Parallelizable memory recurrent units

Florent De Geeter[1,2], Gaspard Lambrechts[1], Damien Ernst[1], and Guillaume Drion[1]

[1]Montefiore Institute, University of Liege, Liege, Belgium
[2]florent.degeeter@uliege.be

Abstract

With the emergence of massively parallel processing units, parallelization has become a desirable property for new sequence models. The ability to parallelize the processing of sequences with respect to the sequence length during training is one of the main factors behind the uprising of the *Transformer* architecture. However, Transformers lack efficiency at sequence generation, as they need to reprocess all past timesteps at every generation step. Recently, *state-space models* (SSMs) emerged as a more efficient alternative. These new kinds of recurrent neural networks (RNNs) keep the efficient update of the RNNs while gaining parallelization by getting rid of nonlinear dynamics (or recurrence). SSMs can reach state-of-the art performance through the efficient training of potentially very large networks, but still suffer from limited representation capabilities. In particular, SSMs cannot exhibit persistent memory, or the capacity of retaining information for an infinite duration, because of their monostability. In this paper, we introduce a new family of RNNs, the *memory recurrent units* (MRUs), that combine the persistent memory capabilities of nonlinear RNNs with the parallelizable computations of SSMs. These units leverage multistability as a source of persistent memory, while getting rid of transient dynamics for efficient computations. We then derive a specific implementation as proof-of-concept: the bistable memory recurrent unit (BMRU). This new RNN is compatible with the parallel scan algorithm. We show that BMRU achieves good results in tasks with long-term dependencies, and can be combined with state-space models to create hybrid networks that are parallelizable and have transient dynamics as well as persistent memory.

1 Introduction

Recurrent neural networks (RNNs) [1, 2] used to be the most popular architecture for tackling sequential tasks thanks to their ability to model a large range of dynamical systems, until being foreshadowed by the Transformer architecture [3]. The impossibility to parallelize RNNs with respect to time for training made Transformers a much suited architecture for building large models and training them on large amount of data. However, for generative modeling tasks, the Transformer suffers from a high computational complexity. Indeed, at every timestep of generation, it needs to process the full sequence of past inputs. RNNs have however regained interest in recent years thanks to the success of state-space models (SSMs) [4]. SSMs use strictly linear dynamics in cascade with static nonlinearities, making their computation easily parallelizable. This allows SSMs to be used in large networks and to be trained on large datasets, because they can benefit from the computational power offered by GPUs. SSMs can compete with Transformers on a variety of tasks [4], and large-language models based on SSMs like Mamba [5] have achieved comparable or better performance than Transformers for similar model sizes. SSMs are more efficient than Transformers during inference, as their recurrent formulation allows them to compute each output only from their previous output and the input, while Transformers require the whole sequence of inputs to be fed (or cached activations along the whole sequence of inputs).

The strictly linear recurrent dynamics of SSMs however creates some limitations: SSMs can only approximate fading-memory systems, i.e. systems that have a unique stable state [6]. Monostable systems such as SSMs encode past information in their transient dynamics, which creates a memory that ineluctably fades over time and makes some tasks unsolvable for a fixed number of layers [7].

Alternatively, multistable systems have several stable states, which allows to encode information for an infinite duration. It is possible to foster multistability in RNNs [8, 9], and multistability has been shown to

drastically improve RNN performance especially in long-term dependencies tasks [9]. Multistability however requires strongly nonlinear recurrent connections, which usually prevents parallelization. In this configuration, multistable RNNs cannot compete with parallelizable alternatives such as SSMs or Transformers.

This work attempts to close this gap through the creation of novel RNN models that are parallelizable over the sequence length but exhibit persistent memory through multistability. Section 2 first introduces the concept of RNNs with internal clocks, i.e. RNNs in which internal states can be updated several times between two consecutive timesteps, as a way to formalize our design approach. Section 3 focuses on the RNN class where infinite updates can occur between two consecutive timesteps. From that, it introduces the concept of *memory recurrent unit* (MRU), a class of RNNs that exhibit persistent memory but no fading memory. Section 4 further introduces *bistable memory recurrent unit* (BMRU), a concrete example of MRU that can be trained on sequential tasks. BMRU properties and performance are finally evaluated on several benchmarks in section 5.

2 RNNs with internal clocks

RNNs encode time-dependencies in a recurrent hidden state h_t whose update depends on the previous hidden state h_{t-1} and current observation of a time-series x_t. It writes

$$h_t = f_\theta \left(x_t, h_{t-1}; \theta \right), \forall t \geq 1 \tag{1}$$

where f_θ represents the update equation of the RNN and θ the parameters of the network. The key mechanism is the recurrent connection that defines a dependency between h_t and h_{t-1}. Nonlinear RNNs such as LSTM [1] and GRU [2] have remained state-of-the-art methods in sequence modeling for several decades.

The classical RNN formulation of equation (1) constraints the internal state h_t to only be updated when the network receives a new input x_t, i.e. only once every timestep t. This constraint forbids the network to modify its internal state evolution between two timesteps. It particularly impedes our goal to quickly encode information on stable equilibria at convergence.

To release this constraint, we define a class of RNNs whose internal dynamics are decoupled from the external dynamics of the input time-series. Each RNN of this class has internal clocks that permit to update their internal states N-times between two timesteps t, where N can be different for each unit. The internal update equation of such RNN unit between two timesteps writes

$$\tilde{h}_t[0] = h_{t-1}, \tag{2a}$$

$$\tilde{h}_t[n] = f_\theta(x_t, \tilde{h}_t[n-1]; \theta), \forall n \geq 1, \tag{2b}$$

$$h_t = \tilde{h}_t[N], \tag{2c}$$

where t is the current timestep, $n \in [0, \dots, N]$ the current internal clock iteration and $N \in \mathbb{N}$ the number of internal clock iterations. $\tilde{h}_t[n]$ is the transient internal state value at iteration n and h_t the final internal state at time t.

Classical RNNs are contained within this class. Indeed if we set $N = 1$, equation (2b) becomes

$$\tilde{h}_t[1] = f_\theta(x_t, \tilde{h}_t[0]; \theta)$$

with $\tilde{h}_t[0] = h_{t-1}$ and $\tilde{h}_t[1] = h_t$, which leads to equation (1).

For $1 < N < \infty$, we have the set of RNNs whose internal dynamics are decoupled from the time-series dynamics, as their internal state is updated N times between two timesteps. Such RNNs can be trained as classical RNNs, but the potential increase in expressive power comes at a computational cost, as the effective sequence length becomes $N \cdot T$ at inference, T being the length of the time-series. Using linear recurrence of the form

$$f_\theta(x_t, \tilde{h}_t[n]; \theta) = A(\theta)\tilde{h}_t[n-1] + B(\theta)x_t,$$

can mitigate this complexity, as the value $\tilde{h}[N]$ can be computed analytically from $\tilde{h}[0]$ while maintaining the possibility to output potentially complex trajectories. This is for instance the approach taken by practical implementations of state-space models (SSMs)[10].

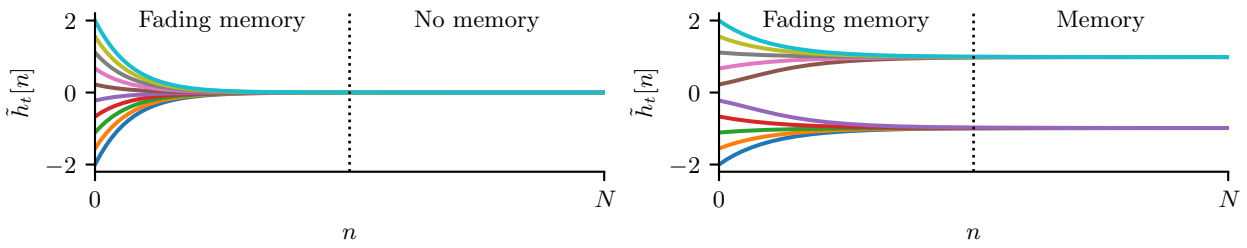


Figure 1: **Monostability vs bistability in a RNN with internal clock**. The figure shows internal state trajectories of the RNN unit described by equation (3) for different initial conditions $\tilde{h}_t[0] = h_{t-1}$. (**left**) Evolution of the system when $\beta = -1.5$. (**right**) Evolution of the system when $\beta = 1.5$.

For $N \to \infty$, we have the specific set of "convergent" RNNs, i.e. RNNs that can converge towards their steady-state values between two timesteps. In this work, we use the framework of convergent RNNs to build a novel type of RNN unit that uses stable states as the basis for their computations.

3 The specific case of convergent RNNs ($N \to \infty$)

We focus on RNNs that converge towards stable equilibria, i.e. RNNs whose update equation $f_\theta(x_t, h_{t-1}; \theta)$ is such that:

$$\forall x_t, \tilde{h}_t[0],\ \exists h^* \in \mathbb{R} : \lim_{N\to\infty} f_\theta^N(x_t, \tilde{h}_t[0]; \theta) = h^*,$$

where:

$$f_\theta^n(x_t, \tilde{h}_t[0]; \theta) = \begin{cases} f_\theta(x_t, f_\theta^{n-1}(x, \tilde{h}_t[0]; \theta); \theta) & \text{if } n > 1, \\ f_\theta(x_t, \tilde{h}_t[0]; \theta) & \text{if } n = 1. \end{cases}$$

Letting the internal clock to loop an infinite number of times ($N \to \infty$) ensures that the RNN has converged towards a stable equilibrium between two time steps. This property, which is specific to this class of RNNs, ensures a stationary output, i.e. that the output remains stable over time if the input barely changes. This property is of great interest, as it ensures that the information encoded in the RNN unit will not fade over time. It however requires specific characteristics to be exploitable in practice.

To illustrate this, we consider the following RNN update equation:

$$h_t = (1 - c) \cdot h_{t-1} + c \cdot \tanh\left(x_t + (\beta + 1) \cdot h_{t-1}\right), \forall t \geq 1, \tag{3}$$

which is a simplified version of the BRC and nBRC derived by Vecoven et al. [8]. To construct the convergent version, we rewrite equation (3) at timestep t with an internal clock

$$\tilde{h}_t[0] = h_{t-1} \tag{4a}$$

$$\tilde{h}_t[n] = (1 - c) \cdot \tilde{h}_t[n-1] + c \cdot \tanh\left(x_t + (\beta + 1) \cdot \tilde{h}_t[n-1]\right), \forall n \geq 1 \tag{4b}$$

$$h_t = \tilde{h}_t[N] \tag{4c}$$

and simulate its response over several internal clock iterations N for different values of previous state value h_{t-1} and two different values of the parameter β (figure 1) (other parameter values are $c = 0.01$ and $x_t = 0$). In both simulations, the trajectories corresponding to different values of h_{t-1} are transiently distinct, until they all converge towards a steady-state. The distinct transient trajectories create fading memory, which can encode quantitative information about h_{t-1} but only for a strictly finite amount of time.

Here, we focus our interest on what happens at convergence. In the case of $\beta = -1.5$ (figure 1, left), all initial h_{t-1} lead to the same equilibrium point, and the RNN unit cannot maintain any stationary information about h_{t-1}. It is restricted to fading memory. This is because equation (3) for $\beta = -1.5$ leads to a monostable system, i.e. a system that has only one stable equilibrium for all input values x_t. Monostable RNNs are incapable of encoding stationary (i.e. persistent) information about the past. It includes all RNNs with

linear recurrence, such as SSMs, as well as all gated RNNs in which gates do not depend on past state values and whose update gate z_t is strictly larger than 0 (for the formulation $(1 - z_t) \odot h_{t-1}$) or smaller than 1 (for the formulation $z_t \odot h_{t-1}$)[11, 12, 13].

In the case of $\beta = 1.5$ (figure 1, right), the trajectories converge towards two different equilibria depending on h_{t-1}: $h_t \simeq +1$ for positive values of h_{t-1} and $h_t \simeq -1$ for negative values of h_{t-1}. The RNN unit maintains a stationary, qualitative information about h_{t-1}. In order words, some information about h_{t-1} is encoded in *persistent memory*. This is because equation (3) for $\beta = 1.5$ leads to a bistable system, i.e. a system that has two stable equilibria for a set of input values x_t.

Constructing a multistable RNN with infinite internal clock cycles would therefore permit to isolate its persistent memory capabilities. But constructing such RNN by looping a large number of times in equation (2b) would drastically slow down inference and breach the convergence guarantee. We rather exploit the fact that the internal state converges towards an equilibria at $N \to \infty$, i.e. $\tilde{h}_t[n] = \tilde{h}_t[n-1] = h_t$. To do so, we rewrite an equivalent formulation for equation (4b) at the convergence point

$$\begin{aligned} \tilde{h}_t[0] &= h_{t-1} \\ h_t &= (1-c) \cdot h_t + c \cdot \tanh\left(x_t + (\beta+1) \cdot h_t\right), \end{aligned}$$

which can be rewritten

$$F(h_t, x_t; \beta) = 0; \quad \tilde{h}_t[0] = h_{t-1} \tag{5}$$

where $F(h_t, x_t; \beta) = h_t - \tanh\left(x_t + (\beta+1) \cdot h_t\right)$. Equation (5) provides a set of necessary conditions that the RNN unit must satisfy at convergence. This equation defines an implicit function that can have multiple solutions h_t for a given input x_t^*. The solution that corresponds to the output at time t depends on the value of h_{t-1}, which dictates towards which stable point h_t^* the cell will converge (i.e., h_{t-1} determines in which basin of attraction the trajectory lies). This solution must also satisfy the stability property to ensure that it is a stable point:

$$\left.\frac{df(x_t, h_t; \beta)}{dh_t}\right|_{x_t^*, h_t^*} < 1$$

where $f(x_t, h_t; \beta) = (1-c) \cdot h_t + c \cdot \tanh\left(x_t + (\beta+1) \cdot h_t\right)$.

The solutions of equation (5) and their stability are shown in figure 2 for $\beta = -1.5$ (A) and $\beta = 1.5$ (B). For $\beta = -1.5$, the function has only one solution h_t for all x_t, and the convergent RNN has a similar input-output function as a static layer. For $\beta = 1.5$, the function has a memory region where two solutions are possible, and the selected solution h_t depends on h_{t-1}. The RNN has encoded some information of the past input in persistent memory.

More generally, the set of necessary conditions that any RNN defined by equation (2b) must satisfy at convergence are determined by the implicit function

$$F_\theta(x_t, h_t; \theta) = 0, \tag{6}$$

where $F_\theta(x_t, h_t; \theta) = h_t - f_\theta(x_t, h_t; \theta)$. It is therefore possible to efficiently compute the outputs of a convergent RNN at each timestep t by using this implicit function as the update function

$$F_\theta(x_t, h_t; \theta) = 0, \forall t = 1, ..., T, \tag{7}$$

where h_{t-1} is used to select the output value when multiple solutions exist. This leads to a recurrent unit that only encodes information in persistent memory. We call this type of recurrent unit a **memory recurrent unit (MRU)**. It is important to note that equation (7) can be solved in parallel for each timestep, as the function only depends on x_t and h_t, but requires h_{t-1} to select the proper solution based on history.

In dynamical systems theory, the implicit function $F_\theta(x_t, h_t; \theta) = 0$ corresponds to the bifurcation diagram of the system described by equation (2b). For instance, equation (5) and associated figure 2 correspond to a hysteresis bifurcation. Hysteresis bifurcations underlie bistability in existing RNN cells such as the bistable recurrent cell (BRC) [8] (see appendix A). More generally, a MRU can be built from any nonlinear dynamics

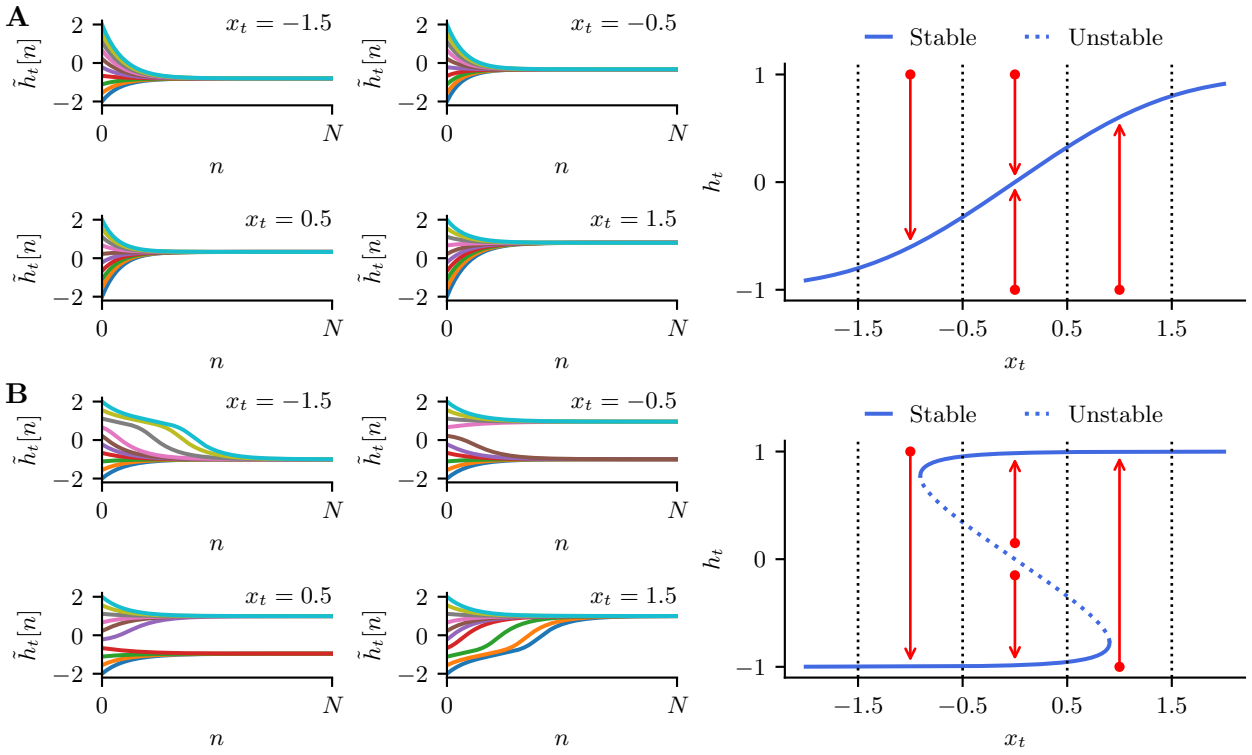


Figure 2: **Convergence properties of the RNN unit described by equation (4b) for different values of input x_t and either $\beta = -1.5$ (A) or $\beta = 1.5$ (B)** (**left**) Internal state trajectories corresponding to different initial conditions $\tilde{h}_t[0] = h_{t-1}$ and 4 different input values x_t. (**right**) Solutions of the steady-state equation equation (5) for $\beta = -1.5$ (**A**) and $\beta = 1.5$ (**B**). The red arrows show convergence trajectories from different initial conditions $\tilde{h}_t[0]$.

by using its bifurcation diagram as an implicit activation function. The output state in the memory region can be computed using iterative methods such as the Newton–Raphson method [14, 15, 16]. Likewise, implicit differentiation can be used to compute exact gradients during backpropagation. The use of iterative methods can however slow down inference and hinder parallelization. In the next section, we show how this drawback can be circumvented, focusing on the hysteresis bifurcation and its associated bistability.

4 A bistable memory recurrent unit

4.1 A computationally efficient hysteresis-based MRU

Instead of directly using the implicit function derived by the hysteresis bifurcation diagram, we propose to create a MRU based on an approximation of this function that maintains all its qualitative properties, but which is computationally efficient and remain parallelizable. The goal of this section is to present this approximation, and to show how we build a bistable memory recurrent unit (BMRU) around it.

We focus our design on the multistable behavior of the implicit function, i.e. $\beta > 0$, as $\beta \leq 0$ leads to a monostable, memory-less function (see figure 2). Under this condition, the original function can be decomposed into three parts: the upper stable points, the lower stable points, and the unstable points that serve as a boundary in the bistable region. We approximate these three parts by modeling the stable points as constant values $\pm\alpha$, and the *boundary* function as a linear function whose slope is the slope of the original function at $(0, 0)$, i.e. $-\frac{1}{\beta}$. The approximation writes

$$h_t = \begin{cases} \alpha \cdot S(x_t) & \text{if } |x_t| \geq \beta, \\ \alpha \cdot S(h_{t-1} + \frac{x_t}{\beta}) & \text{if } |x_t| < \beta, \end{cases} \tag{8}$$

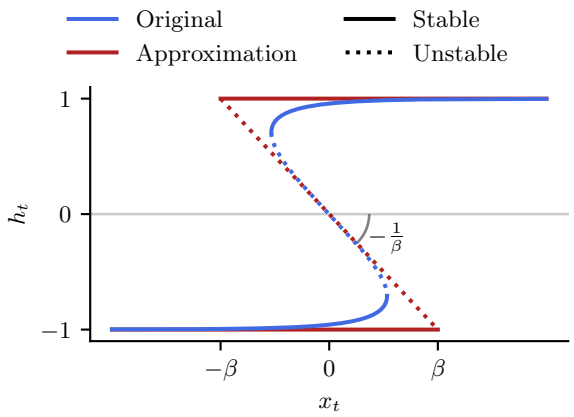


Figure 3: **Comparison between the implicit function and its approximation.** This figure compares the solutions (h_t, x_t) of the implicit function defined by equation (5) (in blue) with the approximation defined by equation (8) (in red) where $\alpha = 1$. Solid lines correspond to stable points, and dashed lines to unstable points.

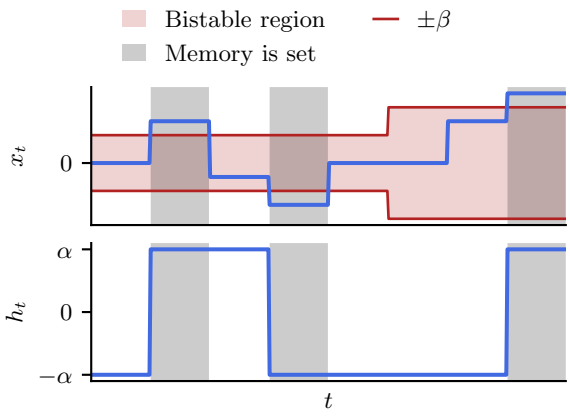


Figure 4: **Example simulation of the cell defined by equation (9) for different values of x_t and β.** The top graph defines two illustrative variations of x_t and β, as well as the bistable region defined by β. The bottom graph shows the effect of variations in x_t and β on the evolution of the state h_t. The gray areas highlight the timesteps at which the memory is updated, which occurs when x_t is outside of the bistable region.

where S denotes the *sign* function (with $S(0) = 1$). Similarly to the original function, h_t is independent of h_{t-1} for a large input, but not for a small input. A comparison between this approximation and the original function is shown in figure 3.

The convergence condition ensures that $h_{t-1} = \pm\alpha$ for all timesteps, which permits to further simplify equation (8) for $|x_t| < \beta$:

$$h_t = \begin{cases} \alpha \cdot S(x_t) & \text{if } |x_t| \geq \beta, \\ h_{t-1} & \text{if } |x_t| < \beta. \end{cases} \tag{9}$$

It is only every time $|x_t|$ is greater than β that h_t is updated. Otherwise, it will remain constant. This property is illustrated in figure 4.

4.2 A learnable bistable memory recurrent unit

Equation (9) defines the input-output properties of the BMRU, but does not contain any learnable parameters. Here, we add such parameters by taking inspiration from gated RNN structure.

First, we observe that the function defined by equation (9) outputs two different values depending on the comparison between the values of $|x_t|$ and β. We can implement this function by introducing a *binary* gate, z_t, that computes this condition, and rewrite equation (9) as a gate-dependent update rule. It writes

$$z_t = H(|x_t| - \beta), \tag{10a}$$

$$h_t = z_t \cdot S(x_t) \cdot \alpha + (1 - z_t) \cdot h_{t-1}, \tag{10b}$$

where t is the current timestep, H the Heaviside function and S the sign function. Equation (10a) computes the binary gate value z_t, which is used to distinguish whether we are inside the bistable region ($z_t = 0$) or outside of it ($z_t = 1$). Equation (10b) then computes the new state value h_t depending on the value of z_t.

We can then extend this formulation to the multidimensional case, where multiple inputs x_t converge at a layer composed of multiple BMRUs. To be coherent with the variables used in gated RNNs, we name the input values x_t and their combination at each BMRU cell $\hat{h}_t$, the *candidate*. x_t becomes a M-dimensional vector where M is the number of inputs to the layer. Moreover, β can be made input-dependent, and is therefore renamed β_t. $\hat{h}_t$, β_t and h_t become N-dimensional vectors, where N is the number of BMRU cells

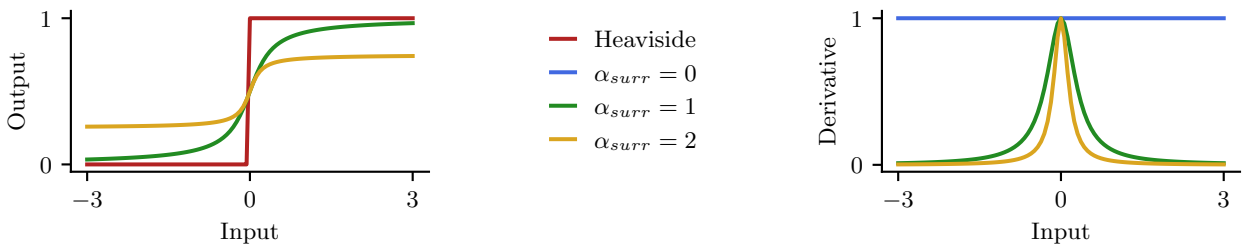


Figure 5: **Surrogate gradient used in BMRU.** (**left**) Comparison between the Heaviside function and the function defined by equation (13) used to approximate the gradient for different values of α_{surr}. (**right**) Impact of α_{surr} on the surrogate gradient defined by equation (12).

in the layer. We use a classical fully connected layer to compute the vectors $\hat{h}_t$ and β_t from the input vector x_t, adding the positivity constraint for β_t. The equations of the multidimensional BMRU network write

$$\hat{h}_t = W_x x_t + b_x, \tag{11a}$$
$$\beta_t = |W_\beta x_t + b_\beta|, \tag{11b}$$
$$z_t = H(|\hat{h}_t| - \beta_t), \tag{11c}$$
$$h_t = z_t \odot S(\hat{h}_t) \odot \alpha + (1 - z_t) \odot h_{t-1}, \tag{11d}$$

where $\odot$ is the hadamard product, t is the current timestep, W_x and W_β are matrices of learnable parameters, b_x, b_β and α are learnable parameters, H is the Heaviside function and S is the sign function.

The use of S and H makes the use of backpropagation difficult, as their gradient is 0 everywhere except in 0 where it is ∞. However, there exist solutions to overcome this problem, and we chose the *surrogate gradient* approach: the non-differentiable functions are used in the forward pass, but the derivatives of other functions, which are differentiable and similar to the non-differentiable ones, are used in the backward pass. This technique is notably used in the context of *spiking neural networks* [17, 18].

For the Heaviside function, we chose the following surrogate gradient (inspired from [18])

$$\frac{df}{dx}(x) = \frac{1}{1 + (\alpha_{surr}\pi x)^2}, \tag{12}$$

where α_{surr} is a tunable parameter. We note that this surrogate derivative comes from the function

$$f(x) = \frac{1}{\pi\alpha_{surr}}\text{atan}(\alpha_{surr}\pi x) + \frac{1}{2}. \tag{13}$$

Figure 5 plots $f(x)$ and $\frac{df}{dx}(x)$ for different values of α_{surr}. The derivative exhibits a localized peak centered at x = 0. As the parameter α_{surr} increases, the width of this peak decreases, making it more concentrated. The case $\alpha_{surr} = 0$ is special and leads to a constant derivative equal to 1, which is also called the *straight-through estimator* [19]. The surrogate gradient used for S is simply $2 \cdot \frac{df}{dx}(x)$, as S has the same shape as H with outputs being either -1 or 1.

4.3 Properties of the bistable MRU

BMRU is a special recurrent cell compared to the usual ones and it has some interesting properties that are highlighted here.

Stationarity. BMRU is by design stationary, meaning that repeating the same input several times will not impact state value. It makes state update independent of input duration when the input is constant. One advantage that comes from this stationarity property is the ability to generalize to longer inputs. Figure 6 illustrates this on a simple benchmark, where the goal is to retain a binary value ± 1 given at $t = 0$ for some time, during which the input is 0. This benchmark can be easily solved using either persistent memory

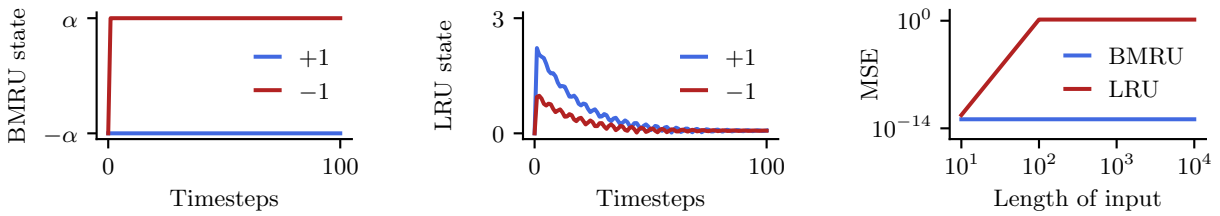


Figure 6: **Persistent memory of BMRU compared to the fading memory of LRU.** Small models with one LRU or BMRU layer of one unit have been trained on a simple benchmark whose inputs start with ±1 followed by 0's. The goal of the models is to output the first input at the last timestep. (**left, center**) Evolution of the states of LRU and BMRU with respect to the timesteps for the two possible inputs. As the LRU state is complex, its norm is plotted. (**right**) MSE computed with both models on the two possible inputs for different sequence lengths.

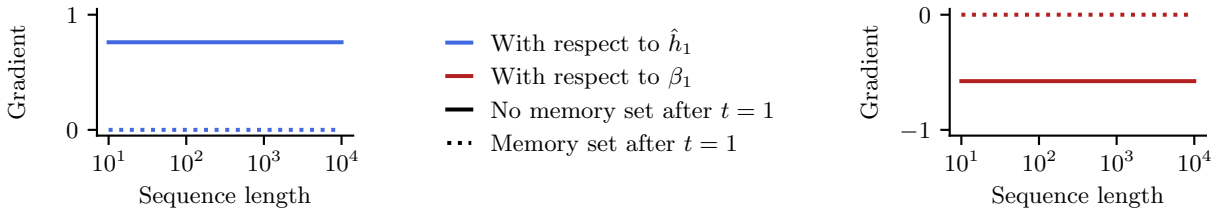


Figure 7: **Consistency of the gradient of BMRU with respect to time.** Evolution of the gradient of the last state h_T with respect to the first candidate $\hat{h}_1$ or β_1. Two cases are considered: either only the first timestep sets the memory, i.e. $|\hat{h}_1| > \beta_1$ and $|\hat{h}_t| < \beta_t \; \forall t > 1$, either another timestep also sets the memory.

(figure 6, left) or fading memory (figure 6, center). However, BMRU encoding the information in stable states, it remains encoded forever, and the performance is not impacted by the input length (figure 6, right). LRU however encodes the information in fading memory, which does not generalize to larger input length, as memory fades over time.

No vanishing nor exploding gradient in the memory region. At every timestep, the state h_t will be either set equal to h_{t-1} or to a value independent of h_{t-1}. When the previous state is kept, the derivative of the new state with respect to the previous one is 1. When the memory is updated, the derivative is 0, and therefore the gradient will not go further back in time during the backward pass. In other words, the information is either kept intact, or overwritten. Mathematically, it writes

$$\frac{\partial h_t}{\partial h_{t-1}} = \begin{cases} 1 & \text{if } z_t = 0, \\ 0 & \text{if } z_t = 1. \end{cases}$$

Figure 7 shows how the gradient of the last state h_T with respect to the first candidate $\hat{h}_1$ or the first β_1 evolves with respect to the sequence length. As expected, it stays constant, with two possible values: a null one if the memory has been updated at a later timestep, or a non-zero value if the memory was never updated after the first timestep. Indeed, in this case, each derivative $\frac{\partial h_t}{\partial h_{t-1}}$ is equal to 1, therefore their product is equal to 1, ensuring a constant gradient across time.

Compatibility with parallel scan. BMRU update equations can be rewritten using an associative operator, therefore allowing the use of the parallel scan, as proved in appendix B. A *scan* is an operation that takes a binary operator $\circledast$ and an ordered set of n elements $[a_0, ..., a_{n-1}]$ and returns the ordered set

$$[a_0, (a_0 \circledast a_1), (a_0 \circledast a_1 \circledast a_2), \ldots, (a_0 \circledast a_1 \circledast \ldots \circledast a_{n-1})] \,.$$

The goal of the parallel scan is to perform a scan with a better time complexity than by performing it sequentially, decreasing the complexity of computing the n outputs from $O(n)$ to $O(\log(n))$ on a GPU.

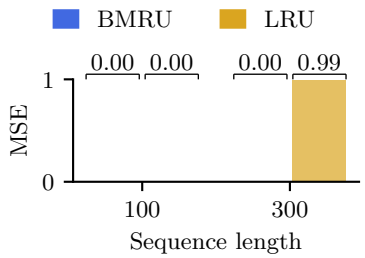


Figure 8: **Results on the copy-first input benchmark.** Test MSEs obtained by BMRU and LRU models on the copy-first-input benchmark for two sequence lengths: 100 and 300.

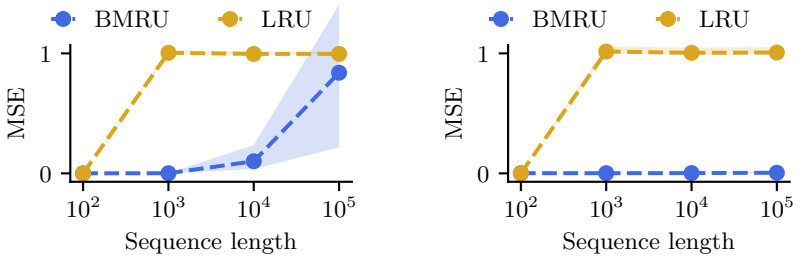


Figure 9: **Generalization capabilities of BMRU and LRU with respect to the sequence length.** MSEs obtained by the BMRU and LRU models trained on the copy-first-input with a sequence length of 100 when they are evaluated on longer sequences with two different levels of noise. (**left**) The noise is sampled from $\mathcal{N}(0, 1)$. (**right**) The noise is sampled from $\mathcal{N}(0, 0.1)$.

5 Experiments

This section aims at analyzing the performance of BMRU on several regression and classification benchmarks, with increasing level of difficulty. The idea is to showcase the properties, strengths but also the limitations of BMRU. Networks of LRU [20] were also trained to highlight differences between the SSMs and BMRU. Each value presented in this section is computed by taking the average over 5 runs.

5.1 Model architecture

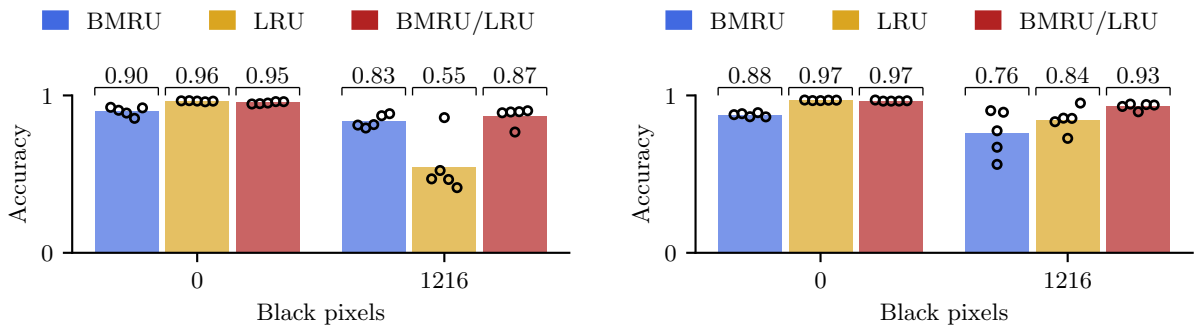
The architecture follows the one used in the SSM literature [4, 20]. It consists of a linear layer that projects the input to some dimensions (called the *model dimensions*, H), followed by recurrent blocks and fully-connected layers applied timestep-wise to extract the predictions. Each recurrent block has a batch normalization layer, recurrent cell (either BMRU or LRU), a GLU activation and a skip connection. These recurrent cells have their own number of dimensions (*state dimensions*, N), which can differ from the model dimensions. More specifically, they receive inputs in H dimensions, update their N dimensional state and generate an output with H dimensions. The recurrent cells can also receive positional embeddings that encode the current timestep into some vector of arbitrary dimensions. Unless it is specified otherwise, we consider the prediction of the model to be its output at last timestep, as our main goal is to evaluate the ability of the models to retain information. The parameters of each experiment are given in appendix C.

5.2 Copy-first-input

The first benchmark is purely synthetic and consists of remembering a real value for some duration while being perturbed by noise. In practice, models receive a two-dimensional input $x_t = (r_t, f_t)$ at each timestep. The first dimension is a real value r_t independently and identically distributed from N(0,1) at all timesteps. The second one is a flag f_t indicating if r_t has to be retained ($f_t = 1$) or not ($f_t = 0$). This flag is only set at the first timestep. Models are trained by computing the MSE between the first input and their last output. In this experiment, we used networks of 2 recurrent blocks, with $H = N = 256$ and without any positional encoding. All the models are trained on 60000 samples, 10% of which are used for validation. The test set also consists of 60000 samples.

Figure 8 shows the test MSE obtained by the networks on two versions of the benchmark, one where information has to be retained for 100 timesteps, the other for 300 timesteps. For the small duration, both models learn. However, for the longer duration, only BMRU manages to learn the task. Although increasing network depth would possibly make LRU learn, our goal here is not to optimize the hyperparameters to get the best loss possible, but rather to highlight the ability of BMRU to retain information for a long duration even with a shallow network.

This benchmark also allows to highlight generalization property of BMRU to longer sequences. One can evaluate this generalization by taking the models trained on 100 timesteps and testing them on longer



(a) 2 rec. blocks

(b) 3 rec. blocks

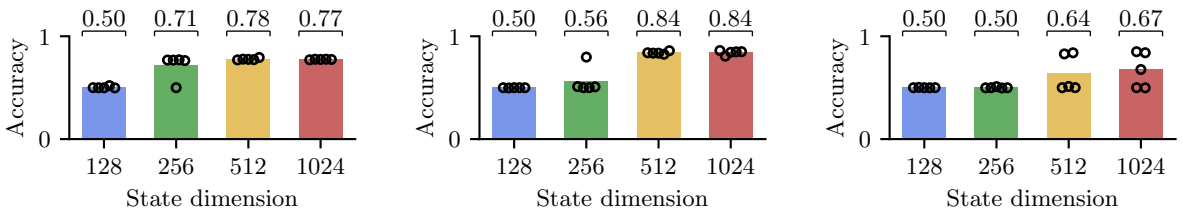
Figure 10: **Results on the permuted sequential MNIST.** Accuracies obtained by BMRU, LRU and BMRU/LRU models on the permuted sequential MNIST, with or without black pixels added at the end of the sequences. Models with 2 and 3 recurrent block have been tested.

sequences. To do that, we created small test sets whose sequences lengths are a power of 10, starting from 10^2 to 10^5. Each test set is composed of 6000 samples. Figure 9 shows the evolution of the MSEs obtained by both cells on these small test sets with respect to the sequences length. LRU does not generalize to larger sequences well, as it encodes the information in fading memory. BMRU is much more resilient as it encodes the information in persistent memory, and the performance only decreases if the stored values are overwritten by high amplitude noise (center graph). If the noise is sufficiently low, the performance is unaffected by the sequence length (right graph).

5.3 Permuted sequential MNIST

The MNIST dataset is one of the most well-known datasets [21]. In this experiment, we use a variant of MNIST, called the Sequential MNIST [22], where the pixels are fed to the models one by one. While being known as an *easy* benchmark, the Sequential MNIST is still relevant when testing new architectures. To increase the difficulty, the pixels are shuffled before being given to the models. Compared to the previous benchmark, models must combine information received at different timesteps in order to predict the correct label. To also assess the memorization capabilities of the models, we add, in some experiments, black pixels at the end of the sequences. This forces the models to not only combine temporal information but also to retain it for a long period. In practice, 1216 black pixels are added, bringing the sequence lengths to 2000. Furthermore, we add positional encodings to the inputs of BMRU, as we observed that it can greatly improve its performance. On the other hand, we observed that adding these encodings to LRU actually impedes its performance, therefore these are only given to BMRU in these experiments.

Figure 10 shows the accuracies obtained by BMRU (blue bars) and LRU models (yellow bars), with or without black pixels, and for two network depths. All recurrent blocks contain 256 neurons, as in the previous experiment. The performance of LRU is better than BMRU on the version without black pixels, but BMRU better maintains its performance when black pixels are added, especially for the more shallow model. LRU indeed requires more depth to handle longer dependencies, whereas BMRU only needs two layers to handle these long dependencies. On the other hand, this experiment also shows that LRU is much better at combining the information from the different timesteps. This motivates our next experiment: as LRU is better at combining the information using fading memory, and BMRU at retaining it using persistent memory, their combination should get the best of both worlds, while still being parallelizable. The results obtained using this combination are shown in Figure 10 (red bars). Note that for a fair comparison, the state dimension of each cell has been divided by 2, in order to maintain the number of neurons in each recurrent block at 256. We can see that the combination has the same performance than LRU alone when there are no black pixels, but its performance does not decrease when black pixels are added. It even stays higher than BMRU alone. This highlights the potential of combining the fading memory and persistent memory properties of these two types of parallelizable RNNs.



(a) 1 rec. block (b) 2 rec. blocks (c) 3 rec. blocks

Figure 11: **Results on the pathfinder benchmark.** Each plot shows the accuracies obtained by BMRU models on the pathfinder benchmark, for different state dimensions (x-axes) and network depths (1, 2 and 3 recurrent blocks from left to right, respectively).

5.4 Pathfinder

The pathfinder benchmark is part of the *long-range arena* group of benchmarks [23]. It consists of 32x32 black and white images where lines are drawn randomly, as well as two dots. The goal of the benchmark is to predict if the two dots are connected by a line or not. Images are fed pixel by pixel, which leads to sequences of 1024 timesteps. The goal of this experiment is to test the capabilities of shallow BMRU networks, i.e. max 3 recurrent blocks, to solve such a difficult benchmark without trying to achieve state-of-the-art performance through parameter tuning. We chose a similar model approach as in the S4 and LRU papers [4, 20]: the predictions of the models are the means of their outputs (instead of their last timestep), and the recurrent layers are bidirectional.

Figure 11 shows the accuracies obtained by BMRU on this benchmark. Note that as there are only two classes, so an accuracy of 50% corresponds to random guess. First, the results show that the BMRU is able to learn on the Pathfinder benchmark, even with only one recurrent layer. Second, these experiments show that the depth of the network is not that important, but its width, i.e. the state dimension, has a bigger impact on performance. This differs from SSMs, for which it is known that their ability to handle longer time dependencies grows with the network depth. The drop of performance when more layers are added can be explained by the *discretization* implied by BMRU and the simplicity of layer interconnections. Indeed, as each neuron of BMRU can only output two values ($\pm\alpha$), each BMRU layer adds more discretization, and makes the learning more difficult.

6 Discussion

The concept of MRU and BMRU are at the junction of several topics that are discussed in this section and compared to relevant works.

Multistable RNNs. Adding multistability in RNNs is not a very explored topic from the point of view of machine learning. Some works have been done to better understand multistability in RNNs from a dynamical system point of view [24, 25], even some that includes *thresholding* functions [26, 27]. However, these works do not include any machine learning experiments. Few works have highlighted how this property can improve the memorization capability of RNNs, either by building new types of RNNs [8] or by enforcing multistability in existing RNNs [9]. This work attempts to highlight the interesting properties of multistable RNNs for sequence learning and its complementarity with the more classical fading memory.

Parallelizable RNNs. Parallelization is a *sine qua non* condition nowadays, and is one the features of Transformers that put them in the front of the scene. This motivates also the research in SSMs. Since the creation of SSMs [28, 4], numerous papers have added their contributions and improvements, which finally led to Mamba [5], the first SSM architecture that was used in large language models. Other works tried to make RNN parallelizable, but most of them end up removing the nonlinearities [11, 29, 12]. There exist works that developed parallelizable RNNs, but not without concession: They had for example to remove the time-dependency in the RNN update [30] or to limit the depth of the network [31]. Recent and promising

approaches reformulate the RNNs equations as a system and use iterative methods to solve it [14, 15, 16]. In this work, we show that multistable RNNs can be parallelizable over the sequence length if we consider the case of convergent RNNs, i.e. RNNs that reach full convergence between two timesteps.

Surrogate gradients. The usage of non-differentiable functions in neural networks does not happen often, therefore surrogate gradients are not that useful in classic deep learning. However, in some fields with restrictions, it allows to benefit from all the advantages of back-propagation while using special functions. For instance, in the topic of *spiking neural networks*, where outputs must be 0's and 1's, surrogate gradients have allowed the networks to be trained like any classical networks [18, 32, 33, 17]. However, the impact of using an approximation during backward pass is difficult to measure, and therefore makes the usage of surrogate gradients delicate. Also referred to as *pseudo-gradient*, this idea is not really new: Bengio et al. [19] introduces the *straight-through estimator* uses a constant pseudo-gradient, i.e. the derivative of a linear activation, while Zeng et al. [34] and Goodman and Zeng [35] use the derivative of a sigmoid to train MLPs and RNNs with threshold units.

Steady-states and equilibria. While BMRU does not really rely on the computation of steady-states, the notion of MRU is built around it and the concept of implicit function. This reminds of *Deep Equilibrium Models* (DEQs) introduced by Bai et al. [36] where layers are formulated as an implicit function and outputs are the steady-states of this implicit function. They use a solver to estimate this steady-state starting from some initial guess. However, this initial guess does not depend on any past information, and the implicit function are typically monostable, therefore these layers do not implement any memory. A MRU can be seen as a variant of DEQs where the implicit function is multistable and the initial guess is chosen to be the previous steady-state.

Future works. In the experiments section, we highlighted three characteristics of BMRU: its property to extend its memory to much longer durations, the gain of performance that can be obtained when combining it with a SSM, and its ability to learn long-term dependencies in difficult benchmarks with shallow networks. All of these deserve to be explored. For instance, we know that SSMs are better in deep networks, which is not the case of BMRU, therefore making mixed models where recurrent blocks are made of more SSM layers than BMRU layers could lead to interesting results. Also, this degradation of performance when more BMRU layers are added could be explored, as improvements in the cell equations and initialization could be made to improve learning in deeper models. Furthermore, BMRU equations have three interesting properties: the quantization of the state, the shape of the update decision, which is a comparison between $\hat{h}_t$ and β_t, and finally the possibility for the update gate z_t to be null. The first one is necessary to have discrete stable states, the second one to approximate the hysteresis bifurcation and the last one to have persistent memory (as $z_t = 0$ implies $h_t = h_{t-1}$). However, these can be used independently of the others. The impact of each definitively deserves to be explored, especially the third one which allows for persistent memory, as we are not aware of another RNN architecture that allows for this property. In addition to that, BMRU update rule is restrictive: either we keep the past state, or we totally forget it. However, nothing prevents from combining the past state and the input, as long as the dependency with respect to the past state stays linear. Also, we note that for introducing the MRU, we put forward a concept of RNNs with internal clocks, the MRU being a specific implementation of such cells when the internal clock iterates an infinite number of times. In this respect, we believe it may also be potentially interesting to further exploit this concept when the clock iterates a finite number of times. Furthermore, the usage of surrogate gradients is practically inexistent in classic deep learning, as all computations are differentiable. While it seems that using surrogate gradients may impede the performance of neural networks by introducing some mismatches between forward and backward passes, we wonder if using non-differentiable functions in classical neural networks may increase the performance and robustness of these networks. Indeed, step functions like the Heaviside one are more resilient to small changes in their output, but they probably make the training more difficult. Finally, BMRU was only tested on classification and regression tasks, but one of its advantages shared with SSMs is its efficient sequence generation capacity. It would be interesting to try it on generation tasks like text generation benchmarks for instance.

7 Conclusion

In this paper, we introduce the concept of *memory recurrent units* (MRU): a new class of RNNs that do not exhibit any transient dynamics but that creates persistent memory through multistability. We also present a concrete implementation of a MRU, the *bistable memory recurrent unit* (BMRU), derived from the hysteresis bifurcation. The equations of BMRU are closed to the usual gated RNN equations, and are compatible with the parallel scan algorithm.

We observe that BMRU can achieve good results in different benchmarks, all requiring learning long-term dependencies. Moreover, combining BMRU with a SSM leads to a parallelizable recurrent model that has both linear transient dynamics and a multistable behavior, which allows to efficiently encode temporal information for very long durations. Indeed, the linear dynamics can encode complex information but this memory will unavoidably fade over time, while multistability encode more qualitative information in a never-fading memory.

Finally, BMRU has shown to be efficient with shallow networks, while SSMs typically requires more layers to learn long-term dependencies.

To conclude, MRUs are a new concept that deserve to be explored and especially since we showed it could lead to a well working implementation, the BMRU, that showed interesting properties and performance. This paves the way for new experiments and improvements for future designs.

Acknowledgments

This work has been the subject of a patent application (Number: EP26151077). Florent De Geeter gratefully acknowledges the financial support of the Walloon Region for Grant No. 2010235 – ARIAC by DW4AI. Gaspard Lambrechts is a postdoctoral researcher of the *Fund for Scientific Research* (FNRS) from the *Wallonia-Brussels Federation* in Belgium. The present research benefited from computational resources made available on Lucia, the Tier-1 supercomputer of the Walloon Region, infrastructure funded by the Walloon Region under the grant agreement n°1910247. This work was supported by the Belgian Government through the Federal Public Service Policy and Support.

References

[1] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[2] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1724–1734.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[4] A. Gu, K. Goel, and C. Re, "Efficiently Modeling Long Sequences with Structured State Spaces," *ArXiv*, Oct. 2021.

[5] A. Gu and T. Dao, "Mamba: Linear-Time Sequence Modeling with Selective State Spaces," in *First Conference on Language Modeling*, Aug. 2024.

[6] S. Boyd and L. Chua, "Fading memory and the problem of approximating nonlinear operators with Volterra series," *IEEE Transactions on Circuits and Systems*, vol. 32, no. 11, pp. 1150–1161, Nov. 1985.

[7] W. Merrill, J. Petty, and A. Sabharwal, "The illusion of state in state-space models," in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML'24, vol. 235. Vienna, Austria: JMLR.org, Jul. 2024, pp. 35 492–35 506.

[8] N. Vecoven, D. Ernst, and G. Drion, "A bio-inspired bistable recurrent cell allows for long-lasting memory," *PLOS ONE*, vol. 16, no. 6, p. e0252676, Jun. 2021.

[9] G. Lambrechts, F. De Geeter, N. Vecoven, D. Ernst, and G. Drion, "Warming up recurrent neural networks to maximise reachable multistability greatly improves learning," *Neural Networks*, vol. 166, pp. 645–669, Sep. 2023.

[10] J. T. H. Smith, A. Warrington, and S. Linderman, "Simplified State Space Layers for Sequence Modeling," in *The Eleventh International Conference on Learning Representations*, Sep. 2022.

[11] E. Martin and C. Cundy, "Parallelizing Linear Recurrent Neural Nets Over Sequence Length," in *International Conference on Learning Representations*, Feb. 2018.

[12] L. Feng, F. Tung, M. O. Ahmed, Y. Bengio, and H. Hajimirsadeghi, "Were RNNs All We Needed?" 2024.

[13] Z. Qin, S. Yang, and Y. Zhong, "Hierarchically Gated Recurrent Neural Network for Sequence Modeling," *Advances in Neural Information Processing Systems*, vol. 36, pp. 33 202–33 221, Dec. 2023.

[14] Y. H. Lim, Q. Zhu, J. Selfridge, and M. F. Kasim, "Parallelizing non-linear sequential models over the sequence length," in *The Twelfth International Conference on Learning Representations*, Oct. 2023.

[15] X. Gonzalez, A. Warrington, J. T. Smith, and S. W. Linderman, "Towards Scalable and Stable Parallelization of Nonlinear RNNs," *Advances in Neural Information Processing Systems*, vol. 37, pp. 5817–5849, Dec. 2024.

[16] F. Danieli, P. Rodriguez, M. Sarabia, X. Suau, and L. Zappella, "ParaRNN: Unlocking Parallel Training of Nonlinear RNNs for Large Language Models," 2025.

[17] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, Nov. 2019.

[18] J. K. Eshraghian, M. Ward, E. O. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training Spiking Neural Networks Using Lessons From Deep Learning," *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1016–1054, Sep. 2023.

[19] Y. Bengio, N. Léonard, and A. Courville, "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation," Aug. 2013.

[20] A. Orvieto, S. L. Smith, A. Gu, A. Fernando, C. Gulcehre, R. Pascanu, and S. De, "Resurrecting Recurrent Neural Networks for Long Sequences," in *Proceedings of the 40th International Conference on Machine Learning*. PMLR, Jul. 2023, pp. 26 670–26 698.

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[22] Q. V. Le, N. Jaitly, and G. E. Hinton, "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units," *ArXiv*, Apr. 2015.

[23] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler, "Long Range Arena: A Benchmark for Efficient Transformers," *ArXiv*, Nov. 2020.

[24] C.-Y. Cheng, K.-H. Lin, and C.-W. Shih, "Multistability in Recurrent Neural Networks," *SIAM Journal on Applied Mathematics*, vol. 66, no. 4, pp. 1301–1320, Jan. 2006.

[25] K. Krishnamurthy, T. Can, and D. J. Schwab, "Theory of Gating in Recurrent Neural Networks," *Physical Review X*, vol. 12, no. 1, p. 011011, Jan. 2022.

[26] R. Edwards, "Analysis of continuous-time switching networks," *Physica D: Nonlinear Phenomena*, vol. 146, no. 1-4, pp. 165–199, Nov. 2000.

[27] M. Tournoy and B. Doiron, "A Step Towards Uncovering The Structure of Multistable Neural Networks," 2022.

[28] A. Gu, I. Johnson, K. Goel, K. K. Saab, T. Dao, A. Rudra, and C. Re, "Combining Recurrent, Convolutional, and Continuous-time Models with Linear State-Space Layers," *Neural Information Processing Systems*, 2021.

[29] M. Beck, K. Pöppel, M. Spanring, A. Auer, O. Prudnikova, M. Kopp, G. Klambauer, J. Brandstetter, and S. Hochreiter, "xLSTM: Extended Long Short-Term Memory," *Advances in Neural Information Processing Systems*, vol. 37, pp. 107 547–107 603, Dec. 2024.

[30] H. Jiang, F. Qin, J. Cao, Y. Peng, and Y. Shao, "Recurrent neural network from adder's perspective: Carry-lookahead RNN," *Neural Networks*, vol. 144, pp. 297–306, Dec. 2021.

[31] J. E. Zini, Y. Rizk, and M. Awad, "An Optimized Parallel Implementation of Non-Iteratively Trained Recurrent Neural Networks," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 11, no. 1, pp. 33–50, Jan. 2021.

[32] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training Deep Spiking Neural Networks Using Backpropagation," *Frontiers in Neuroscience*, vol. 10, 2016.

[33] N. P. Nieves and D. F. M. Goodman, "Sparse Spiking Gradient Descent," in *Neural Information Processing Systems*, May 2021.

[34] Z. Zeng, R. M. Goodman, and P. Smyth, "Learning Finite State Machines With Self-Clustering Recurrent Networks," *Neural Computation*, vol. 5, no. 6, pp. 976–990, Nov. 1993.

[35] R. Goodman and Z. Zeng, "A learning algorithm for multi-layer perceptrons with hard-limiting threshold units," in *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, Sep. 1994, pp. 219–228.

[36] S. Bai, J. Z. Kolter, and V. Koltun, "Deep Equilibrium Models," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.

[37] G. E. Blelloch, "Prefix sums and their applications," School of Computer Science, Carnegie Mellon University Pittsburgh, PA, USA, Tech. Rep., 1990.

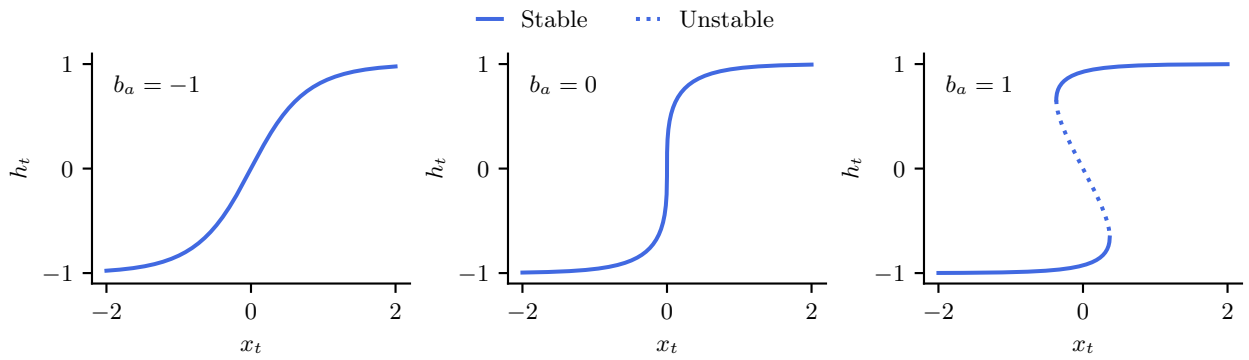


Figure 12: **Hysteresis bifurcation in the bistable recurrent cell.** This figure shows the solutions to the implicit function defined by equation (15) as well as their stability for three values of b_a.

A Hysteresis bifurcation in the bistable recurrent cell

The bistable recurrent cell [8] is described the set of equations

$$c_t = \sigma(U_c x_t + w_c \odot h_{t-1} + b_c), \tag{14a}$$
$$a_t = 1 + \tanh\left(U_a x_t + w_a \odot h_{t-1} + b_a\right), \tag{14b}$$
$$h_t = c_t \odot h_{t-1} + (1 - c_t) \odot \tanh\left(U_h x_t + a_t h_{t-1} + b_h\right), \tag{14c}$$

where c_t is the update gate, a_t the feedback gate, and $U_c, w_c, b_c, U_a, w_a, b_a, U_h, b_h$ are learnable parameters. At steady-state, we have $h_t = h_{t-1}$, and equation (14c) writes

$$(1 - c_t) \odot h_t = (1 - c_t) \odot \tanh\left(U_h x_t + a_t h_t + b_h\right),$$

which, by replacing a_t by equation (14b) at steady-state and dividing each term by $(1 - c_t)$ (c_t being the output of a sigmoid function, its value is strictly smaller than 1), leads to

$$\tanh\left(U_h x_t + (1 + \tanh\left(U_a x_t + w_a \odot h_t + b_a\right))h_t + b_h\right) - h_t = 0. \tag{15}$$

This steady-state function is a static, implicit function that relates the values of the input x_t to the values of the state h_t at convergence. We can show that this function corresponds to a hysteresis singularity with x_t as the bifurcation parameter and b_a as the unfolding parameter. Figure 12 illustrates the unfolding of the hysteresis bifurcation for $b_a = -1$ (monostable), $b_a = 0$ (singular) and $b_a = 1$ (bistable) (other parameter values are $U_h = 1, U_a = w_a = b_h = 0$). Modifications of the parameters $U_h \neq 0, U_a, w_a, b_h$ lead to translations and deformations of the bifurcation plot without alteration of the existence of a hysteresis singularity.

B Compatibility of BMRU with the parallel scan algorithm

The *scan* operation, also called the *prefix sum* operation, computes from a binary operator $\oplus$ and an array of n elements $[a_0, a_1, \ldots, a_{n-1}]$ the array

$$[a_0, a_0 \oplus a_1, \ldots, a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1}].$$

A naive implementation of the scan consists in a simple loop over the input array that accumulates the results and stores them in the output array. This implementation has a time complexity of $O(n)$. However, it is possible to perform the scan with a better time complexity when the operator $\oplus$ is associative and when multiple processors are available.

The parallel scan algorithm introduced by Blelloch [37] is an algorithm that performs the *scan* operation on a array of n elements with a time complexity of $O(\frac{n}{p} + \log(p))$, where p is the number of processors. Assuming

this algorithm is run on a GPU, we can consider $p = n$, which therefore gives a time complexity of $O(\log(n))$. Notably, this algorithm is used with the SSMs during their training [10].

This section proves that BMRU equations are compatible with the parallel scan algorithm.

Theorem 1: *The parallel scan can be used to perform the scan on an array with a binary operator $\oplus$ if and only if $\oplus$ is associative.*

Proof. The proof is given in [37]. □

Furthermore, Blelloch [37] showed that there exists a binary associative operator that computes the ordered set of states from a first-order linear recurrence.

Theorem 2: *Assume a first-order linear recurrence of the following form:*

$$h_t = \begin{cases} b_0 & \text{if } t = 0, \\ a_t \odot h_{t-1} + b_t & \text{if } 0 < t < T, \end{cases}$$

where a_t and b_t $\forall t \in [0, T]$ are independent of $h_{t'}$ $\forall t' \in [0, T]$. Consider the set of pairs $c_t = [a_t, b_t]$ and the binary operator $\circledast$ defined as follows:

$$c_t \circledast c_{t'} \equiv [c_{t,a} \odot c_{t',a}, c_{t',a} \odot c_{t,b} + c_{t',a}]$$

where $c_{t,a}$ and $c_{t,b}$ are the first and second elements of c_t. Then,

1. *The operator $\circledast$ is associative,*
2. *Performing the scan with the operator $\circledast$ on the array $[c_0, \ldots, c_T]$ creates the array $[s_0, \ldots, s_T]$ where $s_t = [y_t, h_t]$ and y_t is defined as:*

$$y_t = \begin{cases} a_0 & \text{if } t = 0, \\ a_t \odot y_{t-1} & \text{if } 0 < t < T. \end{cases}$$

It results that the parallel scan can be used to solve this first-order linear recurrence as $\circledast$ is associative (point 1.), and the solutions h_t will be the second values of the generated pairs s_t (point 2.).

Proof. The proof is given in [37], section 4.1. □

To prove that BMRU is compatible with the parallel scan, we can therefore show that equations (11a) to (11d) can be rewritten as a first-order linear recurrence.

Theorem 3: *Assume a sequence of inputs $[x_1, \ldots, x_T]$ and fixed parameters $W_x, W_\beta, b_x, b_\beta$ and α. Equations (11a) to (11d) describe a first-order linear recurrence, and are therefore compatible with the parallel scan.*

Proof. First, let us separate the equations that are independent of the previous state h_{t-1} from the ones that are not. Equations (11a) to (11c) do not depend on h_{t-1}. Therefore $\hat{h}_t$, β_t and z_t can be computed in parallel for all timesteps.

The last equation, equation (11d), is the only one that has to be analyzed. As a reminder, here it is:

$$h_t = z_t \odot S(\hat{h}_t) \odot \alpha + (1 - z_t) \odot h_{t-1}.$$

The left term is independent of h_{t-1}, hence let us define the set of b_t's as follows:

$$b_t \equiv z_t \odot S(\hat{h}_t) \odot \alpha.$$

Also, the factor that multiplies h_{t-1} is independent of it. Let us define the set of a_t's as follows:

$$a_t \equiv 1 - z_t.$$

Equation (11d) can therefore be rewritten using a_t and b_t:

$$a_t = 1 - z_t, \tag{16a}$$

$$b_t = z_t \odot S(\hat{h}_t) \odot \alpha, \tag{16b}$$

$$h_t = \begin{cases} h_0 & \text{if } t = 0, \\ a_t \odot h_{t-1} + b_t & \text{if } 0 < t < T. \end{cases} \tag{16c}$$

Once again, equations (16a) and (16b) are independent of h_{t-1} and can be computed in parallel for all timesteps. Finally, equation (16c) is a first-order linear recurrence and by using theorem 2, parallel scan can be used to solve it. □

C Training parameters

This section gives all the training parameters used in the experiments of section 5: Tables 1 to 3 respectively give the parameters of sections 5.2 to 5.4.

Parameter	**Value(s)**
Number of samples in dataset	60000
Sequence length	100 - 300
Train / valid ratio	90% / 10%
Epochs	100
Learning rate	Cosine annealing: $10^{-4} \to 10^{-3}$ during 10 first epochs, then $10^{-3} \to 10^{-5}$
Weight decay	0.0001 for BMRU parameters 0.05 for other parameters
Number of recurrent blocks	2
Model dim	256
State dim	256
Positional encoding	No
Activation between blocks	GLU
Bidirectional	No
Number of fully connected layers (after last block)	2
Pooling for prediction	Last timestep
(BMRU) α_{surr}	1
(LRU) r_{min}, r_{max} and θ_{max}	0.0, 0.99 and 2π

Table 1: Training parameters used in section 5.2.

Parameter	**Value(s)**
Number of black pixels	0 - 1216
Pixel normalization	$p/255 - 0.5$, with $p \in [0, 255]$
Train / valid ratio	90% / 10%
Epochs	100
Learning rate	Cosine annealing: $10^{-4} \rightarrow 10^{-3}$ during 10 first epochs, then $10^{-3} \rightarrow 10^{-5}$
Weight decay	0.0001 for BMRU parameters 0.05 for other parameters
Number of recurrent blocks	2 - 3
Model dim	256
State dim	256
Positional encoding	16 dim, only for BMRU
Activation between blocks	GLU
Bidirectional	No
Number of fully connected layers (after last block)	2
Pooling for prediction	Last timestep
(BMRU) α_{surr}	1
(LRU) r_{min}, r_{max} and θ_{max}	0.0, 0.99 and 2π

Table 2: Training parameters used in section 5.3.

Parameter	**Value(s)**
Pixel normalization	$(p - \mu)/\sigma$, with $p \in [0, 255]$, $\mu = 10.94$ and $\sigma = 38.51$
Train / valid ratio	90% / 10%
Epochs	100
Learning rate	Cosine annealing: $10^{-4} \rightarrow 10^{-3}$ during 10 first epochs, then $10^{-3} \rightarrow 10^{-5}$
Weight decay	0.0001 for BMRU parameters 0.05 for other parameters
Number of recurrent blocks	1 - 2 - 3
Model dim	256
State dim	128 - 256 - 512 - 1024
Positional encoding	16 dim
Activation between blocks	GLU
Bidirectional	Yes
Number of fully connected layers (after last block)	2
Pooling for prediction	Mean of all timesteps
(BMRU) α_{surr}	1

Table 3: Training parameters used in section 5.4.