Accelerating Hopfield Network Dynamics: Beyond Synchronous Updates and Forward Euler

Cédric Goemaere, Johannes Deleu, Thomas Demeester FIRST.LAST@UGENT.BE
IDLab, Department of Information Technology at Ghent University – imec, Ghent, Belgium

Editors: Cecília Coelho, Bernd Zimmering, M. Fernanda P. Costa, Luís L. Ferrás, Oliver Niggemann

Abstract

The Hopfield network serves as a fundamental energy-based model in machine learning, capturing memory retrieval dynamics through an ordinary differential equation (ODE). The model's output, the equilibrium point of the ODE, is traditionally computed via synchronous updates using the forward Euler method. This paper aims to overcome some of the disadvantages of this approach. We propose a conceptual shift, viewing Hopfield networks as instances of Deep Equilibrium Models (DEQs). The DEQ framework not only allows for the use of specialized solvers, but also leads to new insights on an empirical inference technique that we will refer to as 'even-odd splitting'. Our theoretical analysis of the method uncovers a parallelizable asynchronous update scheme, which should converge roughly twice as fast as the conventional synchronous updates. Empirical evaluations validate these findings, showcasing the advantages of both the DEQ framework and even-odd splitting in digitally simulating energy minimization in Hopfield networks. The code is available at https://github.com/cgoemaere/hopdeq.

Keywords: Even-odd splitting, Hopfield network, Deep Equilibrium Model

1. Introduction

In 1982, the Hopfield network was suggested as a model for associative memory retrieval (Hopfield, 1982). It restores corrupted memories by solving an ordinary differential equation (ODE) representing the gradient field of a learnable energy function, which holds the true memories at its local minima. In recent years, there has been a renewed interest in Hopfield networks, which has lead to a series of architectural improvements over the original formulation (Krotov and Hopfield, 2016; Demircigil et al., 2017; Krotov, 2021; Krotov and Hopfield, 2021; Ramsauer et al., 2021). In this paper, we consider two formulations of the Hopfield network: the continuous Hopfield network (CHN) of Bengio and Fischer (2015), and Hierarchical Associative Memory (HAM; Krotov, 2021), which extends the framework of classical continuous Hopfield networks (Hopfield, 1984) to arbitrary network architectures.

Both during training and inference, Hopfield networks require an internal energy minimization. Physical compute platforms (i.e., neuromorphic hardware) could solve this optimization problem near-instantaneously and at an extremely low energetic cost (Yi et al., 2023), but unfortunately, such technology is not yet commercially available. In anticipation of these devices, research on Hopfield networks has turned to the use of traditional digital accelerators, such as GPUs, where solving the ODE is computationally intensive, thereby hindering progress in the field. Accelerating the digital energy minimization simulations is

currently an underexplored research direction. However, we consider it an essential step in stimulating future research on Hopfield networks in general, especially at larger scales than currently investigated.

Deviating from convention, we argue that compute-intensive ODE-based dynamics may be unnecessary for modelling the behavior of a Hopfield network. Instead, we propose a conceptual shift, casting Hopfield networks to the framework of Deep Equilibrium Models (DEQs; Bai et al., 2019), which focuses on state dynamics rather than energy. Through this lens, we are able to theoretically motivate an intuitively appealing idea from Bengio et al. (2016) to help accelerate Hopfield networks and uncover the conditions required for its successful application in practice.

Our contributions:

1. We propose a new perspective on Hopfield networks, treating them as DEQs instead of ODEs. This conceptual shift simplifies theoretical analysis and enables the use of specialized solvers that may accelerate convergence.
2. Our analysis reveals that, under specific conditions, CHNs can be interpreted as HAMs, challenging the traditional categorization based solely on energy functions.
3. Revisiting an idea from Bengio et al. (2016), we uncover its nature as a parallelizable asynchronous update scheme that converges twice as fast, as empirically validated on the MNIST dataset across Hopfield networks of varying sizes.

This work expands the scope of our NeurIPS workshop paper (Goemaere et al., 2023) to include both HAMs and CHNs, and offers a much more substantial theoretical analysis and empirical validation, without assuming prior knowledge on Hopfield networks.

2. Preliminaries

This section briefly introduces the key concepts underlying the remainder of the paper. We describe the two types of Hopfield networks considered (CHN and HAM) from the perspective of the current literature. Furthermore, we provide a brief introduction to the DEQ framework and revisit the original idea of Bengio et al. (2016). Please note that we introduce recurring symbols in Table 1.

Continuous Hopfield network (CHN) While originally proposed as a model for associative memory retrieval (Hopfield, 1984), Bengio and Fischer (2015) consider the CHN an energy-based model that iteratively updates its hidden neurons to explain signals coming from the sensory neurons, similar to how the brain works. Based on the Boltzmann machine energy function (Ackley et al., 1985), they propose the energy function

$$E(s) = \frac{1}{2}jjsjj^2 \quad \frac{1}{2}\ (s)^T W\ (s) \quad b^T\ (s): \tag{1}$$

The zero-diagonal weight matrix W ($W_{ii} = 0$) is often constructed as a symmetric matrix, since any anti-symmetric component is cancelled out in Eq. (1). By convention, the first d dimensions of the state s (denoted s_0) constitute the static input x.

In follow-up work, mostly layered instantiations of the CHN have been considered, without intralayer connections (Bengio et al., 2016; Scellier and Bengio, 2017; O'Connor

Symbol	Description
$s \in \mathbb{R}^N$	State vector
$E : \mathbb{R}^N \to \mathbb{R}$	Global energy function
$\sigma : \mathbb{R}^N \to \mathbb{R}^N$	Activation function ‡ (typically non-linear)
$L : \mathbb{R}^N \to \mathbb{R}$	Lagrangian function such that $\frac{\partial L}{\partial s} =$
$W \in \mathbb{R}^{N \times N}$	State weight matrix
$b \in \mathbb{R}^N$	State bias vector
$x \in \mathbb{R}^d$	Input vector
$U \in \mathbb{R}^{N \times d}$	Input weight matrix
	Hadamard product
$\langle s \rangle$	Vector s at equilibrium
$\langle s \rangle^n$	Vector s at the n-th iteration of its DEQ

‡: We use a scalar function applied element-wise to the state vector as $\sigma(s)$, corresponding to an additive Lagrangian L (Krotov, 2021).

Table 1: List of symbols

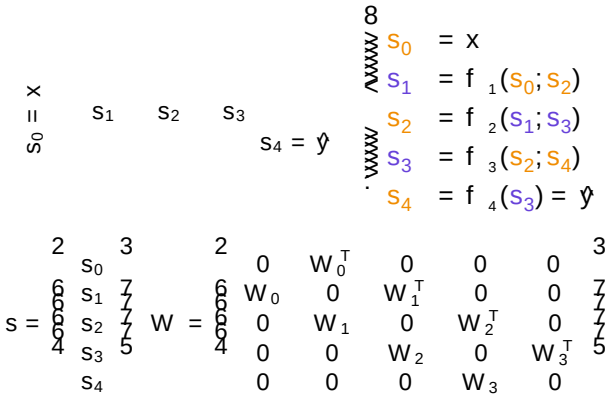

Figure 1: Diagram of a 5-layer Hopfield network (upper left), with an abstract DEQ formulation of the interlayer dynamics (upper right). State s and corresponding weight matrix W (below). Partitioning the layers into even and odd reveals a bipartite structure. Best viewed in color.

et al., 2019; Gammell et al., 2021; Laborieux and Zenke, 2023; Scellier et al., 2023), leading to zero diagonal blocks in W, as illustrated for a 5-layer architecture in Fig. 1.

The output of a CHN is s, which resides at a minimum of E given x. The energy E is guaranteed to decrease over time (Scellier and Bengio, 2017) using the state update rule

$$\frac{ds}{dt} = -\frac{\partial E}{\partial s} = -s + \sigma'(s) \quad (W \sigma(s) + b): \tag{2}$$

In the literature on Hopfield networks, the equilibrium state s is typically obtained by numerical integration of Eq. (2) using the forward Euler method (Bengio and Fischer, 2015; Bengio et al., 2016; Scellier and Bengio, 2017; Gammell et al., 2021). By contrast, in the field of Neural ODEs (Chen et al., 2018), it is customary to use more advanced ODE solvers, and these techniques have been suggested for Hopfield networks as well (Krotov, 2021).

Remark 1 It is common practice to update all states in parallel, known as `synchronous updates'. While fast, this method lacks formal convergence guarantees (Koiran, 1994; Wang, 1998). For guaranteed convergence, one must turn to `asynchronous updates', where the states are sequentially updated, but this can be very slow when applied naively.

Hierarchical Associative Memory (HAM) A HAM (Krotov, 2021) is the multilayer extension of the classical CHN (Hopfield, 1984). The main difference with the CHN from Bengio and Fischer lies in its energy function, which can be defined as

$$E(s) = s^T \sigma(s) - L(s) - \frac{1}{2} \sigma(s)^T W \sigma(s) - b^T \sigma(s); \tag{3}$$

where the Lagrangian function L is defined to be the antiderivative of σ (i.e., $\frac{\partial L}{\partial s} = \sigma$).

By convention, the dynamics of a HAM require the activations $\sigma(s)$ to be at equilibrium, rather than the states s. As a result, its state update rule can be simplified to

$$\frac{ds}{dt} = \frac{\partial E}{\partial \sigma(s)} = -s + W\sigma(s) + b. \tag{4}$$

Notice that the only difference between Eqs. (2) and (4) is the missing $\sigma'(s)$-term. All other aspects remain the same, including the implicit input dependence through s_0 and the structure of W for layered instantiations of the HAM.

Deep Equilibrium Model (DEQ) A DEQ (Bai et al., 2019) is a recurrent neural network that operates on a static input x. It returns an equilibrium point s^*, that is defined implicitly through the fixed point equation $s^* = f(s^*; x)$. The function f may be any arbitrary computation block, from a simple MLP $(Ws + b + Ux)$ to an elaborate, deep architecture (Bai et al., 2019, 2020).

DEQs can be seen as infinite, implicit or adaptive depth models, because the number of iterations – and hence the depth of the unrolled computational graph – may be scaled arbitrarily to match the difficulty of the task at hand (Anil et al., 2022).

Going beyond simple fixed point iteration, DEQs make use of specialized fixed point solvers, such as Anderson acceleration (Anderson, 1965; Walker and Ni, 2011) and Broyden's method (Broyden, 1965). Additionally, their fixed point structure overcomes the need for backpropagation-through-time by offering memory-efficient gradient estimation methods (Bai et al., 2019), based on the implicit function theorem (Krantz and Parks, 2002).

DEQs are part of the broad family of implicit models (Kolter et al., 2020), that includes Neural Differential Equations (Chen et al., 2018; Kidger, 2021) and differentiable optimization (Amos, 2019). However, distinguishing between these models can be ambiguous at times, as they often exhibit interchangeable functionality, allowing, for example, Neural ODEs to be represented as DEQs, and vice versa (Kidger, 2021; Pal et al., 2022).

Even-odd splitting Inspired by block Gibbs sampling in Deep Boltzmann Machines (Salakhutdinov and Hinton, 2009), Bengio et al. (2016) describe a two-step iterative method to accelerate the energy minimization process of multilayer CHNs. Throughout this paper, we will refer to this technique as `even-odd splitting'.

The intuition is to leverage the problem's bipartite structure (see Fig. 1) by splitting it into two smaller, more tractable sub-problems. In the first step, all odd layers are brought to a local energy minimum, conditioned on the fixed values of the even layers, and in the second step, the roles are reversed. Bengio et al. argue that iteratively applying these two steps should converge faster than synchronously updating all layers, under the condition that there are no intralayer connections and only connections between successive layers.[1]

Other than that, the paper does not go into more detail. Even-odd splitting is not its main contribution, and accordingly, no empirical results on the matter are presented. Despite its intuitive appeal, the method has not gained much traction in the community, possibly because of its ineffectiveness in CHNs in practice (see Section 4). Independently, Scellier et al. (2023) introduced a similar technique that shows good performance, but also lacks a theoretical foundation.

1. Notice how the architecture in Fig. 1 satisfies both conditions.

3. Even-odd splitting from a Deep Equilibrium perspective

Intuitively, even-odd splitting seems like a promising way to accelerate Hopfield networks, but gaining more theoretical insights is difficult, due to the contemporary energy-based formulation of Hopfield networks. To address this, we propose a conceptual shift towards the DEQ framework, which enables a clearer, step-by-step analysis of the procedure, from the splitting of the states to the local energy optimization. Our findings reveal that even-odd splitting is not to blame for its ineffectiveness in CHNs. Rather, it is the CHN itself that is inherently inefficient. Contrary to the traditional energy-based distinction, we demonstrate that, under specific conditions, CHNs can be interpreted as HAMs, albeit with a more complicated non-linearity. Finally, we uncover an elegant mathematical formulation of even-odd splitting in HAMs and discuss its advantages compared to the commonly used synchronous updates.

While the transformation to the DEQ framework primarily serves to streamline our theoretical analysis below, it is worth noting that this reinterpretation also offers practical computational benefits that we exploit in our experiments in Section 4. We provide a brief overview of these advantages in Appendix A.

3.1. From Hopfield network to DEQ

Strictly speaking, Hopfield networks are by definition energy-based models, and yet, their dynamics are entirely defined by the ODEs in Eqs. (2) and (4). In fact, contrary to Neural ODEs (Chen et al., 2018), only the equilibrium point matters in a Hopfield network, not the trajectory to get there. In this particular case, the exact time dynamics are irrelevant, and the problem can be solved much faster by casting the ODE as a DEQ (Pal et al., 2022). As this has not been previously described for Hopfield networks, we explicitly derive the formulation in Appendix B, resulting in:

$$\text{CHN as DEQ:} \quad \tilde{s} = \rho'(\tilde{s}) \odot (\tilde{W}\rho(\tilde{s}) + \tilde{b} + U\rho(x)); \tag{5}$$

$$\text{HAM as DEQ:} \quad \tilde{s} = \sigma(\tilde{W}\rho(\tilde{s}) + \tilde{b} + U\rho(x)). \tag{6}$$

Here, the tilde on $\tilde{s}$, $\tilde{W}$, and $\tilde{b}$ indicates a change in dimensionality resulting from the separation of the input layer $s_0 = x$. For readability, however, we chose to leave it out in the following sections.

Example 1 In a 5-layer Hopfield network like Fig. 1, we can easily derive that

$$\tilde{W} = \begin{bmatrix} 0 & W_1^T & 0 & 0 \\ W_1 & 0 & W_2^T & 0 \\ 0 & W_2 & 0 & W_3^T \\ 0 & 0 & W_3 & 0 \end{bmatrix}; \quad U = \begin{bmatrix} W_0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

3.2. Splitting the states into even & odd

The first step towards understanding even-odd splitting is to introduce a distinction between the even and odd layers, as formalized in Theorem 2 and proven in Appendix C.

Theorem 2 Even-odd splitting rearranges the layered structure of s using a permutation matrix P, such that $s = [s_1; s_2; s_3; \dots]$ is converted into $Ps = [s_{even}; s_{odd}]$, with $s_{even} = [s_2; s_4; \dots]$ and $s_{odd} = [s_1; s_3; \dots]$. Under this change of variables, the fixed point iteration procedures for the CHN and HAM, according to Eqs.(5) and (6), are transformed into

$$\text{CHN iteration:} \quad \begin{cases} s_{even}^{n+1} = \sigma'(s_{even}^n) \odot W_P^T (s_{odd}^n) + b_{even} \\ s_{odd}^{n+1} = \sigma'(s_{odd}^n) \odot W_P (s_{even}^n) + b_{odd} + U_{odd}(x) \end{cases}; \tag{7}$$

$$\text{HAM iteration:} \quad \begin{cases} s_{even}^{n+1} = W_P^T (s_{odd}^n) + b_{even} \\ s_{odd}^{n+1} = W_P (s_{even}^n) + b_{odd} + U_{odd}(x) \end{cases}: \tag{8}$$

Example 2 In a 5-layer Hopfield network like Fig. 1, we have

$$W_P = \begin{matrix} W_1^T & 0 \\ W_2 & W_3^T \end{matrix}; \quad U_{odd} = \begin{matrix} W_0 \\ 0 \end{matrix};$$

with the matrix at block position $(i; j)$ in W_P representing the influence of s_{2j+2} on s_{2i+1}.

3.3. Finding the local energy optimum

The next step is to bring each layer to its local energy minimum, given its neighbors. When viewed in parallel, the task amounts to computing s_{even} given s_{odd}, and vice versa. In this regard, HAMs are exceptionally well suited for even-odd splitting. Given a fixed value of s_{odd} in Eq. (8), the equilibrium value s_{even} is retrieved after a single iteration, and vice versa. By contrast, in CHNs (i.e., Eq. (7)), finding this local equilibrium point still requires a computationally expensive fixed point iteration, thereby nullifying any practical benefits (see Section 4). Here, the DEQ framework proves particularly effective, enabling us to derive an analytic solution, as detailed and proven in Appendix D, resulting in Theorem 3.

Theorem 3 Under relatively mild conditions for σ and up to an input preprocessing step, a well-behaved CHN can be transformed into a functionally equivalent HAM with effective non-linearity $\sigma_\&^{-1} := \sigma \,\&\, \sigma^{-1}$, where $\&(s) := s \odot \sigma'(s)$, with $\odot$ representing the Hadamard division.

Remark 4 In practice, one typically picks a non-linearity σ that can be quickly computed, such as a sigmoid. Nevertheless, Theorem 3 states that a well-behaved CHN internally models a function inversion $\&^{-1}$, which it performs iteratively at inference. Hence, a CHN can be thought of as a HAM with a non-linearity that is more involved to compute.

3.4. DEQ formulation of even-odd splitting in HAMs

Consider a Hopfield network with a layered architecture, such as the one in Fig. 1. For an odd number $2k+1$ of layers, the output layer s_{2k} belongs to s_{even}, such that s_{odd} serves only as an auxiliary variable, which need not be explicitly modelled, and vice versa for an even number of layers. In HAMs, this entails a substitution of the expression for s_{odd}^{n+1} in Eq. (8) into the one for s_{even}^{n+2}, resulting in

$$s_{even}^{n+2} = W_P^T \sigma(W_P (s_{even}^n) + b_{odd} + U_{odd}(x)) + b_{even}. \tag{9}$$

From a different perspective, Eq. (9) may be interpreted as a single iteration of the DEQ

$$s_{even} = W_P^T \quad W_P \quad (s_{even}) + b_{odd} + U_{odd} \quad (x) \quad + b_{even}; \tag{10}$$

corresponding to the DEQ formulation of even-odd splitting in HAMs and certain CHNs (by Theorem 3).

3.5. Advantages of even-odd splitting in Hopfield networks

In Hopfield networks, even-odd splitting comes with two notable advantages over the traditional synchronous updates: convergence is reached faster and is always guaranteed.

Faster convergence By advancing two time steps (i.e., $s^n_{even} \to s^{n+2}_{even}$ in Eq. (9)) in a single iteration, even-odd splitting should converge twice as fast as fully synchronous updates of Eq. (6), as given by Eq. (8). Crucially, one fixed point iteration of Eq. (10) still requires exactly as many computations as an iteration of Eq. (6) when not accounting for multiplications by zero (i.e., assuming an optimal block-sparse matrix multiplication). To verify this, observe that a single iteration always corresponds to all states being updated exactly once (in Eq. (9): first, s_{odd} at step $n+1$ and then s_{even} at step $n+2$).

Guaranteed convergence As mentioned in Remark 1, asynchronous update schemes have convergence guarantees for Hopfield networks, unlike synchronous updates. Interestingly, the order in which the states are asynchronously updated, traditionally chosen to be random, does not influence this property (Koiran, 1994).[2] Through a clever grouping of the states into even and odd layers, even-odd splitting essentially establishes an ordering of asynchronous state updates that achieves maximal parallelism in layered architectures.

In Appendix E, we provide some intuition into the problems that may occur when using synchronous updates and highlight how even-odd splitting naturally overcomes them.

4. Experimental results

4.1. Setup

To assess the impact of even-odd splitting and DEQ solvers in Hopfield networks, we performed an ablation study on several CHNs and HAMs of different depths. The models were trained on the MNIST dataset (LeCun, 1998; Cohen et al., 2017) for 10 epochs, with the relevant techniques active both during training and testing.

We evaluated convergence speed and performance on the test set and list the results in Table 2. Rather than relying on wall time or FLOPS, which depend on the exact implementation and hardware, we quantify convergence speed by the number of iterations required to reach convergence[3]. Recall, however, that one iteration of even-odd splitting involves finding two local equilibrium points (s_{odd} & s_{even}). In HAMs, each point is retrieved in a single step, but in CHNs, it may require several iterations (here: 10).

Further details are provided in Appendix F.

2. The order does, however, influence the convergence speed

3. In our experiments, `convergence' denotes a relative residual $\frac{\|s^{n+1} - s^n\|_2}{\|s^{n+1}\|_2} < 10^{-4}$.

	Model	#iters to conv.	Speedup	Test acc. (%)
3 layers	CHN (10 epochs)	75.5 (2.1)	0.5x	97.9 (0.1)
	CHN (3 epochs)	39.1 (3.8)	1x	97.0 (0.2)
	CHN-DEQ	20.6 (0.2)	1.9x	97.2 (0.3)
	CHN-EO	16.8 (0.5)	0.1x	97.1 (0.1)
	CHN-EO-DEQ	16.2 (1.0)	0.1x	97.1 (0.2)
3 layers	HAM	11.9 (0.4)	1x	97.9 (0.0)
	HAM-DEQ	9.9 (0.3)	1.2x	97.9 (0.1)
	HAM-EO	8.0 (0.2)	1.5x	97.9 (0.1)
	HAM-EO-DEQ	6.6 (0.2)	1.8x	97.9 (0.1)
5 layers	HAM	36.0 (1.8)	1x	97.1 (0.1)
	HAM-DEQ	33.0 (0.6)	1.1x	97.1 (0.2)
	HAM-EO	18.3 (0.5)	2.0x	97.1 (0.1)
	HAM-EO-DEQ	17.7 (0.3)	2.0x	97.1 (0.1)
7 layers	HAM	67.1 (2.9)	1x	95.6 (0.2)
	HAM-DEQ	56.0 (1.4)	1.2x	95.6 (0.1)
	HAM-EO	32.2 (0.8)	2.1x	95.5 (0.2)
	HAM-EO-DEQ	31.0 (1.0)	2.2x	95.5 (0.2)

: trained for only 4 epochs; consistently became unstable during 5^{th} epoch

: 1 iteration of even-odd splitting in CHNs comprises 2x10 local iterations

Table 2: Impact of DEQ solver (`DEQ') and even-odd splitting (`EO') on the mean number of iterations until convergence and MNIST test accuracy. Speedup refers to the reduction in state updates needed to reach convergence w.r.t. the base model (including within-iteration updates). Results aggregated across 5 runs.

4.2. Discussion

Our experiments reveal that combining the DEQ solver with even-odd splitting signi cantly accelerates convergence , with both methods e ective individually and most impactful when used together (see Table 2). This aligns with our expectations from Section 3.5, where we predicted that even-odd splitting would halve the iteration count.

In CHNs , we observed a trade-o between speed and test accuracy. The vanilla CHN gradually demands more iterations as training progresses, a common phenomenon in DEQs (Bai et al., 2021). In the other con gurations, though, it was far less prominent.

To ensure a fair comparison, we limited the vanilla CHN's training to 3 epochs. Even then, the DEQ solver still shows a substantial speedup, as it constitutes a more powerful method to solve the CHN's function inversion at inference (see Remark 4). Conversely, even-odd splitting does not o er a practical speedup. The added cost of resolving local equilibria in each iteration signi cantly outweighs the reduction in total iterations.

In HAMs , both the DEQ solver and even-odd splitting enhance convergence without trade-o s in test accuracy. The simpler model dynamics make these methods particularly e ective, delivering optimal performance when combined. In contrast to its substantial role in CHNs, the DEQ solver's more limited advantage in HAMs indicates that, here, its value primarily lies in stabilizing initial conditions, where early dynamics di er from those in the stable regime. For a visual comparison of the state dynamics in the di erent models, we refer the reader to Figs. 3 and 4 in Appendix G.

5. Limitations

Our experiments focused on the intrinsic effects of DEQ solvers and even-odd splitting on the convergence speed of Hopfield networks, which may not directly translate to reduced compute times. For small architectures simulated on modern processors, the added solver complexity or the sequential nature of even-odd splitting could offset the convergence gains. In our tests, even-odd splitting showed notable computational benefits, whereas DEQ solvers were slightly less efficient. Although these methods may offer greater advantages with larger models, our conclusions on scalability are limited given this study's proof-of-concept scope.

State-of-the-art performance was not our objective; instead, we opted for minimal hyperparameter tuning, only enough to ensure model stability. The impact of design choices (e.g., initialization, non-linearity, optimizer, number of layers) in Hopfield networks remains underexplored, leaving room for further improvement.

One key aspect is the choice of Lagrangian L, which defines the family of HAM models. In particular, we did not investigate the HAM extension of the Modern Hopfield Network (Krotov and Hopfield, 2021; Ramsauer et al., 2021), which could be an interesting direction for future research.

Lastly, while we focused on Hopfield networks, even-odd splitting may also benefit other architectures with bipartite structures. Similarly, DEQ solvers – often designed as root finders – could be useful in accelerating other energy-based models, provided that the gradient can be formulated easily.

6. Conclusion

The goal of this paper was to accelerate the digital simulation of energy minimization in Hopfield networks. To that end, we proposed a conceptual shift, away from the traditional energy-based view, toward the DEQ framework, which allows for simpler analysis and offers several computational advantages, such as specialized solvers and lower memory complexity. This perspective enabled us to derive theoretical underpinnings for even-odd splitting, an intuitive idea from Bengio et al. (2016), and uncovered a correspondence between two commonly used types of Hopfield networks, the CHN and HAM. Our analysis revealed that the CHN essentially performs a function inversion at inference, which may not be computationally optimal. The experimental results demonstrate the effectiveness of both the DEQ framework (with its specialized solvers) and even-odd splitting, especially when combined. Specifically, we observed that these techniques reduced the required amount of iterations to reach convergence, without compromising on test accuracy.

In light of these findings, we advocate for the use of the DEQ framework as a basis for both theoretical analysis and practical implementation of Hopfield networks, given its concise, equilibrium-focused notation and its computational efficiency. For researchers looking to speed up their Hopfield networks without sacrificing performance, we recommend the use of DEQ solvers and even-odd splitting, alongside traditional solutions like optimized code and high-performance hardware. Additionally, we encourage researchers to focus on HAMs instead of CHNs, as the primary difference lies in the choice of non-linearity, with HAMs being more computationally efficient. The tools and methodologies presented in this work aim to facilitate the practical scaling-up of Hopfield networks, which we hope will stimulate further research into this growing field.

Acknowledgments

We are grateful to the anonymous reviewers for their valuable feedback and suggestions, which helped improve the presentation and clarity of this paper.
This research was funded by the Research Foundation { Flanders (FWO-Vlaanderen) through CG's PhD Fellowship fundamental research with grant number 11PR824N, and the research project fundamental research G0C2723N.

References

Ackley, David H, Geo rey E Hinton, and Terrence J Sejnowski (1985). \A learning algorithm for Boltzmann machines." In: Cognitive science 9.1, pp. 147{169.

Amos, Brandon (2019). \Di erentiable Optimization-Based Modeling for Machine Learning." PhD thesis. Carnegie Mellon University.

Anderson, Donald G (1965). \Iterative procedures for nonlinear integral equations." In: Journal of the ACM (JACM) 12.4, pp. 547{560.doi : 10.1145/321296.321305 .

Anil, Cem et al. (2022). \Path Independent Equilibrium Models Can Better Exploit Test-Time Computation." In: Advances in Neural Information Processing Systems 35, pp. 7796{7809.

Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun (2019). \Deep Equilibrium Models." In: Advances in Neural Information Processing Systems Vol. 32.

Bai, Shaojie, Vladlen Koltun, and J. Zico Kolter (2020). \Multiscale deep equilibrium models." In: Advances in Neural Information Processing Systems Vol. 33, pp. 5238{5250.

Bai, Shaojie, Vladlen Koltun, and J. Zico Kolter (2021). \Stabilizing Equilibrium Models by Jacobian Regularization." In: International Conference on Machine Learning.

Bengio, Yoshua and Asja Fischer (2015). \Early Inference in Energy-Based Models Approximates Back-Propagation." In: ArXiv abs/1510.02777.

Bengio, Yoshua et al. (2016). \Feedforward Initialization for Fast Inference of Deep Generative Networks is biologically plausible." In: ArXiv abs/1606.01651.

Bernstein, Jeremy et al. (2020). \Learning compositional functions via multiplicative weight updates." In: Neural Information Processing Systems

Broyden, Charles G (1965). \A class of methods for solving nonlinear simultaneous equations." In: Mathematics of Computation 19.92, pp. 577{593.doi : 10.2307/2003941 .

Chen, Ricky TQ et al. (2018). \Neural Ordinary Di erential Equations." In: Neural Information Processing Systems Vol. 31.

Cohen, Gregory et al. (2017). \EMNIST: Extending MNIST to handwritten letters." In: 2017 international joint conference on neural networks (IJCNN). IEEE, pp. 2921{2926. doi : 10.1109/IJCNN.2017.7966217 .

Demircigil, Mete et al. (2017). \On a Model of Associative Memory with Huge Storage Capacity." In: Journal of Statistical Physics 168, pp. 288{299.doi : 10.1007/s10955-017-1806-y .

Gammell, Jimmy et al. (2021). \Layer-Skipping Connections Improve the E ectiveness of Equilibrium Propagation on Layered Networks." In: Frontiers in Computational Neuroscience 15, p. 627357.doi : 10.3389/fncom.2021.627357 .

Ghaoui, L. et al. (2019). \Implicit Deep Learning." In: SIAM Journal on Mathematics of Data Science doi : 10.1137/20M1358517.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks." In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, pp. 249–256.

Goemaere, Cédric, Johannes Deleu, and Thomas Demeester (2023). "Accelerating Hierarchical Associative Memory: A Deep Equilibrium Approach." In: Associative Memory & Hopfield Networks in 2023. url: https://openreview.net/forum?id=Vmndp6HnfR.

Grother, Patrick J. (1995). "NIST special database 19." In: Handprinted forms and characters database, National Institute of Standards and Technology 10, p. 69. doi: 10.18434/T4H01C.

Hopfield, John J. (1982). "Neural networks and physical systems with emergent collective computational abilities." In: Proceedings of the National Academy of Sciences 79.8, pp. 2554–2558. doi: 10.1073/pnas.79.8.2554.

Hopfield, John J. (1984). "Neurons with graded response have collective computational properties like those of two-state neurons." In: Proceedings of the National Academy of Sciences 81.10, pp. 3088–3092. doi: 10.1073/pnas.81.10.3088.

Almeida, Luis B. (1987). "A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment." In: Proceedings of the IEEE First International Conference on Neural Networks (San Diego, CA). Vol. II. Piscataway, NJ: IEEE, pp. 609–618. doi: 10.5555/104134.104145.

Kidger, Patrick (2021). "On Neural Differential Equations." PhD thesis. University of Oxford.

Koiran, Pascal (1994). "Dynamics of Discrete Time, Continuous State Hopfield Networks." In: Neural Computation 6.3, pp. 459–468. doi: 10.1162/neco.1994.6.3.459.

Kolter, J. Zico, David Duvenaud, and Matt Johnson (2020). url: https://implicit-layers-tutorial.org/.

Krantz, Steven George and Harold R Parks (2002). The implicit function theorem: history, theory, and applications. Springer Science & Business Media.

Krotov, Dmitry (2021). "Hierarchical Associative Memory." In: ArXiv abs/2107.06446.

Krotov, Dmitry and John J. Hopfield (2016). "Dense associative memory for pattern recognition." In: Advances in neural information processing systems. Vol. 29.

Krotov, Dmitry and John J. Hopfield (2021). "Large Associative Memory Problem in Neurobiology and Machine Learning." In: International Conference on Learning Representations.

Laborieux, Axel and F. T. Zenke (2023). "Improving equilibrium propagation without weight symmetry through Jacobian homeostasis." In: ArXiv abs/2309.02214.

Laborieux, Axel et al. (2020). "Scaling Equilibrium Propagation to Deep ConvNets by Drastically Reducing its Gradient Estimator Bias." In: Frontiers in Neuroscience. doi: 10.3389/fnins.2021.633674.

LeCun, Yann (1998). "The MNIST database of handwritten digits." In: http://yann.lecun.com/exdb/mnist/.

O'Connor, Peter, Efstratios Gavves, and Max Welling (2019). "Initialized Equilibrium Propagation for Backprop-Free Training." In: International Conference on Learning Representations.

Ota, Toshihiro and Masato Taki (2023). \iMixer: hierarchical Hop eld network implies an invertible, implicit and iterative MLP-Mixer." In: ArXiv abs/2304.13061.

Pal, Avik, Alan Edelman, and Christopher Rackauckas (2022). \Continuous Deep Equilibrium Models: Training Neural ODEs faster by integrating them to In nity." In: Submitted to Transactions on Machine Learning Research url : https://openreview.net/forum?id=mlI9f7u6Zo .

Pineda, Fernando J. (1987). \Generalization of back-propagation to recurrent neural networks." In: Physical review letters 59.19, p. 2229. doi : 10.1103/PhysRevLett.59.2229 .

Ramsauer, Hubert et al. (2021). \Hop eld Networks is All You Need." In: International Conference on Learning Representations

Revay, Max, Ruigang Wang, and Ian Manchester (2021). \Lipschitz-Bounded Equilibrium Networks." In: Submitted to International Conference on Learning Representations url : https://openreview.net/forum?id=bodgPrarPUJ .

Salakhutdinov, Ruslan and Geo rey Hinton (2009). \Deep Boltzmann Machines." In: Arti cial Intelligence and Statistics. PMLR, pp. 448{455.

Scellier, Benjamin and Yoshua Bengio (2017). \Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation." In: Frontiers in Computational Neuroscience 11. doi : 10.3389/fncom.2017.00024 .

Scellier, Benjamin et al. (2023). \Energy-based learning algorithms for analog computing: a comparative study." In: Neural Information Processing Systems

Walker, Homer F. and Peng Ni (2011). \Anderson Acceleration for Fixed-Point Iterations." In: SIAM Journal on Numerical Analysis 49.4, pp. 1715{1735. doi : 10.1137/10078356X.

Wang, Lipo (1998). \On the dynamics of discrete-time, continuous-state Hop eld neural networks." In: IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 45.6, pp. 747{749. doi : 10.1109/82.686695 .

Winston, Ezra and J. Zico Kolter (2020). \Monotone operator equilibrium networks." In: Neural Information Processing Systems

Yi, Su-in et al. (2023). \Activity-di erence training of deep neural networks using memristor crossbars." In: Nature Electronics 6.1, pp. 45{51. doi : 10.1038/s41928-022-00869-w .

Appendix A. Advantages of viewing Hop eld networks as DEQs

Below, we describe the advantages that come with our transition from the energy-based view of Hop eld networks to the DEQ framework of Section 3.1.

First, we would like to highlight that the DEQ formulation fully encompasses the Hop- eld network's original ODE formulation. In fact, solving the DEQs in Eqs. (5) and (6) with a damped Picard iteration is mathematically equivalent to solving the model's ODE with the forward Euler method, where the damping factor corresponds to the time step size. Using specialized DEQ solvers allows for even faster convergence,[4] as shown in Section 4.

Additionally, while Hop eld networks are typically trained using backpropagation-through-time, DEQs o er more memory-e cient methods, such as recurrent backpropagation (Pineda, 1987; Almeida, 1987). While similar algorithms have been suggested for the energy-based setting (Scellier and Bengio, 2017), they aim to approximate these exact methods and are often sensitive to the exact choice of hyperparameters (Scellier et al., 2023).

Furthermore, unlike for Hop eld networks, the stability of DEQs is a widely studied area, that includes regularization terms and even parametrizations that are provably stable (Bai et al., 2019; Ghaoui et al., 2019; Winston and Kolter, 2020; Bai et al., 2021; Revay et al., 2021).

Moving past computational advantages, we argue that the DEQ framework is a more natural way of reasoning about the dynamics of Hop eld networks, as it o ers a compre- hensible, concise formulation that operates directly at the equilibrium level. Compared to the energy-based setting, the DEQ framework makes it signi cantly easier to study the characteristics of techniques like even-odd splitting, that act on the states at equilibrium.

While the close relationship between DEQs and Hop eld networks has been noticed before (Krotov, 2021; Laborieux and Zenke, 2023; Ota and Taki, 2023), remarkably, none of the many advantages described here are exploited in these works.

Appendix B. Derivation of DEQ formulation of CHN & HAM

Given the expression for the state update rule $\frac{ds}{dt}$ in a Hop eld network, we may implicitly de ne s^* as the equilibrium state for which $\frac{ds}{dt}(s^*) = 0$, thereby yielding the desired DEQ formulation. We work out the details below for the CHN and HAM.

DEQ formulation of CHN

For CHNs, $\frac{ds}{dt}$ is given by Eq. (2). Setting this to zero, we nd the following DEQ:

$$s^* = \rho'(s^*) \quad (W \quad \rho(s^*) + b): \tag{11}$$

At rst glance, this DEQ seems to be independent of the input and therefore always converge to the same s^*. However, recall that the input x is implicitly applied through the rst d states.

When the equilibrium state s^* is split up into the input x and hidden state $\tilde{s}^*$, i.e., $s^* = [x; \tilde{s}^*]$, we can reformulate Eq. (11) with an explicit input dependence as

$$\tilde{s}^* = \rho'(\tilde{s}^*) \quad (\tilde{W} \quad \rho(\tilde{s}^*) + \tilde{b} + U \quad (x)); \tag{12}$$

4. The DEQ solver cannot guarantee energy minimization and may move towards spurious extrema. In this work, however, we will assume that it always nds the true energy minimum.

where the tilde on $\tilde{s}$, $\tilde{W}$ and $\tilde{b}$ indicates a change in dimensions due to the slicing operation.

Example 3 In a 5-layer CHN like Fig. 1, we can easily derive that

$$\tilde{W} = \begin{bmatrix} 0 & W_1^T & 0 & 0 \\ W_1 & 0 & W_2^T & 0 \\ 0 & W_2 & 0 & W_3^T \\ 0 & 0 & W_3 & 0 \end{bmatrix}; \quad U = \begin{bmatrix} W_0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

DEQ formulation of HAM
For HAMs, the derivation is entirely identical as for CHNs. Starting from Eq. (4), we find

$$\tilde{s} = \tilde{W}\sigma(\tilde{s}) + \tilde{b} + U\sigma(x). \tag{13}$$

For readability, we leave out the tilde in the rest of the paper.

Appendix C. Splitting the states into even & odd: full derivation

Below, we provide a detailed derivation of Eqs. (7) and (8), serving as a proof for Theorem 2. We begin with the more concise derivation of even-odd splitting in HAMs. For the CHN, the process is entirely analogous, therefore, we provide only a rough sketch of the derivation.

C.1. Derivation of even-odd splitting in HAMs

First, we multiply both sides of Eq. (6) with a general permutation matrix P and find:

$$\begin{aligned} Ps &= PW\sigma(s) + Pb + PU\sigma(x) \\ &= PWP^TP\sigma(s) + Pb + PU\sigma(x) \\ &= PWP^T\sigma(Ps) + Pb + PU\sigma(x); \end{aligned} \tag{14}$$

where P can be brought inside σ, as it applies the same element-wise non-linearity over all states.[5] In even-odd splitting, P transforms $s = [s_1; s_2; s_3; \ldots]$ into $Ps = [s_{even}; s_{odd}]$, with $s_{even} = [s_2; s_4; \ldots]$ and $s_{odd} = [s_1; s_3; \ldots]$. Using this specific P, we find

$$PWP^T = \begin{bmatrix} 0 & W_P^T \\ W_P & 0 \end{bmatrix}; \qquad Ps = \begin{bmatrix} s_{even} \\ s_{odd} \end{bmatrix};$$

$$Pb = \begin{bmatrix} b_{even} \\ b_{odd} \end{bmatrix}; \qquad PU = \begin{bmatrix} 0 \\ U_{odd} \end{bmatrix}.$$

Example 4 In a 5-layer HAM like Fig. 1, we have

$$W_P = \begin{bmatrix} W_1^T & 0 \\ W_2 & W_3^T \end{bmatrix}; \quad U_{odd} = \begin{bmatrix} W_0 \\ 0 \end{bmatrix};$$

with the matrix at block position $(i; j)$ in W_P representing the influence of s_{2j+2} on s_{2i+1}. The locations of the zero matrices in W_P and U_{odd} correspond to skip connections, which are technically allowed, as long as they are between even and odd layers.[6]

5. For a more general σ, one should use a permuted version σ_P.
6. The second condition of Bengio et al. was phrased too restrictively.

With the proper substitutions in Eq. (14), we find

$$\begin{cases} s_{even} = W_P^T (s_{odd}) + b_{even} \\ s_{odd} = W_P (s_{even}) + b_{odd} + U_{odd} (x) \end{cases} \tag{15}$$

Therefore, the synchronous state update rule corresponding to the fixed point iteration procedure for the HAM, according to Eq. (6),

$$s^{n+1} = W (s^n) + b + U (x);$$

in which the state superscript n denotes the iteration index, can be written as

$$\begin{bmatrix} s_{even}^{n+1} \\ s_{odd}^{n+1} \end{bmatrix} = \begin{bmatrix} 0 & W_P^T \\ W_P & 0 \end{bmatrix} \begin{bmatrix} s_{even}^n \\ s_{odd}^n \end{bmatrix} + \begin{bmatrix} b_{even} \\ b_{odd} \end{bmatrix} + \begin{bmatrix} 0 \\ U_{odd} \end{bmatrix} (x);$$

or simplified,

$$\begin{cases} s_{even}^{n+1} = W_P^T (s_{odd}^n) + b_{even} \\ s_{odd}^{n+1} = W_P (s_{even}^n) + b_{odd} + U_{odd} (x) \end{cases}.$$

C.2. Derivation of even-odd splitting in CHNs

The derivation of even-odd splitting in CHNs is entirely analogous to the one for HAMs, with the subtle difference of the added 0-term. This mainly poses a challenge in finding the equivalent for Eq. (14), as it requires Proposition 5.

Proposition 5 For a permutation matrix P and vectors $a; b \in R^N$: $P(a \quad b) = Pa \quad Pb$

Using Proposition 5, we may start from

$$Ps = P\,{}^0(s) \quad (PW (s) + Pb + PU (x));$$

from which we can proceed, entirely analogously to the case of the HAM, to eventually find

$$\begin{cases} s_{even}^{n+1} = {}^0(s_{even}^n) \quad W_P^T (s_{odd}^n) + b_{even} \\ s_{odd}^{n+1} = {}^0(s_{odd}^n) \quad W_P (s_{even}^n) + b_{odd} + U_{odd} (x) \end{cases}.$$

Appendix D. Correspondence between CHN and HAM

As outlined in Section 2, the distinction between a CHN and a HAM is conventionally made on the basis of the energy functions. However, via the DEQ formulations of Section 3.1, we can demonstrate a correspondence between these two models, as formalized in Theorem 3. This appendix gradually builds up towards the theorem's proof.

Definition 6 In a well-behaved CHN, s is always unique and not identically zero.

Lemma 7 A CHN with an element-wise can only be well-behaved if ${}^0(0) \neq 0$.

Proof (Contraposition) If ${}^0(0) = 0$, then $s = 0$ is always an equilibrium state of the CHN. Thus, s is either identically zero or not unique, and the CHN is not well-behaved. ■

Lemma 8 In a well-behaved CHN with an element-wise , $^{0}(s)$ contains no zeros.

Proof (from Bengio and Fischer, 2015) Consider the i-th state s_i and assume $^{0}(s_i) = 0$. Then, in the i-th equation of Eq. (5), the right-hand side equals 0, reducing the whole to $s_i = 0$. However, by Lemma 7, we know that in a well-behaved CHN $^{0}(s_i) = 0 \implies s_i \neq 0$. This is a contradiction. ■

Theorem 3 (restated) Under relatively mild conditions for and up to an input preprocessing step, a well-behaved CHN can be transformed into a functionally equivalent HAM with e ective non-linearity $_{\&} := \ \& \ ^{1}$, where $\&(s) := s \ ^{0}(s)$, with representing the Hadamard division.

Proof For a well-behaved CHN, Lemma 8 allows us to rewrite the DEQ of Eq. (5) as

$$\&(s) = W \ (s) + b + U \ (x);$$

where we have introduced $\&(s) := s \ ^{0}(s)$, with representing the Hadamard (element-wise) division. Furthermore, if is chosen such that $\&$ is bijective (as is the case for most common choices of), we may introduce a change of variables $s_{\&} := \&(s)$ and rearrange the DEQ to

$$s_{\&} = W \ (\& \ ^{1}(s_{\&})) + b + U \ (x):$$

Under the substitution $_{\&} := \ \& \ ^{1}$, we nd

$$s_{\&} = W \ _{\&}(s_{\&})) + b + U \ _{\&}(\&(x)); \tag{16}$$

which coincides exactly with Eq. (6), the DEQ of a HAM, with non-linearity $_{\&}$ and input preprocessing using $\&$. ■

While Eq. (16) has direct access to the inverted function $\& \ ^{1}$, a CHN does not. Instead, it uses its xed point structure to approximate this inverse function during inference. Specifically, for a single state s_i, we have $s_i = \ ^{0}(s_i) \ C_i$, where C_i is a constant depending on the value of all other states, which we assume are kept xed here. Depending on $\&$, this xed point equation may converge very slowly, therefore requiring many iterations. If an e cient implementation of $\& \ ^{1}$ would be available, it would almost always be better to directly use Eq. (16) instead of Eq. (11). Conversely, certain activation functions may lead to a stable CHN, despite not resulting in a bijective $\&$. Nevertheless, for some of these CHNs, an equivalent HAM may still be formulated.

Example 5 A CHN with = ReLU, analytically extended such that $^{0}(0) = 1$, would result in a non-bijective $\&$. Nonetheless, it can quickly be recognized as a HAM with the same non-linearity, but where the states are limited to be strictly non-negative. Moreover, the HAM is linear, and an analytic expression for the equilibrium may be found. By contrast, it is not clear how stable a naive implementation of this CHN would be, and using the HAM counterpart would likely be the better choice.

Appendix E. Redundancy of synchronous updates

The state substitution in Eq. (9) reveals an interesting phenomenon arising in synchronously updated HAMs. First, it is important to reiterate that, as mentioned in Appendix A, iteratively applying Eqs. (7) and (8) is equivalent to minimizing the energy E from Eq. (3) by solving the ODE of Eq. (4) using the forward Euler method with synchronous state updates and a time step size equal to 1. As illustrated in Fig. 2, this scenario corresponds exactly to simultaneously solving two DEQs of the form of Eq. (10), one at time step n (solid), the other at $n+1$ (dashed). In other words, synchronously updating the states corresponds to solving two internal DEQs with independent state dynamics.

This redundancy is not beneficial. Below, we discuss two problems that arise in this case and describe how even-odd splitting avoids them. As an alternative solution, one may also turn to a particular state initialization, which induces dynamics identical to even-odd splitting, albeit at a lower computational efficiency.

Problem #1: Lack of convergence guarantees

Fig. 2 nicely illustrates how state convergence under synchronous updates can only be guaranteed over two time steps (i.e., a length-2 limit cycle exists), as has long been known for Hopfield networks (Koiran, 1994; Wang, 1998). Absolute convergence can only be achieved when both the solid and the dashed DEQ converge to the same equilibrium point. This may not always be the case, as it depends on both the input x and the specifics of the DEQ, such as state initialization, parameter values, and choice of non-linearity .

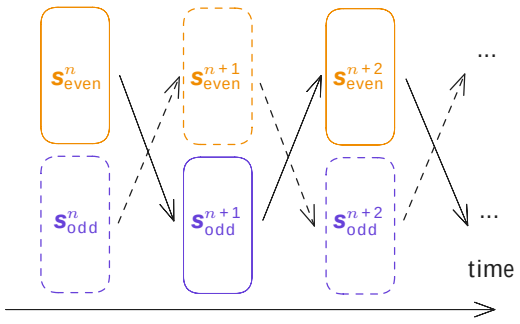

Figure 2: A view of synchronous updates across time reveals two separate even-odd DEQs (solid & dashed)

However, this behavior can easily be guaranteed by simply iterating a single DEQ (e.g., the solid one in Fig. 2) and defining the second DEQ as a time-shifted copy of the first one. This is precisely what even-odd splitting does and what Eq. (10) describes.

Problem #2: Partial gradient flow

In practice, the total number of iterations is typically kept fixed and determined in advance. In the case of Fig. 2, this means that whichever DEQ contains the output prediction $\hat{y}$ after the final iteration at time T, will always be the one receiving the gradient, which can then backpropagate through time, either explicitly or using memory-efficient DEQ methods. The other DEQ will not receive any gradient information whatsoever, whereas its parameters, which are shared with the first DEQ, are still updated.

Since the gradients depend on only a single DEQ, the parameters will adapt to the initial behavior of that DEQ, which may not be optimal for the other one. This might be problematic if the model is ever to run for a different number of iterations, with a different polarity. In Eq. (10), this is never a problem. As the DEQ internally advances two time steps at once, the final time T will always be even. Additionally, it never explicitly models the other DEQ, whose behavior at initialization is therefore irrelevant.

Inducing even-odd splitting through state initialization
State initialization plays an important role in guaranteeing convergence. To illustrate what might go wrong, let us assume that $n = 0$ in Fig. 2 and that s^0_{even} and s^0_{odd} are initialized at zero, as is commonly done. In the solid DEQ, s^1_{odd} receives information from both the input x and s^0_{even}, and updates its states accordingly. However, in the dashed DEQ, s^1_{even} only receives information from s^0_{odd}, which is not input-dependent, and uses that to update its states. This means that the dashed DEQ is not exactly a time-shifted version of the solid DEQ anymore: its initialization for s_{even} will be s^1_{even}, which does not necessarily equal s^0_{even}.

To avoid this discrepancy, we may design an initialization scheme such that $s^1_{\text{even}} = s^0_{\text{even}}$ by construction. Setting

$$s^0_{\text{even}} = W_P^T \ (s^0_{\text{odd}}) + b_{\text{even}}$$

guarantees this equality, as can be seen from Eq. (8). In fact, this initialization scheme directly induces even-odd splitting in synchronously updated HAMs. After all, we find that

$$s^0_{\text{even}} = s^1_{\text{even}} \Rightarrow \ s^1_{\text{odd}} = s^2_{\text{odd}} \Rightarrow \ s^2_{\text{even}} = s^3_{\text{even}} \Rightarrow \ \cdots;$$

which corresponds exactly to even-odd splitting, where the even/odd layers are alternately kept fixed for a single time step. Note, however, that synchronous updates still waste computations on these fixed values, making Eq. (9) the more sensible update rule to follow.

Appendix F. Experimental setup

Below is an overview that should contain all information required to reproduce the results from Section 4. The code is available at https://github.com/cgoemaere/hopdeq.

Data

- Dataset: EMNIST-MNIST (Cohen et al., 2017). This is a drop-in replacement for the MNIST dataset (LeCun, 1998), but with a known conversion process from the original NIST digits (Grother, 1995).
- Input preprocessing: rescaling pixel intensities from [0, 255] to [0, 1]
- Batch size: 64
- Epochs: 10
- No data augmentation

Model

- Architecture (with a constant amount of hidden neurons)
  - 3 layers: 784-1990-10
  - 5 layers: 784-1280-510-200-10
  - 7 layers: 784-1024-512-256-128-70-10

- Non-linearity : sigmoid(4x 2) (shifted sigmoid; same as Laborieux et al. (2020))
- State initialization: zero initialization, i.e., $s^{n=0} = \mathbf{0}$
- Weight initialization: Xavier initialization (Glorot and Bengio, 2010) per layer (not on full W, but on W_i), as we want bidirectional operation between layers. The biases were initialized using a normal distribution with mean 0.0 and standard deviation 0.01.
- Forward iterations (chosen large enough to ensure state convergence during training): 40 (3 layers), 80 (5 layers), 120 (7 layers)
- DEQ solver: Anderson acceleration with windows size $m = 4$, Tikhonov regularization (constant: 10 ^{10}), and safe-guarding
- Damping (tuned to maintain stability during training)
  - CHN: damping of 0.5, i.e., if the DEQ is $s = f(s)$, then we use $s^{n+1} = 0{:}5s^n + 0{:}5f(s^n)$ as update rule. From an ODE perspective, this means that time moves half as fast (i.e., step size $h = 0{:}5$). To compensate for that, we multiply the provided number of forward and backward iterations with a factor 2, so that the same amount of ODE time is simulated.
  - CHN-EO & HAM: no damping, i.e., if the DEQ is $s = f(s)$, then we use $s^{n+1} = f(s^n)$ as update rule.

Training

- Loss function: Mean Square Error
- Backward method: Recurrent Backpropagation (Pineda, 1987; Almeida, 1987) with Picard iteration (always; Anderson acceleration was unstable here)
- Backward iterations: 8 (3 layers), 16 (5 layers), 24 (7 layers)
- Optimizer
  - Type: Madam (Bernstein et al., 2020) (chosen as a substitute for layerwise learning rates; Madam automatically scales weight updates according to $jj\Delta Wjj{=}jjWjj$, as advised by Scellier and Bengio (2017))
  - Learning rate: 0.01 (3 layers), 0.005 (5 layers & 7 layers)
  - Learning rate decay: linear decay to $1{=}10^{\text{th}}$ of the initial learning rate mentioned above, over the course of the 10 epochs (inspired by Bernstein et al., 2020)
  - Hyperparameters (see implementation): p_scale = 1024; g_bound = 3
- No gradient clipping, dropout or other commonly used training techniques
- GPU: 1x GTX-1080Ti

Appendix G. Visual comparison of state dynamics in different configurations of Hopfield networks

In Figs. 3 and 4, we provide a visual comparison of the state dynamics in the different models from Section 4. First, notice how the use of DEQ solvers helps guarantee convergence in samples that would otherwise not have converged. Additionally, even-odd splitting seems to boost convergence speed overall, by a factor close to two, as expected. We can see that the initial dynamics of the models differ from their regular regime, as the trajectories of all samples start out similarly and only diverge after a few iterations. As for the low density region in the models using DEQ solvers (most noticeable in the bottom right subplots), we hypothesize that this is due to the solver occasionally finding the exact fixed point solution, bringing the relative residual to zero.

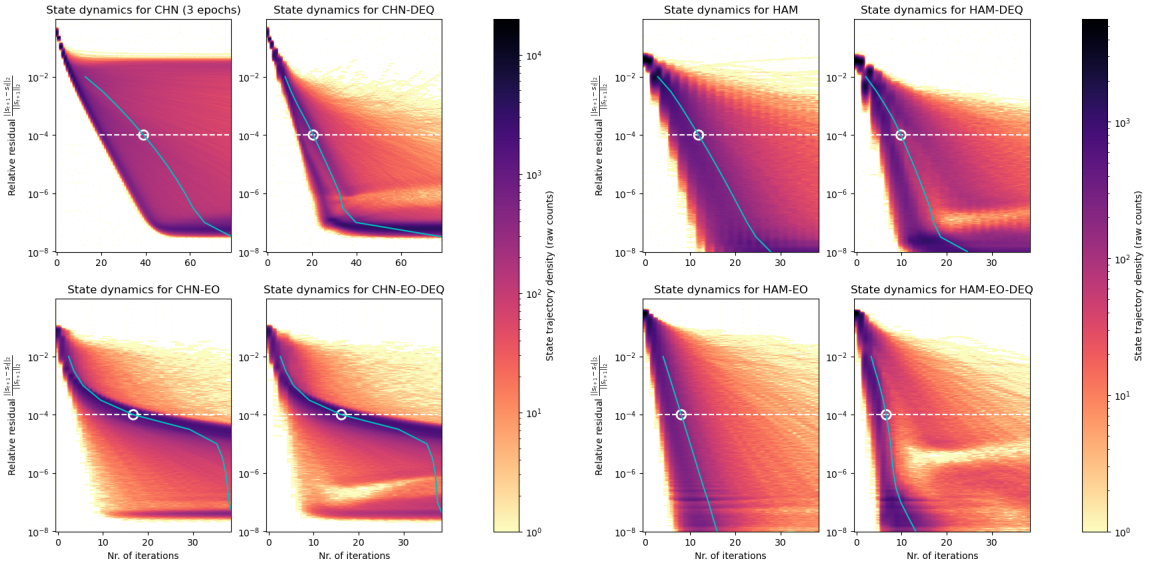

Figure 3: Density heatmap of the state trajectories for a 3-layer CHN (left) and HAM (right), and the impact of using DEQ solvers ('DEQ') and even-odd splitting ('EO'). The horizontal axis represents the number of iterations of the DEQ. The vertical axis represents the relative residual, which is used to determine the state convergence (the lower, the more converged). The limit of 10^{-4} as chosen criterion for convergence is indicated with a white dashed line. For every setting, we show the cumulative results of 5 different seeds, run on the entire MNIST test set. In cyan, we show the mean number of iterations corresponding to a given convergence criterion. The white circular marker at the limit of 10^{-4} corresponds to the value reported in Table 2.