t = 1, \dots, T, \tag{A.2}$$

$$\mathbf{x}_0 = h(\theta). \tag{A.3}$$

RNNs can be composed of L layers that are linked sequentially through $\mathbf{u}_t^i = \mathbf{o}_t^{i-1}$ with $\mathbf{u}_t^1 = \mathbf{u}_t$ and $\mathbf{o}_t = \mathbf{o}_t^L$, where $\mathbf{o}_t^i$ denotes the output of layer i and $\mathbf{u}_t^i$ its input. In this case, each layer i has its own update function f^i, output function g^i and initialisation function h^i.

In the following, we give the update function f and output function g of a single layer for each architecture considered in this work. As far as the initial hidden state is concerned, it is always chosen to zero, i.e., $h(\theta) = 0$. Note that $\sigma(x) = \frac{1}{1+e^{-x}}$ denote the sigmoid activation function, and $\odot$ to denote the Hadamard product.

Long short-term memory. The LSTM update and output functions are defined from the following intermediate values.

$$\mathbf{f}_t = \sigma(\mathbf{W}_{fu}\mathbf{u}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{b}_t) \tag{A.4}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_{iu}\mathbf{u}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{b}_i) \tag{A.5}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{ou}\mathbf{u}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{b}_r) \tag{A.6}$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_{cu}\mathbf{u}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c) \tag{A.7}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \tag{A.8}$$

$$\mathbf{h}_t = \mathbf{r}_t \odot \tanh(\mathbf{c}_t) \tag{A.9}$$

The hidden state is given by $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t; \theta) = [\mathbf{h}_t, \mathbf{c}_t]$, and the output is given by $\mathbf{o}_t = g(\mathbf{x}_t; \theta) = \mathbf{h}_t$. The parameters of the LSTM network are $\theta = (\mathbf{W}_{fu}, \mathbf{W}_{fh}, \mathbf{W}_{iu}, \mathbf{W}_{ih}, \mathbf{W}_{ou}, \mathbf{W}_{oh}, \mathbf{W}_{cu}, \mathbf{W}_{ch}, \mathbf{b}_t, \mathbf{b}_i, \mathbf{b}_r, \mathbf{b}_c)$.

Gated recurrent unit. The GRU update and output functions are defined from the following intermediate values.

$$\mathbf{z}_t = \sigma(\mathbf{W}_{zu}\mathbf{u}_t + \mathbf{W}_{zh}\mathbf{h}_{t-1} + \mathbf{b}_z) \tag{A.10}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{ru}\mathbf{u}_t + \mathbf{W}_{rh}\mathbf{h}_{t-1} + \mathbf{b}_r) \tag{A.11}$$

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tanh(\mathbf{W}_{hu}\mathbf{u}_t + \mathbf{r}_t \odot \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{A.12}$$

The hidden state is given by $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t; \theta) = \mathbf{h}_t$, and the output is given by $\mathbf{o}_t = g(\mathbf{x}_t; \theta) = \mathbf{h}_t$. The parameters of the GRU network are $\theta = (\mathbf{W}_{zu}, \mathbf{W}_{zh}, \mathbf{W}_{ru}, \mathbf{W}_{rh}, \mathbf{W}_{hu}, \mathbf{W}_{hh}, \mathbf{b}_z, \mathbf{b}_r, \mathbf{b}_h)$.

Bistable recurrent cell. The BRC update and output functions are defined from the following intermediate values.

$$\mathbf{c}_t = \sigma(\mathbf{W}_{cu}\mathbf{u}_t + w_c \odot \mathbf{h}_{t-1} + \mathbf{b}_c) \tag{A.13}$$

$$\mathbf{a}_t = 1 + \tanh(\mathbf{W}_{au}\mathbf{u}_t + w_a \odot \mathbf{h}_{t-1} + \mathbf{b}_a) \tag{A.14}$$

$$\mathbf{h}_t = \mathbf{c}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{c}_t) \odot \tanh(\mathbf{W}_{hu} + \mathbf{a}_t \odot \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{A.15}$$

The hidden state is given by $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t; \theta) = \mathbf{h}_t$, and the output is given by $\mathbf{o}_t = g(\mathbf{x}_t; \theta) = \mathbf{h}_t$. The parameters of the BRC network are $\theta = (\mathbf{W}_{cu}, \mathbf{w}_c, \mathbf{W}_{au}, \mathbf{w}_a, \mathbf{W}_{hu}, \mathbf{b}_c, \mathbf{b}_a, \mathbf{b}_h)$.

Neuromodulated bistable recurrent cell. The NBRC update and output functions are defined from the following intermediate values.

$$\mathbf{c}_t = \sigma(\mathbf{W}_{cu}\mathbf{u}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c) \tag{A.16}$$

$$\mathbf{a}_t = 1 + \tanh(\mathbf{W}_{au}\mathbf{u}_t + \mathbf{W}_{ah}\mathbf{h}_{t-1} + \mathbf{b}_a) \tag{A.17}$$

$$\mathbf{h}_t = \mathbf{c}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{c}_t) \odot \tanh(\mathbf{W}_{hu} + \mathbf{a}_t \odot \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{A.18}$$

The hidden state is given by $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t; \theta) = \mathbf{h}_t$, and the output is given by $\mathbf{o}_t = g(\mathbf{x}_t; \theta) = \mathbf{h}_t$. The parameters of the NBRC network are $\theta = (\mathbf{W}_{cu}, \mathbf{W}_{ch}, \mathbf{W}_{au}, \mathbf{W}_{ah}, \mathbf{W}_{hu}, \mathbf{b}_c, \mathbf{b}_a, \mathbf{b}_h)$.

Minimal gated unit. The MGU update and output functions are defined from the following intermediate values.

$$\mathbf{f}_t = \sigma(\mathbf{W}_{fu}\mathbf{u}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{b}_f) \tag{A.19}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_{hu}\mathbf{u}_t + \mathbf{W}_{hh}(\mathbf{f}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \tag{A.20}$$

$$\mathbf{h}_t = \mathbf{f}_t \odot \tilde{\mathbf{h}} + (1 - \mathbf{f}_t) \odot \mathbf{h}_{t-1} \tag{A.21}$$

The hidden state is given by $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t; \theta) = \mathbf{h}_t$, and the output is given by $\mathbf{o}_t = g(\mathbf{x}_t; \theta) = \mathbf{h}_t$. The parameters of the MGU network are $\theta = (\mathbf{W}_{fu}, \mathbf{W}_{fh}, \mathbf{W}_{hu}, \mathbf{W}_{hh}, \mathbf{b}_f, \mathbf{b}_h)$.

Appendix B. Partially observable Markov decision process

Formally, a POMDP P is an 8-tuple $P = (\mathcal{S}, \mathcal{A}, \mathcal{O}, p_0, T, R, O, \gamma)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, and $\mathcal{O}$ is the observation space. The initial state distribution p_0 gives the probability $p_0(\mathbf{s}_0)$ of $\mathbf{s}_0 \in \mathcal{S}$ being the initial state of the decision process. The dynamics are described by the transition distribution T that gives the probability $T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ of $\mathbf{s}_{t+1} \in \mathcal{S}$ being the state resulting from action $\mathbf{a}_t \in \mathcal{A}$ in state $\mathbf{s}_t \in \mathcal{S}$. The reward function R gives the immediate reward $r_t = R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ obtained after each transition. The observation distribution O gives the probability $O(\mathbf{o}_t \mid \mathbf{s}_t)$ to get observation $\mathbf{o}_t \in \mathcal{O}$ in state $\mathbf{s}_t \in \mathcal{S}$. Finally, the discount factor $\gamma \in [0, 1[$ gives the relative importance of future rewards.

Taking a sequence of t actions $(\mathbf{a}_{0:t-1})$ in the POMDP conditions its execution and provides a sequence of $t+1$ observations $(\mathbf{o}_{0:t})$. Together, they compose the history $\eta_{0:t} = (\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}) \in \mathcal{H}_{0:t}$ until timestep t, where $\mathcal{H}_{0:t}$ is the set of such histories. Let $\eta \in \mathcal{H}$ denote a history of arbitrary length sampled in the POMDP, and let $\mathcal{H} = \bigcup_{t=0}^{\infty} \mathcal{H}_{0:t}$ denote the set of histories of arbitrary length.

A policy $\pi \in \Pi$ in a POMDP is a mapping from histories to actions, where $\Pi = \mathcal{H} \to \mathcal{A}$ is the set of such mappings. A policy $\pi^* \in \Pi$ is said to be optimal when it maximises the expected discounted sum of future rewards starting from any history $\eta_{0:t} \in \mathcal{H}_{0:t}$ at time $t \in \mathbb{N}_0$

$$\pi^* \in \operatorname*{argmax}_{\pi \in \Pi} \mathbb{E}_{\pi,P}\left[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'} \;\middle|\; \eta_{0:t}\right], \ \forall \eta_{0:t} \in \mathcal{H}_{0:t}, \ \forall t \in \mathbb{N}_0. \tag{B.1}$$

The history-action value function, or $\mathcal{Q}$-function, is defined as the maximal expected discounted reward that can be gathered, starting from a history $\eta_{0:t} \in \mathcal{H}_{0:t}$ at time $t \in \mathbb{N}_0$ and an action $\mathbf{a}_t \in \mathcal{A}$

$$\mathcal{Q}(\eta_{0:t}, \mathbf{a}_t) = \max_{\pi \in \Pi} \mathbb{E}_{\pi,P}\left[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'} \;\middle|\; \eta_{0:t}, \mathbf{a}_t\right], \ \forall \eta_{0:t} \in \mathcal{H}_{0:t}, \ \forall \mathbf{a}_t \in \mathcal{A}, \ \forall t \in \mathbb{N}_0. \tag{B.2}$$

The $\mathcal{Q}$-function is also the unique solution of the Bellman equation (Kaelbling, Littman, & Cassandra, 1998; Porta, Spaan, & Vlassis, 2004; Smallwood & Sondik, 1973)

$$\mathcal{Q}(\eta, \mathbf{a}) = \mathbb{E}_P\left[r + \gamma \max_{\mathbf{a}' \in \mathcal{A}} \mathcal{Q}(\eta', \mathbf{a}') \;\middle|\; \eta, \mathbf{a}\right], \ \forall \eta \in \mathcal{H}, \ \forall \mathbf{a} \in \mathcal{A} \tag{B.3}$$

where $\eta' = \eta \cup (\mathbf{a}, \mathbf{o}')$ and r is the immediate reward obtained when taking action $\mathbf{a}$ in history η. From (B.1) and (B.2), it can be noticed that any optimal policy satisfies

$$\pi^*(\eta) \in \operatorname*{argmax}_{\mathbf{a} \in \mathcal{A}} \mathcal{Q}(\eta, \mathbf{a}), \ \forall \eta \in \mathcal{H}. \tag{B.4}$$

Appendix C. Deep recurrent Q-learning

The DRQN (Hausknecht & Stone, 2015) algorithm aims at learning a parametric approximation $\mathcal{Q}_\theta$ of the $\mathcal{Q}$-function, where $\theta \in \mathbb{R}^{d_\theta}$ is the parameter vector of a recurrent neural network. This algorithm is motivated by Eq. (B.4) that shows that an optimal policy can be derived from the $\mathcal{Q}$-function. The strategy consists of minimising with respect to θ, for all $(\eta, \mathbf{a})$, the distance between the estimation $\mathcal{Q}_\theta(\eta, \mathbf{a})$ of the LHS of Eq. (B.3), and the estimation of the expectation $\mathbb{E}_P[r + \gamma \max_{\mathbf{a}' \in \mathcal{A}} \mathcal{Q}_\theta(\eta', \mathbf{a}')]$ of the RHS of Eq. (B.3). This is done by using transitions $(\eta, \mathbf{a}, r, \mathbf{o}', \eta')$ sampled in the POMDP, with $\eta' = \eta \cup (\mathbf{a}, \mathbf{o}')$.

In practice, this algorithm interleaves the generation of episodes and the update of the estimation $\mathcal{Q}_\theta$. Indeed, in the DRQN algorithm, the episodes are generated with the ε-greedy policy derived from the current estimation $\mathcal{Q}_\theta$. This stochastic policy selects actions according to $\operatorname{argmax}_{\mathbf{a} \in \mathcal{A}} \mathcal{Q}_\theta(\cdot, \mathbf{a})$ with probability

$1-\varepsilon$, and according to an exploration policy with probability ε. This exploration policy is defined by a probability distribution $\mathcal{E}(\mathcal{A}) \in \mathcal{P}(\mathcal{A})$ over the actions, where $\mathcal{P}(\mathcal{A})$ is the set of probability measures over the action space $\mathcal{A}$. The DRQN algorithm also introduces a truncation horizon H such that the histories generated in the POMDP have a maximum length of H. Moreover, a replay buffer of histories is used and the gradient is evaluated on a batch of histories sampled from this buffer. Furthermore, the parameters θ are updated with the Adam algorithm (Kingma & Ba, 2014). Finally, the target $r_t + \gamma \max_{\mathbf{a}\in\mathcal{A}} \mathcal{Q}_{\theta'}(\eta_{0:t+1}, \mathbf{a})$ is computed using a past version $\mathcal{Q}_{\theta'}$ of the estimation $\mathcal{Q}_\theta$ with parameters θ' that are updated to θ less frequently, which eases the convergence towards the target, and ultimately towards the $\mathcal{Q}$-function.

Algorithm 4: DRQN - $\mathcal{Q}$-function approximation

Parameters: $N \in \mathbb{N}$ the buffer capacity.
$C \in \mathbb{N}$ the target update period in term of episodes.
$E \in \mathbb{N}$ the number of episodes.
$H \in \mathbb{N}$ the truncation horizon.
$I \in \mathbb{N}$ the number of gradient steps after each episode.
$\varepsilon \in \mathbb{R}$ the exploration rate.
$\mathcal{E}(\mathcal{A}) \in \mathcal{P}(\mathcal{A})$ the exploration policy probability distribution.
$\alpha \in \mathbb{R}$ the learning rate.
$B \in \mathbb{N}$ the batch size.
$\theta \in \mathbb{R}^{d_\theta}$ the initial parameters of the network.
$\theta' \in \mathbb{R}^{d_\theta}$ the initial parameters of the target network.

Initialise weights θ randomly
Fill replay buffer $\mathcal{B}$ with random transitions from the exploration policy $\mathcal{E}(\mathcal{A})$.
if *warmup* **then**
  Let $\mathcal{D}$ be the set of histories η (input sequences) in replay buffer $\mathcal{B}$.
  `Warmup` $(\mathcal{D}, \theta)$ using default parameters of the `Warmup` algorithm.
for $e = 0, \dots, E-1$ **do**
  if $e \bmod C = 0$ **then**
    Update target network with $\theta' \leftarrow \theta$
  `// Generate new episode, store history and rewards`
  Draw an initial state $\mathbf{s}_0$ according to p_0 and observe $\mathbf{o}_0$
  Let $\eta_{0:0} = (\mathbf{o}_0)$
  for $t = 0, \dots, H-1$ **do**
    Select $\mathbf{a}_t \sim \mathcal{E}(\mathcal{A})$ with probability ε, otherwise select $\mathbf{a}_t = \operatorname{argmax}_{\mathbf{a}\in\mathcal{A}} \{\mathcal{Q}_\theta(\eta_{0:t}, \mathbf{a})\}$
    Take action $\mathbf{a}_t$ and observe r_t and $\mathbf{o}_{t+1}$
    Let $\eta_{0:t+1} = (\mathbf{o}_0, \mathbf{a}_0, \mathbf{o}_1, \dots, \mathbf{o}_{t+1})$
    if $|\mathcal{B}| < N$ **then** add $(\eta_{0:t}, \mathbf{a}_t, r_t, \mathbf{o}_{t+1}, \eta_{0:t+1})$ in replay buffer $\mathcal{B}$ **else** replace oldest transition in replay buffer $\mathcal{B}$ by $(\eta_{0:t}, \mathbf{a}_t, r_t, \mathbf{o}_{t+1}, \eta_{0:t+1})$
    if $\mathbf{o}_{t+1}$ *is terminal* **then**
      break
  `// Optimise recurrent Q-network`
  for $i = 0, \dots, I-1$ **do**
    Sample B transitions $(\eta^b_{0:t}, \mathbf{a}^b_t, r^b_t, \mathbf{o}^b_{t+1}, \eta^b_{0:t+1})$ uniformly from the replay buffer $\mathcal{B}$
    Compute targets
    $$y^b = \begin{cases} r^b_t + \gamma \max_{\mathbf{a}\in\mathcal{A}} \left\{\mathcal{Q}_{\theta'}(\eta^b_{0:t+1}, \mathbf{a})\right\} & \text{if } \mathbf{o}^b_{t+1} \text{ is not terminal} \\ r^b_t & \text{otherwise} \end{cases}$$
    Compute loss $L = \sum_{b=0}^{B-1} \left(y^b - \mathcal{Q}_\theta(\eta^b_{0:t}, \mathbf{a}^b_t)\right)^2$
    Compute direction g using Adam optimiser, perform gradient step $\theta \leftarrow \theta + \alpha g$

The DRQN training procedure is detailed in Algorithm . In this algorithm, the output of the RNN is $\mathbf{y}_t = g(\mathbf{h}_t; \theta) \in \mathbb{R}^{|\mathcal{A}|}$, and it gives $\mathcal{Q}_\theta(\eta_{0:t}, \mathbf{a}), \ \forall \mathbf{a} \in \mathcal{A}$. The hidden states are given by $\mathbf{h}_k = f(\mathbf{h}_{k-1}, \mathbf{x}_k; \theta), \ \forall k \in \mathbb{N}_0$, with the inputs given by $\mathbf{x}_k = (\mathbf{a}_{k-1}, \mathbf{o}_k), \ \forall t \in \mathbb{N}$ and $\mathbf{x}_0 = (\mathbf{0}, \mathbf{o}_0)$. From the approximation $\mathcal{Q}_\theta$, the policy π_θ is given by $\pi_\theta(\eta) = \operatorname{argmax}_{\mathbf{a}\in\mathcal{A}} \mathcal{Q}_\theta(\eta, \mathbf{a})$.

In the experiments, the following hyperparameters have been chosen: $N = 8192$, $C = 20$, $I = 10$, $\varepsilon = 0.2$, $\alpha = 1 \times 10^3$, $B = 32$. The exploration policy and truncation horizon depend on the environment and are thus detailed in the following appendix.

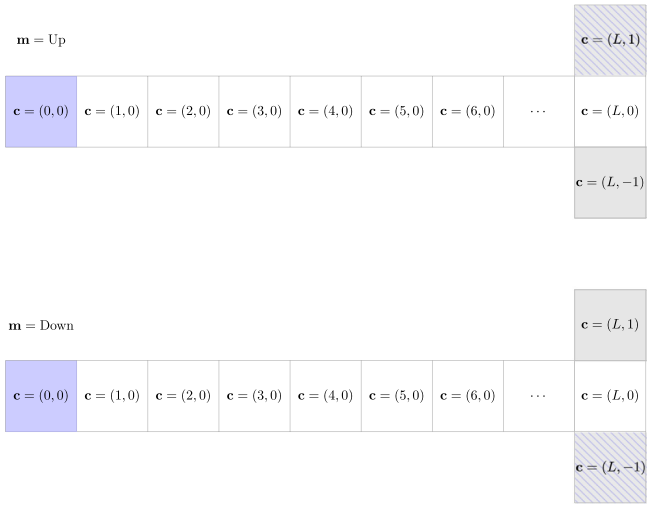


Fig. D.25. T-Maze state space. Initial states in blue, terminal states in grey, and treasure states hatched.

Appendix D. T-maze environment

The T-Maze environment is a POMDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, p_0, T, R, O, \gamma)$ parameterised by the maze length $L \in \mathbb{N}$. The formal definition of this environment is given below.

State space. The discrete state space $\mathcal{S}$ is composed of the set of positions $\mathcal{C}$ for the agent in each of the two maze layouts $\mathcal{M}$, as illustrated in Fig. D.25. The maze layout determines the position of the treasure. Formally, we have

$$\begin{cases} \mathcal{S} &= \mathcal{M} \times \mathcal{C} & \text{(a)} \\ \mathcal{M} &= \{\text{Up}, \text{Down}\} & \text{(b)} \\ \mathcal{C} &= \{(0,0), \dots, (L,0)\} \cup \{(L,1), (L,-1)\} & \text{(c)} \end{cases} \tag{D.1}$$

A state $\mathbf{s}_t \in \mathcal{S}$ is thus defined by $\mathbf{s}_t = (\mathbf{m}_t, \mathbf{c}_t)$ with $\mathbf{m}_t \in \mathcal{M}$ and $\mathbf{c}_t \in \mathcal{C}$. Let us also define $\mathcal{F} = \{\mathbf{s}_t = (\mathbf{m}_t, \mathbf{c}_t) \in \mathcal{S} \mid \mathbf{c}_t \in \{(L,1), (L,-1)\}\}$ the set of terminal states, four in number.

Action space. The discrete action space $\mathcal{A}$ is composed of the four possible moves that the agent can take

$$\mathcal{A} = \{(1,0), (0,1), (-1,0), (0,-1)\} \tag{D.2}$$

that correspond to Right, Up, Left and Down, respectively.

Observation space. The discrete observation space $\mathcal{O}$ is composed of the four partial observations of the state that the agent can perceive

$$\mathcal{O} = \{\text{Up}, \text{Down}, \text{Corridor}, \text{Junction}\}. \tag{D.3}$$

Initial state distribution. The two possible initial states are $\mathbf{s}_0^{\text{Up}} = (\text{Up}, (0,0))$ and $\mathbf{s}_0^{\text{Down}} = (\text{Down}, (0,0))$, depending on the maze in which the agent lies. The initial state distribution $p_0 : \mathcal{S} \to [0,1]$ is thus given by

$$p_0(\mathbf{s}_0) = \begin{cases} 0.5 & \text{if } \mathbf{s}_0 = \mathbf{s}_0^{\text{Up}} \\ 0.5 & \text{if } \mathbf{s}_0 = \mathbf{s}_0^{\text{Down}} \\ 0 & \text{otherwise} \end{cases} \tag{D.4}$$

Transition distribution. The transition distribution function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is given by

$$T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) = \delta_{f(\mathbf{s}_t, \mathbf{a}_t)}(\mathbf{s}_{t+1}) \tag{D.5}$$

where $\mathbf{s}_t \in \mathcal{S}$, $\mathbf{a}_t \in \mathcal{A}$ and $\mathbf{s}_{t+1} \in \mathcal{S}$, and f is given by

$$f(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} \mathbf{s}_{t+1} = (\mathbf{m}_t, \mathbf{c}_t + \mathbf{a}_t) & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{c}_t + \mathbf{a}_t \in \mathcal{C} \\ \mathbf{s}_{t+1} = (\mathbf{m}_t, \mathbf{c}_t) & \text{otherwise} \end{cases} \tag{D.6}$$

where $\mathbf{s}_t = (\mathbf{m}_t, \mathbf{c}_t) \in \mathcal{S}$ and $\mathbf{a}_t \in \mathcal{A}$.

Reward function. The reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is given by

$$R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) = \begin{cases} 0 & \text{if } \mathbf{s}_t \in \mathcal{F} \\ 0 & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{s}_{t+1} \notin \mathcal{F}, \mathbf{s}_t \neq \mathbf{s}_{t+1} \\ -0.1 & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{s}_{t+1} \notin \mathcal{F}, \mathbf{s}_t = \mathbf{s}_{t+1} \\ 4 & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{s}_{t+1} \in \mathcal{F}, \mathbf{c}_{t+1} = \begin{cases} (L, 1) & \text{if } \mathbf{m}_{t+1} = \text{Up} \\ (L, -1) & \text{if } \mathbf{m}_{t+1} = \text{Down} \end{cases} \\ -0.1 & \text{if } \mathbf{s}_t \notin \mathcal{F}, \mathbf{s}_{t+1} \in \mathcal{F}, \mathbf{c}_{t+1} = \begin{cases} (L, -1) & \text{if } \mathbf{m}_{t+1} = \text{Up} \\ (L, +1) & \text{if } \mathbf{m}_{t+1} = \text{Down} \end{cases} \end{cases} \tag{D.7}$$

where $\mathbf{s}_t = (\mathbf{m}_t, \mathbf{c}_t) \in \mathcal{S}$, $\mathbf{a}_t \in \mathcal{A}$ and $\mathbf{s}_{t+1} = (\mathbf{m}_{t+1}, \mathbf{c}_{t+1}) \in \mathcal{S}$.

Observation distribution. In the T-Maze, the observations are deterministic. The observation distribution $O : \mathcal{S} \times \mathcal{O} \to [0, 1]$ is given by

$$O(\mathbf{o}_t \mid \mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{o}_t = \text{Up}, \mathbf{c}_t = (0, 0), \mathbf{m}_t = \text{Up} \\ 1 & \text{if } \mathbf{o}_t = \text{Down}, \mathbf{c}_t = (0, 0), \mathbf{m}_t = \text{Down} \\ 1 & \text{if } \mathbf{o}_t = \text{Corridor}, \mathbf{c}_t \in \{(1, 0), \dots, (L-1, 0)\} \\ 1 & \text{if } \mathbf{o}_t = \text{Junction}, \mathbf{c}_t \in \{(L, 0), (L, 1), (L, -1)\} \\ 0 & \text{otherwise} \end{cases} \tag{D.8}$$

where $\mathbf{s}_t = (\mathbf{m}_t, \mathbf{c}_t) \in \mathcal{S}$ and $\mathbf{o}_t \in \mathcal{O}$.

Exploration policy. The exploration policy $\mathcal{E} : \mathcal{A} \to [0, 1]$ is a stochastic policy that is given by $\mathcal{E}(\text{Right}) = 1/2$ and $\mathcal{E}(\text{Other}) = 1/6$ where Other $\in$ {Up, Left, Down}. It enforces to explore the right of the maze layouts. This exploration policy, tailored to the T-Maze environment, allows one to speed up the training procedure, without interfering with the study of this work.

Truncation horizon. The truncation horizon H of the DRQN algorithm is chosen such that the expected displacement of an agent moving according to the exploration policy in a T-Maze with an infinite corridor on both sides is greater than L. Let $r = \mathcal{E}(\text{Right})$ and $l = \mathcal{E}(\text{Left})$. In this infinite T-Maze, starting at 0, the expected displacement after one timestep is $\bar{x}_1 = r - l$. By independence, $\bar{x}_H = H\bar{x}_1$ such that, for $\bar{x}_H \geq L$, the time horizon is given by

$$H = \left\lceil \frac{L}{r - l} \right\rceil. \tag{D.9}$$

Appendix E. Generalisation to other hyperparameters

In this section, we study the generalisation of the results of this work to other hyperparameters. More precisely, we vary the number of recurrent layers, the number of neurons in each layer, the batch size, and the learning rate. In Appendix E.1, we study if the VAA increases when learning occurs for the copy first input benchmark with $T = 50$. In Appendix E.2, we study if the warmup procedure and the double layer architecture improve learning for the permuted row sequential MNIST benchmark with $N = 472$. Finally, in Appendix E.3, we study the impact of the warmup procedure on the copy first input benchmark with $T = 300$ for different values of k. All averages and standard deviations reported were computed over three different training sessions.

E.1. Generalisation of the correlation between multistability and learning

In Fig. E.26 and Fig. E.27, we can see the evolution of the loss on the validation set and of the VAA for different hyperparameters. There is a clear correlation between learning and multistability, for all choices of hyperparameters. More precisely, it can be seen that learning loss decrease generally starts when the VAA starts increasing. Moreover, the loss is highly correlated with the VAA.

Table F.1
Denoising benchmark: Test MSE after hyperparameter selection.

		L	H	α	B	MSE
LSTM	Classical	3	512	1×10^{-3}	32	0.9970 ± 0.0089
	Double	3	256	5×10^{-4}	64	$\mathbf{0.0004 \pm 0.0001}$
	Warmup	3	256	1×10^{-3}	64	0.0010 ± 0.0009
GRU	Classical	1	512	1×10^{-3}	32	0.2656 ± 0.4593
	Double	2	256	1×10^{-3}	32	$\mathbf{0.0002 \pm 0.0001}$
	Warmup	1	512	5×10^{-4}	64	$\mathbf{0.0002 \pm 0.0001}$
MGU	Classical	3	512	1×10^{-3}	32	0.3356 ± 0.5695
	Double	2	256	5×10^{-4}	32	$\mathbf{0.0003 \pm 0.0000}$
	Warmup	1	256	1×10^{-3}	32	$\mathbf{0.0003 \pm 0.0002}$
Chrono	Classical	1	512	1×10^{-3}	32	$\mathbf{0.0003 \pm 0.0002}$
	Double	1	512	1×10^{-3}	32	0.0004 ± 0.0003
	Warmup	1	512	1×10^{-3}	32	0.0004 ± 0.0002
BRC	Classical	3	256	1×10^{-3}	32	$\mathbf{0.0006 \pm 0.0001}$
NBRC	Classical	1	512	1×10^{-3}	32	$\mathbf{0.0001 \pm 0.0000}$

Table F.2
Line Sequential MNIST: Test accuracy after hyperparameter selection.

		L	H	α	B	Accuracy
LSTM	Classical	2	512	1×10^{-3}	64	0.6693 ± 0.4807
	Double	2	256	5×10^{-4}	32	$\mathbf{0.9519 \pm 0.0058}$
	Warmup	3	256	5×10^{-4}	32	0.9475 ± 0.0008
GRU	Classical	3	512	5×10^{-4}	32	$\mathbf{0.9578 \pm 0.0087}$
	Double	2	512	1×10^{-4}	32	0.9549 ± 0.0011
	Warmup	1	512	1×10^{-4}	32	0.9555 ± 0.0053
MGU	Classical	2	512	5×10^{-4}	64	$\mathbf{0.9576 \pm 0.0085}$
	Double	2	512	5×10^{-4}	64	0.9562 ± 0.0045
	Warmup	2	256	5×10^{-4}	32	0.9485 ± 0.0073
Chrono	Classical	1	256	1×10^{-3}	32	0.9545 ± 0.0020
	Double	1	256	1×10^{-3}	32	$\mathbf{0.9575 \pm 0.0029}$
	Warmup	1	256	1×10^{-3}	32	0.9562 ± 0.0017
BRC	Classical	2	512	5×10^{-4}	32	$\mathbf{0.9589 \pm 0.0064}$
NBRC	Classical	2	512	5×10^{-4}	64	$\mathbf{0.9600 \pm 0.0006}$

E.2. Generalisation of the warmup procedure

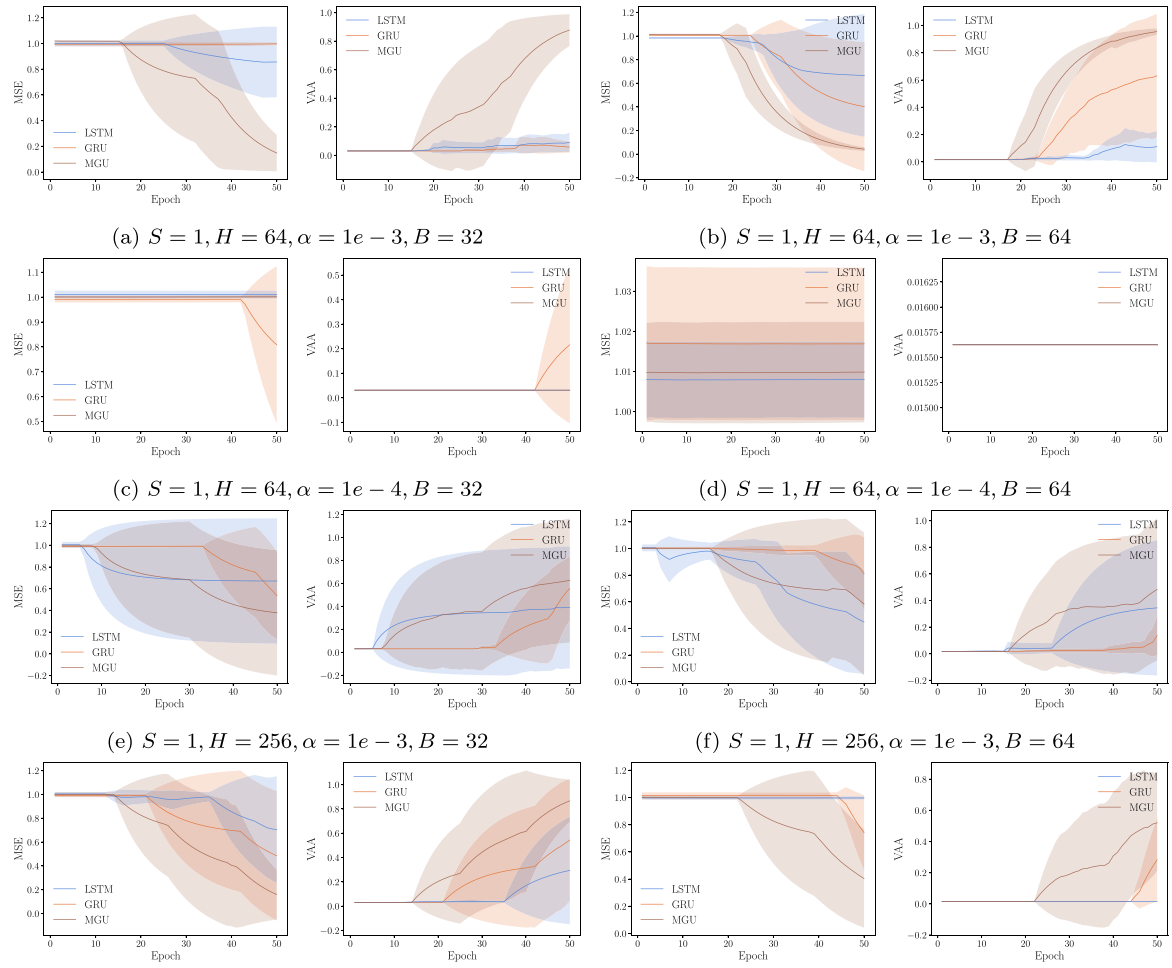
In Fig. E.28 and Fig. E.29, we can see the evolution of the loss on the validation set and the test set accuracy for different hyperparameters. It can be seen from those figures that the warmup procedure and the double layer architecture with partial warmup both improve on the classically initialised GRU architecture. Those improvements are consistent over all hyperparameters choices. It can be noted that the warmup procedure is sometimes better than the double layer architecture in terms of speed of convergence, notably when using a single RNN layer and a small hidden size.

E.3. Impact of the parameter k in the warmup procedure

In Fig. E.30, we can see the impact of the target VAA* k used in the warmup procedure on the final test loss, for the copy first input benchmark for different sequence lengths T. It can be seen that for this benchmark with long time dependencies, the higher k, the lower the MSE.

Appendix F. Hyperparameters optimisation

In this section, we report the best hyperparameters obtained for each cell version and the final test loss obtained for those



(a) $S = 1, H = 64, \alpha = 1e-3, B = 32$ (b) $S = 1, H = 64, \alpha = 1e-3, B = 64$

(c) $S = 1, H = 64, \alpha = 1e-4, B = 32$ (d) $S = 1, H = 64, \alpha = 1e-4, B = 64$

(e) $S = 1, H = 256, \alpha = 1e-3, B = 32$ (f) $S = 1, H = 256, \alpha = 1e-3, B = 64$

(g) $S = 1, H = 256, \alpha = 1e-4, B = 32$ (h) $S = 1, H = 256, \alpha = 1e-4, B = 64$

Fig. E.26. Evolution of the validation loss (left) and of the VAA (right) of LSTM, GRU and MGU networks, for the copy first input benchmark.

hyperparameters, in Table F.1 for the denoising benchmark with $N = 100$ and in Table F.2 for the line-sequential MNIST benchmark with $N = 472$. The hyperparameter selection procedure is described hereafter. First, the dataset is split into the learning set and the test set. Then, the learning set is split into three sets: the training set, the validation set and the selection set. The network is then trained according to the standard procedure: the final weights are those that have obtained the lowest loss on the validation set, throughout the training on the training set. Those weights are then evaluated on the selection set. This procedure is repeated five times for each set of hyperparameters, with different splits of the learning set each time. Note that those 5 different splits are the same for all cell versions. The set of hyperparameters having obtained the lowest loss on average on the selection set is selected. Using those hyperparameters, the cells are then trained 5 times on the learning set, using a standard training–validation split, and the average score obtained on the test set is reported. The sets of hyperparameters that are considered are given by a grid search.

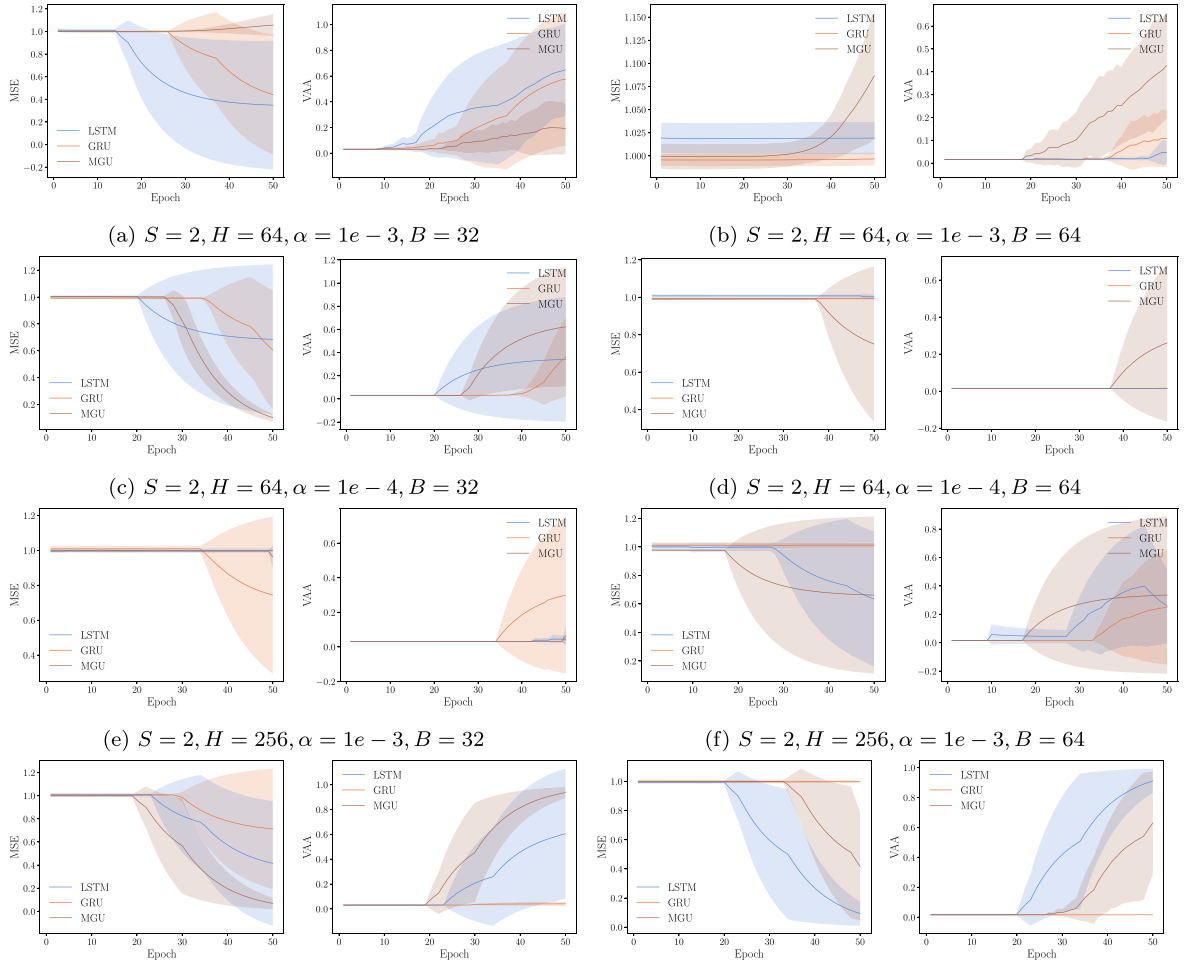
Appendix G. Performance of the double-layer architecture without partial warmup

In this section, we show the performance of all cells on the copy first input and denoising benchmarks including the double-layer architecture without partial warmup. As can be seen from Figs. G.31 and G.32 the double-layer architecture without partial warmup generally performs worse than the classic architecture. This ablation study confirms that the partial warmup is the most important factor for the double-layer architecture performance.

Appendix H. RNNs with auxiliary losses

Trinh et al. (2018) introduces a new method to pretrain and train RNNs on very long sequences using auxiliary losses to teach the networks to correctly encode input sequences. To do this, input sequences are split into small sequences. After each small sequence, the final hidden state is given to a decoder RNN, which must then either reconstruct the sequence or predict the next timesteps. This method has been tested on several image classification datasets where images are fed pixel by pixel, and it has achieved good results. We were wondering if this method also promotes multistability when applied to benchmarks with long-term dependencies. Therefore, we tested it on the copy, denoising and permuted line-sequential MNIST benchmarks. We adapt this implementation[3] to run our tests. For each experiment we ran 20 epochs of pretraining and 50 epochs of training. The VAA is computed at the end of each epoch. As for the auxiliary loss, we have used the reconstruction loss, i.e. the MSE between the real sequence and the reconstructed sequence. During training, the auxiliary loss is also taken into account to make the gradient step.

[3] https://github.com/younggyoseo/rnn-auxiliary-loss



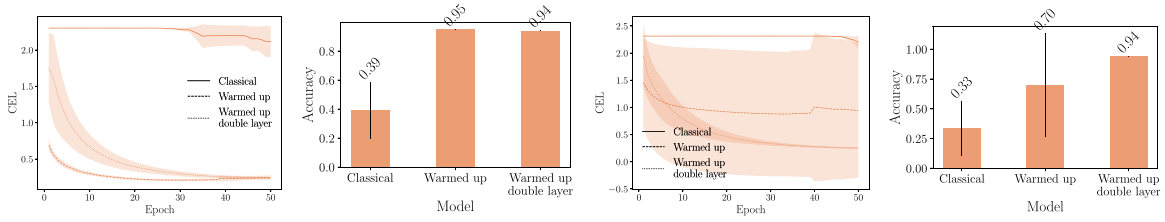
(a) $S = 2, H = 64, \alpha = 1e-3, B = 32$ (b) $S = 2, H = 64, \alpha = 1e-3, B = 64$

(c) $S = 2, H = 64, \alpha = 1e-4, B = 32$ (d) $S = 2, H = 64, \alpha = 1e-4, B = 64$

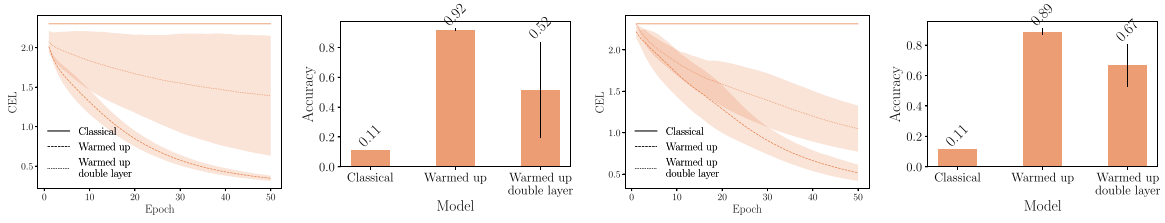
(e) $S = 2, H = 256, \alpha = 1e-3, B = 32$ (f) $S = 2, H = 256, \alpha = 1e-3, B = 64$

(g) $S = 2, H = 256, \alpha = 1e-4, B = 32$ (h) $S = 2, H = 256, \alpha = 1e-4, B = 64$

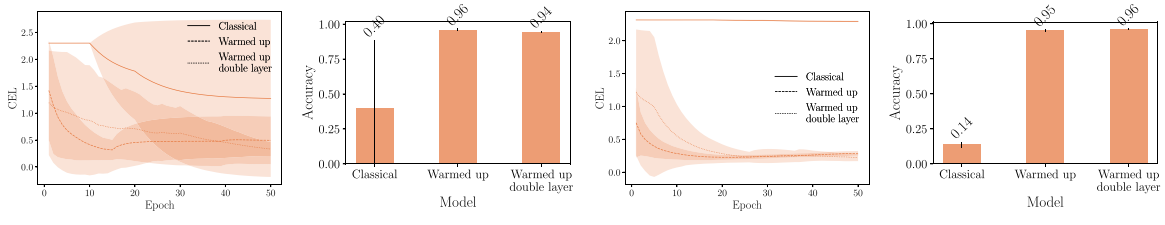
Fig. E.27. Evolution of the validation loss (left) and of the VAA (right) of LSTM, GRU and MGU networks, for the copy first input benchmark.



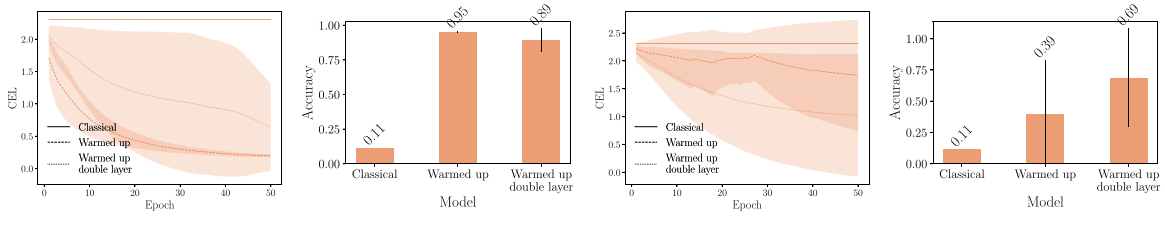
(a) $S = 1, H = 64, \alpha = 1e-3, B = 32$ (b) $S = 1, H = 64, \alpha = 1e-3, B = 64$



(c) $S = 1, H = 64, \alpha = 1e-4, B = 32$ (d) $S = 1, H = 64, \alpha = 1e-4, B = 64$

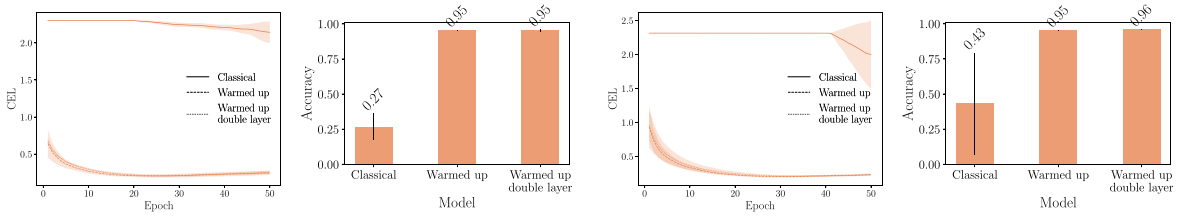


(e) $S = 1, H = 256, \alpha = 1e-3, B = 32$ (f) $S = 1, H = 256, \alpha = 1e-3, B = 64$

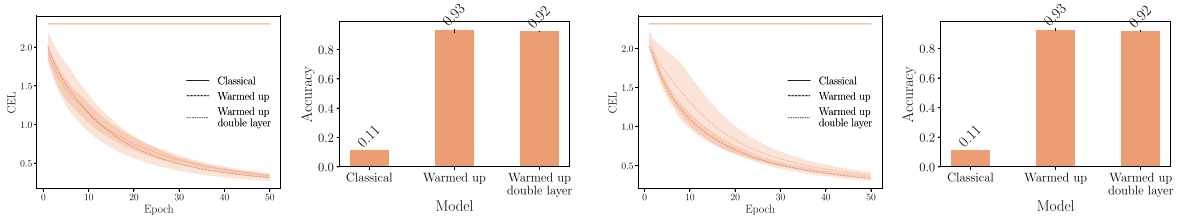


(g) $S = 1, H = 256, \alpha = 1e-4, B = 32$ (h) $S = 1, H = 256, \alpha = 1e-4, B = 64$

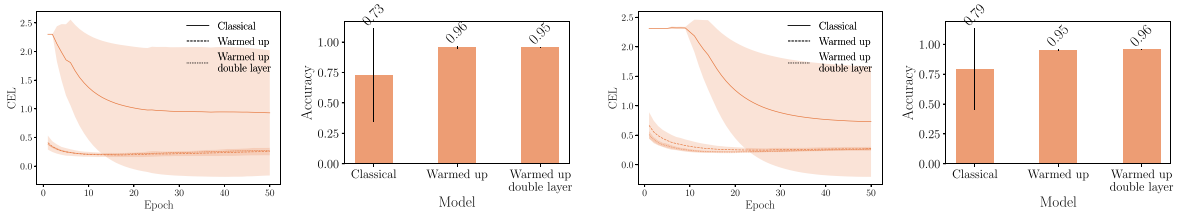
Fig. E.28. Evolution of the validation loss (left) and test set accuracy after 50 epochs (right) of GRU networks, for the permuted line-sequential MNIST benchmark with $N = 472$.



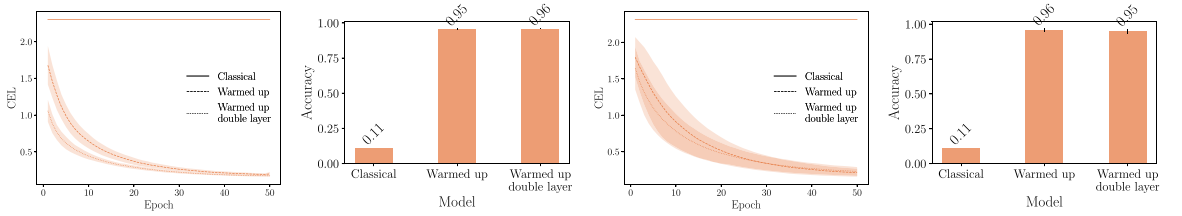
(a) $S = 2, H = 64, \alpha = 1e-3, B = 32$ (b) $S = 2, H = 64, \alpha = 1e-3, B = 64$



(c) $S = 2, H = 64, \alpha = 1e-4, B = 32$ (d) $S = 2, H = 64, \alpha = 1e-4, B = 64$

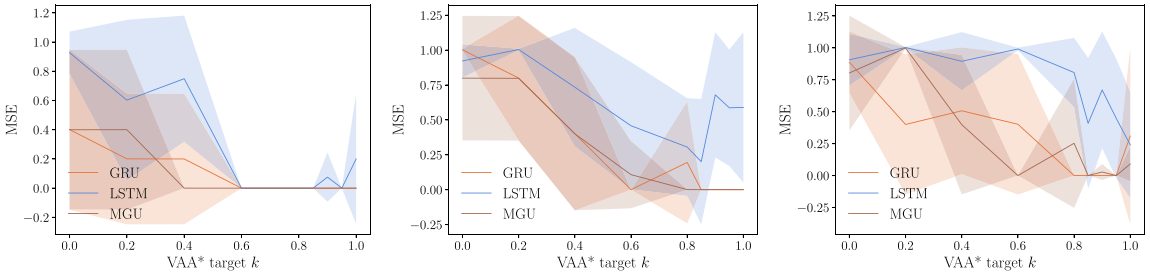


(e) $S = 2, H = 256, \alpha = 1e-3, B = 32$ (f) $S = 2, H = 256, \alpha = 1e-3, B = 64$



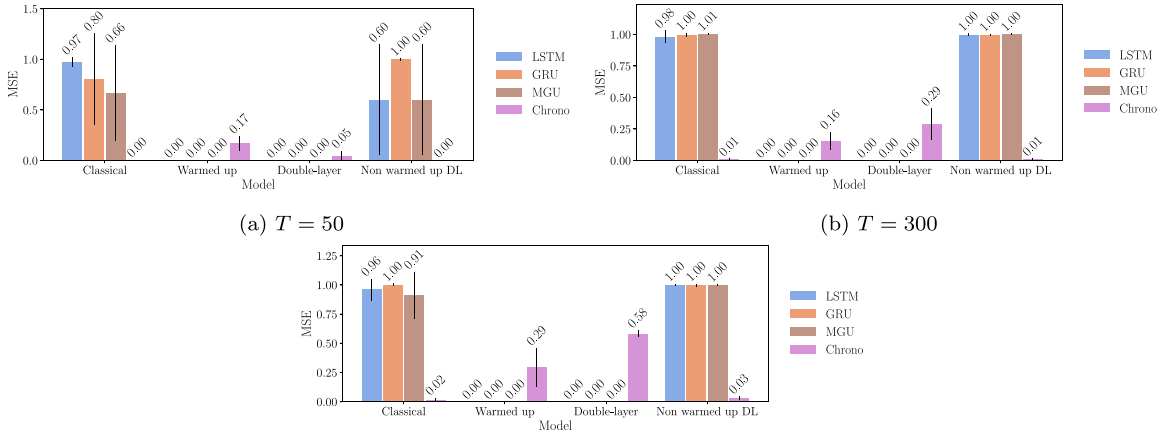
(g) $S = 2, H = 256, \alpha = 1e-4, B = 32$ (h) $S = 2, H = 256, \alpha = 1e-4, B = 64$

Fig. E.29. Evolution of the validation loss (left) and test set accuracy after 50 epochs (right) of GRU networks, for the permuted line-sequential MNIST benchmark with $N = 472$.



(a) $T = 50$ (b) $T = 300$ (c) $T = 600$

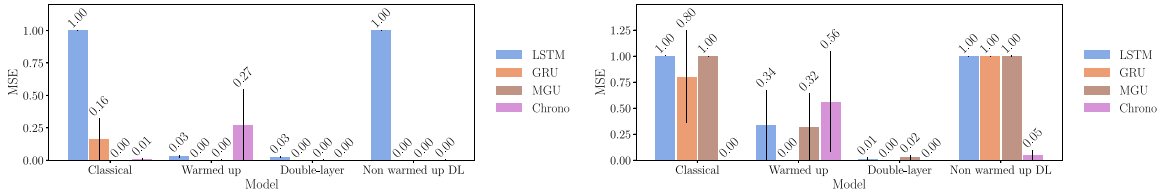
Fig. E.30. Mean squared error (±standard deviation) of different architecture for different value of target VAA* k on the copy first input test set for different values of T.



(a) $T = 50$ (b) $T = 300$

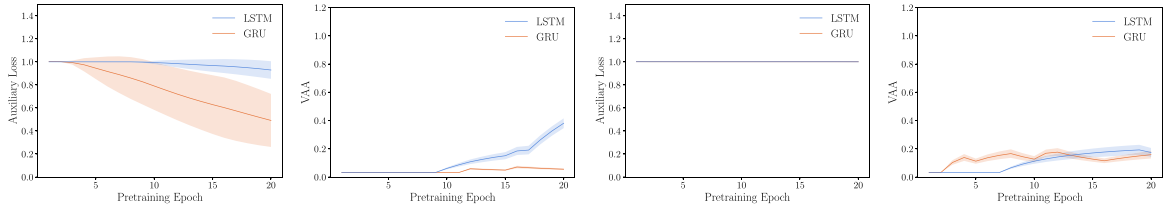
(c) $T = 600$

Fig. G.31. Test MSE loss for the copy first input benchmark with different sequence lengths T. Mean and standard deviation are reported after 50 epochs.



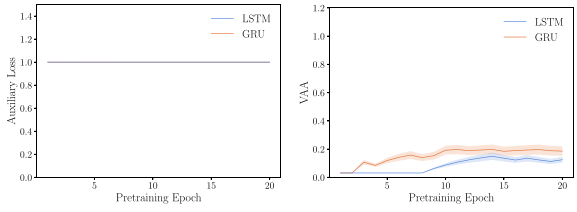
(a) $N = 5$ (b) $N = 100$

Fig. G.32. Test MSE loss for the denoising benchmark with different forgetting periods N and $T = 200$. Mean and standard deviation are reported after 50 epochs.



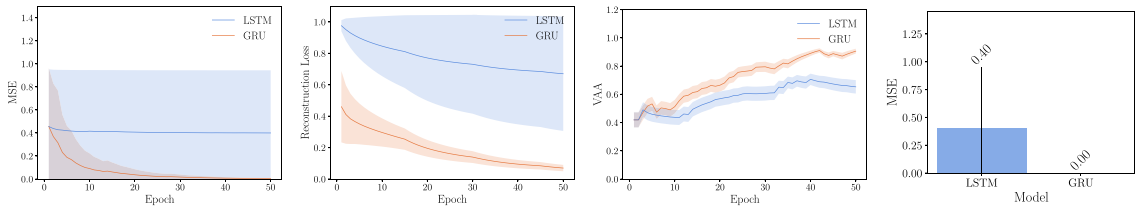
(a) $T = 50$ and the reconstruction loss is evaluated on 30 timesteps.

(b) $T = 300$ and the reconstruction loss is evaluated on 200 timesteps.

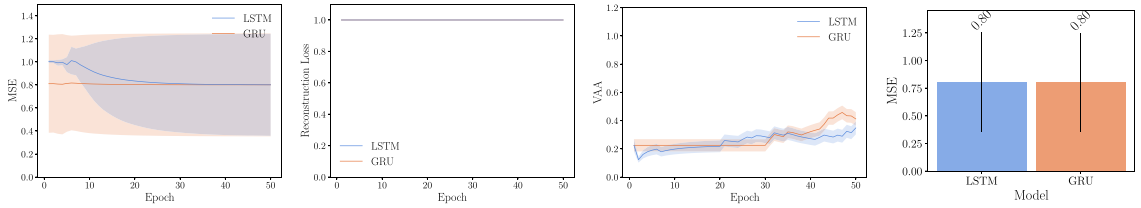


(c) $T = 600$ and the reconstruction loss is evaluated on 400 timesteps.

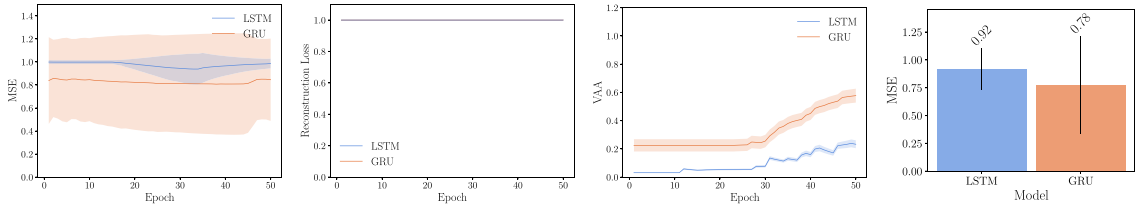
Fig. H.33. Pretraining on the copy first input benchmark for different sequence lengths.



(a) $T = 50$ and the reconstruction loss is evaluated on 30 timesteps.

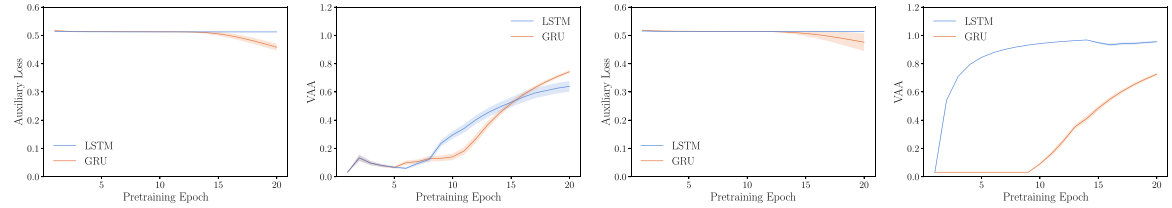


(b) $T = 300$ and the reconstruction loss is evaluated on 200 timesteps.



(c) $T = 600$ and the reconstruction loss is evaluated on 400 timesteps.

Fig. H.34. Training on the copy first input benchmark for different sequence lengths.



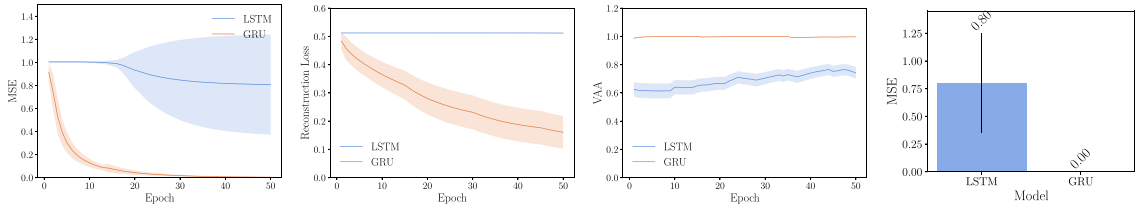
(a) $N = 5$ (b) $N = 100$

Fig. H.35. Pretraining on the denoising benchmark for different forgetting periods and $T = 200$. The reconstruction loss is evaluated on 150 timesteps.

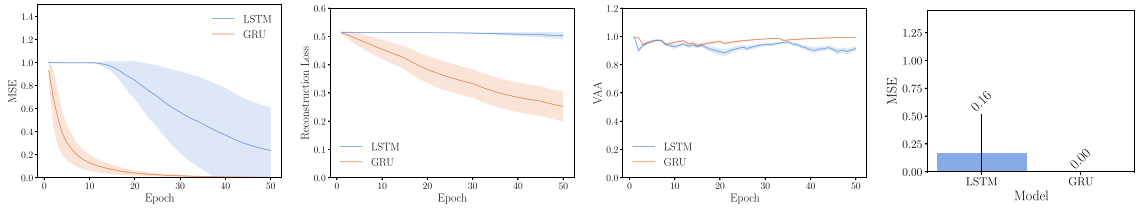
Fig. H.33, Fig. H.35 and Fig. H.37 show the evolutions of the reconstruction loss and the VAA on the three benchmarks during the pretraining, while Fig. H.34, Fig. H.36 and Fig. H.38 show the evolution of the validation MSE (or accuracy on validation set for the permuted-line sequential MNIST), the reconstruction loss and the VAA during training as well as the testing MSE/accuracy.

Globally, adding this auxiliary loss promotes multistability: the VAA increases during the pretraining, but not in all cases. The RNNs have more difficult to decrease the reconstruction loss when the sequences are longer, i.e. when there is more noise. However, this method increases the performances, especially when the sequences are not too long. For instance, GRU achieves very good results on the denoising benchmark, no matter the forgetting period. Concerning the permuted sequential-line MNIST benchmark, the models manage to obtain very low reconstruction losses. This is due to the padding of black pixels, which can be easily reconstructed. However, GRU managed to learn on this benchmark, which indicates that the auxiliary loss has been useful.

To conclude this section, adding an auxiliary loss to teach the models to reconstruct and thus to correctly encode the input sequences has led to good results. Indeed, that method has also been shown to promote multistability. However, its big drawback is its duration: for each experiment, we have run 20 pretraining epochs, which can take a lot of time especially for big datasets like MNIST. In comparison, in the experiments of Section 6.2, each warmup consisted of 100 gradient steps, which is much more lightweight. Also, this approach leads to equivalent or worse results than the warmup, depending on the benchmarks and the sequence lengths. Finally, concerning the benchmarks, it is important to make a distinction: in Trinh et al. (2018), this method has been tested on benchmarks with long sequences but where each timestep contain information. It is quite different from the benchmarks we have used here where large portions of the sequences consist of noise.

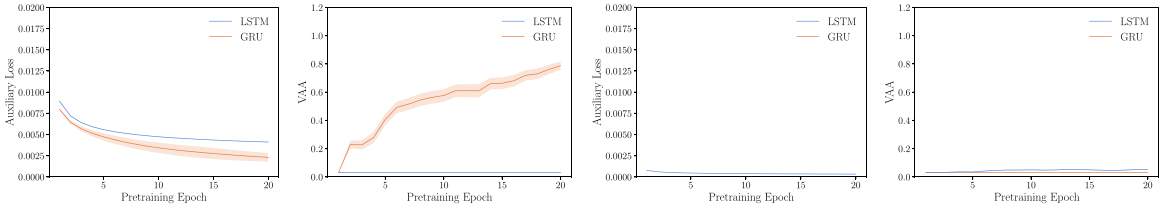


(a) $N = 5$



(b) $N = 100$

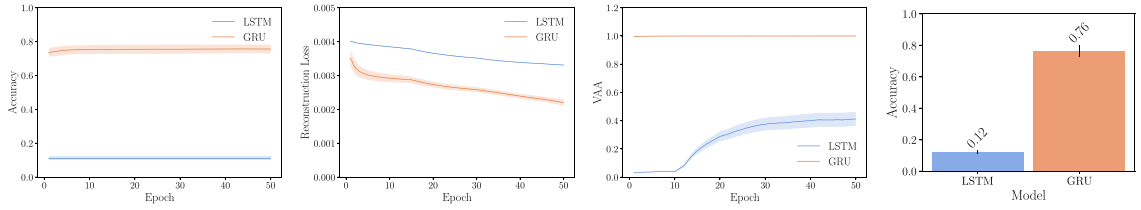
Fig. H.36. retraining on the denoising benchmark for different forgetting periods and $T = 200$. The reconstruction loss is evaluated on 150 timesteps.



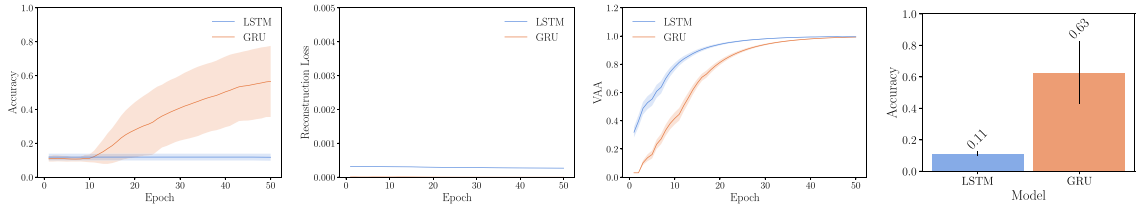
(a) $N = 72$ and the reconstruction loss is evaluated on 50 timesteps.

(b) $N = 472$ and the reconstruction loss is evaluated on 350 timesteps.

Fig. H.37. Pretraining on the permuted-line sequential MNIST benchmark for different forgetting periods.



(a) $N = 72$ and the reconstruction loss is evaluated on 50 timesteps.



(b) $N = 472$ and the reconstruction loss is evaluated on 350 timesteps.

Fig. H.38. Training on the permuted-line sequential MNIST benchmark for different forgetting periods.

References

Bakker, B. (2001). Reinforcement learning with long short-term memory. *Advances in Neural Information Processing Systems*, *14*.

Bengio, Y., Frasconi, P., & Simard, P. (1993). The problem of learning long-term dependencies in recurrent networks. In *IEEE International Conference on Neural Networks* (pp. 1183–1188). IEEE.

Ceni, A., Ashwin, P., & Livi, L. (2020). Interpreting recurrent neural networks behaviour via excitable network attractors. *Cognitive Computation*, *12*(2), 330–356.

Chen, J., & Chaudhari, N. (2009). Segmented-memory recurrent neural networks. *IEEE Transactions on Neural Networks*, *20*(8), 1267–1280. http://dx.doi.org/10.1109/TNN.2009.2022980, URL http://ieeexplore.ieee.org/document/5164893/.

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259.

Chung, J., Ahn, S., & Bengio, Y. (2017). Hierarchical multiscale recurrent neural networks. http://dx.doi.org/10.48550/arXiv.1609.01704, URL arXiv:1609.01704[cs].

Doya, K. (1993). Bifurcations of recurrent neural networks in gradient descent learning. *IEEE Transactions on Neural Networks*, *1*(75), 218.

Hausknecht, M., & Stone, P. (2015). Deep recurrent Q-learning for partially observable MDPs. In *2015 AAAI Fall Symposium Series*.

Hihi, S., & Bengio, Y. (1995). Hierarchical recurrent neural networks for long-term dependencies. *8*, In *Advances in Neural Information Processing Systems*. MIT Press, URL https://proceedings.neurips.cc/paper/1995/hash/c667d53acd899a97a85de0c201ba99be-Abstract.html.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Ienco, D., Interdonato, R., & Gaetano, R. (2019). Supervised level-wise pretraining for recurrent neural network initialization in multi-class classification. URL arXiv:1911.01071[cs, stat].

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*(1–2), 99–134.

Katz, G. E., & Reggia, J. A. (2017). Using directional fibers to locate fixed points of recurrent neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(8), 3636–3646.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Koutnik, J., Greff, K., Gomez, F., & Schmidhuber, J. (2014). A clockwork RNN. In *Proceedings of the 31st International Conference on Machine Learning* (pp. 1863–1871). PMLR, URL https://proceedings.mlr.press/v32/koutnik14.html, ISSN: 1938-7228.

Lin, T., Horne, B. G., Tino, P., & Giles, C. L. (1996). Learning long-term dependencies in NARX recurrent neural networks. *IEEE Transactions on Neural Networks*, *7*(6), 1329–1338. http://dx.doi.org/10.1109/72.548162.

Maheswaranathan, N., Williams, A., Golub, M., Ganguli, S., & Sussillo, D. (2019). Reverse engineering recurrent networks for sentiment classification reveals line attractor dynamics. *Advances In Neural Information Processing Systems*, *32*.

Menezes, J. M. P., & Barreto, G. A. (2008). Long-term time series prediction with the NARX network: An empirical evaluation. *Neurocomputing*, *71*(16), 3335–3343. http://dx.doi.org/10.1016/j.neucom.2008.01.030, URL https://www.sciencedirect.com/science/article/pii/S0925231208003081.

Mikolov, T., Joulin, A., Chopra, S., Mathieu, M., & Ranzato, M. (2015). Learning longer memory in recurrent neural networks. URL arXiv:1412.7753[cs].

Ong, B. T., Sugiura, K., & Zettsu, K. (2014). Dynamic pre-training of deep recurrent neural networks for predicting environmental monitoring data. In *2014 IEEE International Conference on Big Data (Big Data)* (pp. 760–765). Washington, DC, USA: IEEE, http://dx.doi.org/10.1109/BigData.2014.7004302, URL http://ieeexplore.ieee.org/document/7004302/.

Pasa, L., & Sperduti, A. (2014). Pre-training of recurrent neural networks via linear autoencoders. *27*, In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., URL https://proceedings.neurips.cc/paper/2014/hash/f0fcf351df4eb6786e9bb6fc4e2dee02-Abstract.html.

Pasa, L., Testolin, A., & Sperduti, A. (2015). Neural networks for sequential data: a pre-training approach based on hidden Markov models. *Neurocomputing*, *169*, 323–333. http://dx.doi.org/10.1016/j.neucom.2014.11.081, URL https://linkinghub.elsevier.com/retrieve/pii/S0925231215003689.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning* (pp. 1310–1318). PMLR.

Porta, J. M., Spaan, M. T., & Vlassis, N. (2004). Value iteration for continuous-state POMDPs.

Sagheer, A., & Kotb, M. (2019). Unsupervised Pre-training of a deep LSTM-based stacked autoencoder for multivariate time series forecasting problems. *Scientific Reports*, *9*(1), 19038. http://dx.doi.org/10.1038/s41598-019-55320-6, URL https://www.nature.com/articles/s41598-019-55320-6, Number: 1 Publisher: Nature Publishing Group.

Smallwood, R. D., & Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, *21*(5), 1071–1088.

Sussillo, D., & Barak, O. (2013). Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural Computation*, *25*(3), 626–649.

Tallec, C., & Ollivier, Y. (2018). Can recurrent neural networks warp time? In *International Conference on Learning Representations*.

Tang, Z., Wang, D., & Zhang, Z. (2016). Recurrent neural network training with dark knowledge transfer. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 5900–5904). Shanghai: IEEE, http://dx.doi.org/10.1109/ICASSP.2016.7472809, URL http://ieeexplore.ieee.org/document/7472809/.

Trinh, T. H., Dai, A. M., Luong, M.-T., & Le, Q. V. (2018). Learning longer-term dependencies in RNNs with auxiliary losses. URL arXiv:1803.00144[cs, stat].

Van Der Westhuizen, J., & Lasenby, J. (2018). The unreasonable effectiveness of the forget gate. arXiv preprint arXiv:1804.04849.

Vecoven, N., Ernst, D., & Drion, G. (2021). A bio-inspired bistable recurrent cell allows for long-lasting memory. *PLoS One*, *16*(6), Article e0252676.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

Williams, R. J., & Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. In *Backpropagation: theory, architectures, and applications* (pp. 433–486). USA: L. Erlbaum Associates Inc..

Zhou, G.-B., Wu, J., Zhang, C.-L., & Zhou, Z.-H. (2016). Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing*, *13*(3), 226–234.