nature electronics

SUPPLEMENTARY INFORMATION

https://doi.org/10.1038/s41928-020-0436-6

In the format provided by the authors and unedited.

Power-efficient combinatorial optimization using intrinsic noise in memristor Hopfield neural networks

Fuxi Cai[1,2,6], Suhas Kumar[1,6], Thomas Van Vaerenbergh[1,6], Xia Sheng[1], Rui Liu[1,3], Can Li[1], Zhan Liu[1], Martin Foltin[1], Shimeng Yu[4], Qiangfei Xia[5], J. Joshua Yang[5], Raymond Beausoleil[1], Wei D. Lu[2] and John Paul Strachan[1]✉

[1]Hewlett Packard Labs, Hewlett Packard Enterprise, Palo Alto, CA, USA. [2]Department of Electrical Engineering, University of Michigan, Ann Arbor, USA. [3]Department of Electrical and Electronics Engineering, Arizona State University, Tempe, AZ, USA. [4]School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA. [5]Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA. [6]These authors contributed equally: Fuxi Cai, Suhas Kumar, Thomas Van Vaerenbergh. ✉e-mail: john-paul.strachan@hpe.com

SUPPLEMENTARY INFORMATION for "Power efficient combinatorial optimization using intrinsic noise in memristor Hopfield neural networks"

CONTENTS:

1 Problem landscape

Good examples of the enterprise-relevant NP-hard problems that future combinatorial optimization accelerators should target to solve can be found in the benchmark sets provided by the the Mixed Integer Programming (MIP) community. Whereas we mentioned in our introduction that Ref.[1] demonstrated that some typical papers discussing the solution of QUBO's using exact algorithms were focusing typically on problem sizes of a few 100 variables on a single CPU, it should be noted that plenty of larger-scale MIP instances are known to be solvable using an out-of-the-box solver on a standard desktop in even less than one hour (cfr. the 'easy' category in MIPLIB2017[2]). However, the latter, 'easier' problems do not necessarily have a quadratic term (and are hence not a QUBO), are relatively sparse, or have another problem-specific feature in which the usage of the extensive range of exact methods incorporated in software frameworks such as Gurobi or CPLEX can be beneficial. The Max-Cut problem instances discussed in this paper, tend to be not easily solvable (using the MIPLIB-definition of easy) on a single CPU for problems of system size $N = 60, 100$. Nevertheless, it is important to acknowledge that when mapping problem instances in the MIP-format to the QUBO-formalism that corresponds with our annealing accelerator, we might, for some problem instances, miss opportunities to use problem-specific tricks. This is one of the reasons why we believe that future architectures relying on annealing accelerators, should use hybrid algorithms that contain some form of pre-processing and can identify whether running either the entire problem or some of its sub-problems on accelerator hardware is justified or not, by potentially even allowing to run analogue accelerators, digital meta-heuristics and/or exact techniques in parallel where appropriate.

2 Mapping the Max-Cut Problem to Hopfield Network

Maximum cut or the Max-Cut problem is a classic NP-hard problem which finds a maximum cut in a graph[3]. In the Max-Cut problem, a partition of S of all the vertices V needs to be determined such that the number of edges (which are referred to as the "Cut") between S and its complementary subset is maximized. In a graph $G(V,E)$, a cut vector can be defined as:

$$x_i = \begin{cases} +1 & i \in S \\ -1 & i \in V \backslash S, \end{cases} \tag{4}$$

where all the vertices in S are represented by +1, and all the vertices in its complementary set are noted as -1. Based on references[3,4], we derive the Hamiltonian for the Max-Cut problem starting from the following figure of merit:

$$F.O.M. = \max \sum_{ij \in \sigma(S)} a_{ij} = \max \sum_{i<j} a_{ij} \frac{1 - x_i x_j}{2}, \quad \sigma(S) = ij \in E : i \in S, j \in V \backslash S, \tag{5}$$

where a_{ij} is the weight of the so-called Adjacency Matrix A.

This optimization problem can be transformed to a Hopfield energy function as follows:

$$\begin{aligned} E &= \min \sum_{i<j} a_{ij} \frac{x_i x_j - 1}{2} \\ &= \min \frac{1}{2} \left(\sum_{i<j} a_{ij} x_i x_j - \sum_{i<j} a_{ij} \right) \\ &= \min \frac{1}{2} \sum_{i<j} a_{ij} x_i x_j - C \\ &= \min \frac{1}{2} \sum_{i<j} a_{ij} x_i x_j \\ &= \min -\frac{1}{2} \sum_{i<j} (-a_{ij}) x_i x_j \end{aligned} \tag{6}$$

Consequently, the Max-Cut problem can be solved by minimizing the following Hopfield network energy function:

$$E = -\frac{1}{2} \sum_{i<j} w_{ij} x_i x_j, \quad w_{ij} = -a_{ij}, \tag{7}$$

and the weight matrix of the Hopfield network is the negative Adjacency matrix

$$w_{ij} = -a_{ij} \tag{8}$$

where the number of Max-Cut can be derived by:

$$Max\ Cut = -\frac{1}{2} \sum_{i<j} w_{ij} - E \tag{9}$$

3 Working principles of the hysteretic threshold function

In this section, we show that using a hysteretic threshold function with $w < 0$ can enable fluctuations even in an otherwise ideal system. We utilize the the following update process for a neuron $v_{i,t}$ at a specific location i and at a specific instance in time (iteration for a discrete-time system) t to implement a hysteretic threshold function:

$$\begin{aligned} u_{i,t} &= \sum_{j \neq i} W_{ij} v_{j,t} \\ \theta_{\text{hys:}i,t} &= \theta_{i,t} - w_t v_{i,t} \\ v_{i,t+1} &= \begin{cases} +1 & \text{if } u_{i,t} \geq \theta_{\text{hys:}i,t} \\ -1 & \text{if } u_{i,t} < \theta_{\text{hys:}i,t} \end{cases} \\ u_{i,t+1} &= \sum_{j \neq i} W_{ij} v_{j,t+1}, \end{aligned} \tag{10}$$

where θ_{hys} is the hysteretic threshold, w_t is the width of the hysteresis that may be tuned as a function of time or iterations of calculations and $v_{i,t+1}$ is the next (updated) state of the neuron that will be used in the next time step or iteration. For the below analysis, we use $\theta_{i,t} = 0$, consistent with the rest of this study.

3.1 Evidence that $w < 0$ can produce fluctuations in a noise-free system

Here we will show that the case of $w < 0$ when using Eq. 10 can produce fluctuations in the output (state of the system) even when there are no other sources of noise in the system. For the following analysis, we will consider updating of only one output, and without loss of generality, for simplicity, we will discuss the case in which all the other outputs of the system are kept constant. Since we are flipping only the neuron at the position i and there is no self feedback, $u_{i,t+1} = u_{i,t}$.

Case A: $w_t = x$, where $x > 0$

Case A1: $v_{i,0} = -1$
$\theta_{\text{hys:}i,0} = -(w_t).(-1) = x$

Case A1(i) Suppose that $u_{i,0} > x$,
Then $v_{i,1} = +1$
$\theta_{\text{hys:}i,1} = -(w_t).(+1) = -x$
$u_{i,1} = u_{i,0}$.
From suppositions of Case A and Case A1(i), $u_{i,1} > -x$
Then $v_{i,2} = +1$
For all further iterations the output v does not flip, and is held at +1.

Case A1(ii) Suppose that $u_{i,0} < x$,
Then $v_{i,1} = -1$
$\theta_{\text{hys:}i,1} = -(w_t).(-1) = +x$
$u_{i,1} = u_{i,0}$.
From suppositions of Case A and Case A1(ii), $u_{i,1} < x$
Then $v_{i,2} = -1$
For all further iterations the output v does not flip, and is held at -1.

A similar analysis of all possible cases within Case A leads to the conclusion that for $w_t > 0$, the output does not continually fluctuate but is instead reinforced at a specific value.

Case B: $w_t = -x$, where $x > 0$

Case B1: $v_{i,0} = -1$
$\theta_{\text{hys:}i,0} = -(w_t).(-1) = -x$

Case B1(i) Suppose that $u_{i,0} > -x$,
Then $v_{i,1} = +1$
$\theta_{\text{hys:}i,1} = -(w_t).(+1) = +x$
$u_{i,1} = u_{i,0}$.
Case B1(i)(a): It is possible from suppositions of Case B and Case B1(i) that $u_{i,1} < x$
Then $v_{i,2} = -1$
For every further iteration the output v flips between +1 and -1.
Case B1(i)(b): It is possible from suppositions of Case B and Case B1(i) that $u_{i,1} > x$
Then $v_{i,2} = +1$
For all further iterations the output v does not flip, and is held at +1.

Case B1(ii) Suppose that $u_{i,0} < -x$,
Then $v_{i,1} = -1$
$\theta_{\text{hys:}i,1} = -(w_t).(-1) = -x$
$u_{i,1} = u_{i,0}$.
From suppositions of Case B and Case B1(ii), $u_{i,1} < -x$
Then $v_{i,2} = -1$
For all further iterations the output v does not flip, and is held at -1.

Therefore for $w_t < 0$, there is a possibility that, even if the input of a neuron does not change, the output can continually flip and is not reinforced. This implies that fluctuations in the output are created even when there are no other sources of noise in the system.

3.2 Evidence that intrinsic noise is suppressed for $w > 0$ and propagated for $w < 0$

Here we will show that intrinsic noise in the system that may provide small-magnitude fluctuations will be suppressed for $w > 0$ and can be propagated for $w < 0$, thereby enabling simulated annealing without requiring an external source of noise or pseudo random numbers.

3.2.1 Case $w > 0$

Let us suppose that $w_t = x$, where $x > 0$. We are flipping only the neuron at the position i and since there is no self feedback, $u_{i,t+1} = u_{i,t}$.

Suppose that $v_{i,0} = +1$

$\theta_{\text{hys:}i,0} = -(w_t).(1) = -x$

Suppose that $-x < u_{i,0} < +x$,

$v_{i,1} = +1$

$\theta_{\text{hys:}i,1} = -(w_t).(+1) = -x$

$u_{i,1} > -x$

$v_{i,2} = +1$

For all further iterations the output v does not flip, and is held at $+1$.

Case A: Now introduce a small change in $u_{i,2}$ due to intrinsic noise of δ such that $x < u_{i,2} - \delta > -x$

$\theta_{\text{hys:}i,2} = -(w_t).(+1) = -x$

$-x < u_{i,2} - \delta < +x$

$v_{i,3} = +1$

For all further iterations, v does not flip and is held at $+1$.

Case B: Now introduce a large change in $u_{i,2}$ due to intrinsic noise of δ such that $u_{i,2} - \delta < -x$

$\theta_{\text{hys:}i,2} = -(w_t).(+1) = -x$

$u_{i,2} - |\delta| < -x$

$v_{i,2} = -1$

$\theta_{\text{hys:}i,2} = -(w_t).(-1) = +x$

$u_{i,3} - \delta < +x$,

$v_{i,3} = -1$

Although v flipped once due to excessive noise, for all further iterations, v does not flip and is held at -1.

3.2.2 Case $w < 0$

Let us suppose that $w_t = -x$, where $x > 0$. Consider a Hopfield network defined by the symmetric and zero-diagonal weights matrix

$$W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} = \begin{bmatrix} 0 & 0.5 \\ 0.5 & 0 \end{bmatrix}. \tag{11}$$

Let us suppose that $v = \begin{bmatrix} v_{1,0} & v_{2,0} \end{bmatrix} = [+1 + 1]$, $x = 0.4$, and we update the system asynchronously (sequentially).

$u_{1,0} = v_{2,0}.W_{21} = (+1).(0.5) = +0.5$

$\theta_{\text{hys:}1,0} = -(w_t).(+1) = +0.4$

$u_{1,0} > \theta_{\text{hys:}1,0}$

$v_{1,1} = +1$

$u_{2,0} = v_{1,1}.W_{12} = (+1).(0.5) = +0.5$

$\theta_{\text{hys:}2,0} = -(w_t).(+1) = +0.4$

$u_{2,0} > \theta_{\text{hys:}2,0}$

$v_{2,1} = +1$ (end of iteration 1, v does not change)

$u_{1,1} = v_{2,1}.W_{21} = (+1).(0.5) = +0.5$

$\theta_{\text{hys:}1,1} = -(w_t).(+1) = +0.4$

$u_{1,1} > \theta_{\text{hys:}1,1}$

$v_{1,2} = +1$

$u_{2,1} = v_{1,2}.W_{12} = (+1).(0.5) = +0.5$

$\theta_{\text{hys:}2,1} = -(w_t).(+1) = +0.4$

$u_{2,1} > \theta_{\text{hys:}2,1}$

$v_{2,2} = +1$ (end of iteration 2, v does not change)

For all further iterations, the output vector $\begin{bmatrix} v_{1,t} & v_{2,t} \end{bmatrix}$ is held at $[+1 + 1]$. Now that we have identified a stable configuration of this system that does not undergo any fluctuations, let us study the effect of introducing a small degree of noise represented by $\delta = 0.2$ in the weight W_{21} from $t = 1$.

$u_{1,0} = v_{2,0}.W_{21} = (+1).(0.5) = +0.5$

$\theta_{\text{hys:}1,0} = -(w_t).(+1) = +0.4$

$u_{1,0} > \theta_{\text{hys:1,0}}$
$v_{1,1} = +1$
$u_{2,0} = v_{1,1}.W_{12} = (+1).(0.5) = +0.5$
$\theta_{\text{hys:2,0}} = -(w_t).(+1) = +0.4$
$u_{2,0} > \theta_{\text{hys:2,0}}$
$v_{2,1} = +1$ (end of iteration 1, v does not change)
$u_{1,1} = v_{2,1}.(W_{21} - \delta) = (+1).(0.3) = +0.3$
$\theta_{\text{hys:1,1}} = -(w_t).(+1) = +0.4$
$u_{1,1} < \theta_{\text{hys:1,1}}$
$v_{1,2} = -1$
$u_{2,1} = v_{1,2}.W_{12} = (-1).(0.5) = -0.5$
$\theta_{\text{hys:2,1}} = -(w_t).(+1) = +0.4$
$u_{2,1} < \theta_{\text{hys:2,1}}$
$v_{2,2} = -1$ (end of iteration 2, v flips in sign)
$u_{1,2} = v_{2,2}.(W_{21} - \delta) = (-1).(0.3) = -0.3$
$\theta_{\text{hys:1,2}} = -(w_t).(-1) = -0.4$
$u_{1,2} > \theta_{\text{hys:1,2}}$
$v_{1,3} = +1$
$u_{2,2} = v_{1,3}.W_{12} = (+1).(0.5) = +0.5$
$\theta_{\text{hys:2,2}} = -(w_t).(-1) = -0.4$
$u_{2,2} > \theta_{\text{hys:2,2}}$
$v_{2,3} = +1$ (end of iteration 3, v flips in sign)
$u_{1,3} = v_{2,3}.(W_{21} - \delta) = (+1).(0.3) = +0.3$
$\theta_{\text{hys:1,3}} = -(w_t).(+1) = +0.4$
$u_{1,3} < \theta_{\text{hys:1,3}}$
$v_{1,4} = -1$
$u_{2,3} = v_{1,4}.W_{12} = (-1).(0.5) = -0.5$
$\theta_{\text{hys:2,3}} = -(w_t).(+1) = +0.4$
$u_{2,3} < \theta_{\text{hys:2,3}}$
$v_{2,4} = -1$ (end of iteration 4, v flips in sign)
$u_{1,4} = v_{2,4}.(W_{21} - \delta) = (-1).(0.3) = -0.3$
$\theta_{\text{hys:1,4}} = -(w_t).(-1) = -0.4$
$u_{1,4} > \theta_{\text{hys:1,4}}$
$v_{1,5} = +1$
$u_{2,4} = v_{1,5}.W_{12} = (+1).(0.5) = +0.5$
$\theta_{\text{hys:2,4}} = -(w_t).(-1) = -0.4$
$u_{2,4} > \theta_{\text{hys:2,4}}$
$v_{2,5} = +1$ (end of iteration 5, v flips in sign)

For every further iteration, the output vector $\begin{bmatrix} v_{1,t} & v_{2,t} \end{bmatrix}$ flips in sign. Therefore, for the case of $w < 0$, a small noise or error within the system can cause the system to undergo fluctuations.

This shows that for $w < 0$ in a hysteretic threshold can both introduce fluctuations in an otherwise noise-free ideal system, and also 'amplify' small magnitudes of noise in the system.

4 Possible circuit implementation of the hysteretic threshold function

Fig. S1a is a proposed circuit implementation of the hysteretic threshold function. This combines two comparators C_1 and C_2 biased with 1 V and 0 V each, with the input feeding into C_1's positive input terminal (in_1^+) and the output being read from the positive output of C_1 (out_1^+). Two D flip-flops are included to hold the outputs of the comparators, while the pass transistor circuit determines the threshold value to be applied at the next clock cycle. Figs. S1b-d show results of circuit simulations of the comparator circuit for the extreme cases that will introduce a corner case latency: only one bit in the high conductance state in a given column of 60 devices (for a problem size of $N = 60$), which will lead to a difference between the threshold and input values of 8 mV. It is apparent that the maximum latency is 0.3 ns. While this analysis coarsely serves to estimate the maximum latency, we acknowledge that a more detailed analysis, along with experiments, would be needed to capture other analogue corner cases (e.g., offsets in the TIA). We have included the simulated performance metrics of this circuit in the performance projection in Table 1.

Suppose that the threshold voltages are $\theta_1 = 0.2$ V and $\theta_2 = 0.8$ V.

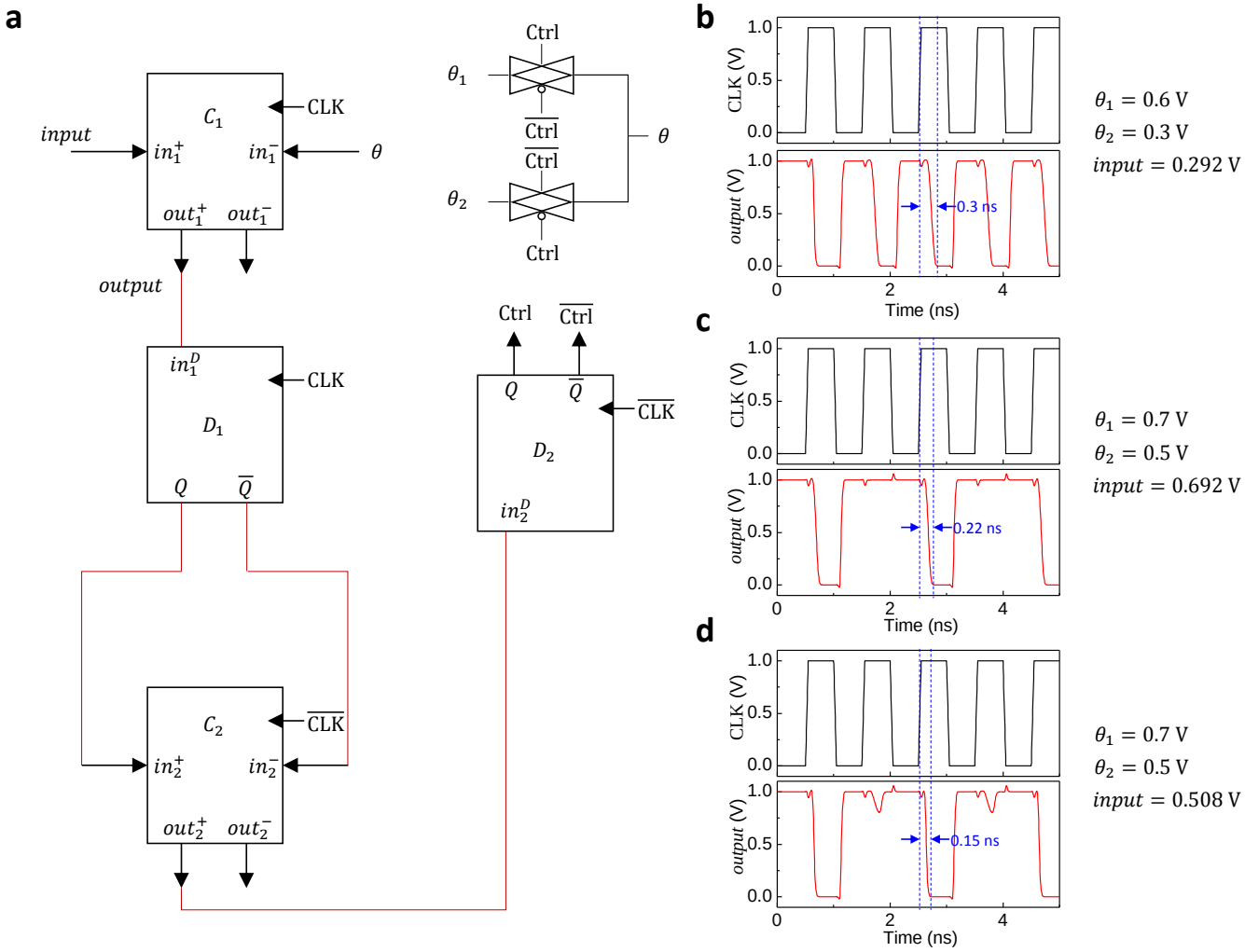

Figure S1. Proposed comparator circuit for implementation of the hysteretic threshold. a) One possible circuit implementation for the hysteretic threshold function with two comparators, two D flip-flops and two pass transistors. b)-c) are clock and output signals for three extreme cases with the input differing from the threshold values by 8 mV, as annotated. The time delays between the clock and the settling of the comparator output are marked in blue. The extreme case of a difference of 8 mV provides the longest settling time of 0.3 ns, while for all other cases, the settling time is much shorter.

Case A: Suppose that $input = 0.4$ V (such that $\theta_1 > input > \theta_2$) and initial state is $out_2^+ = 1$ V.

At clock cycle 1:

Ctrl $= 1$ V, $v_{in_1^-} = \theta = \theta_1$

$v_{in_1^-} > input$

$v_{out_1^+} = 0$ V, $(v_{out_1^+} < v_{out_1^-})$

$v_{out_2^+} = 0$ V

At clock cycle 2:

Ctrl $= 0$ V, $v_{in_1^-} = \theta = \theta_2$

$v_{in_1^-} < input$

$v_{out_1^+} = 1$ V, $(v_{out_1^+} > v_{out_1^-})$

$v_{out_2^+} = 1$ V

For every further iteration, the output ($v_{out_1^+}$) flips between 0 V and V_{DD} (or '1').

Case B: Suppose that $input = 0.9$ V (such that $input > \theta_1$) and initial state is $out_2^+ = 1$ V.

At clock cycle 1:

Ctrl $= 1$ V, $v_{in_1^-} = \theta = \theta_1$

$v_{in_1^-} < input$

$v_{out_1^+} = 1$ V, $(v_{out_1^+} > v_{out_1^-})$

$v_{out_2^+} = 1$ V

At clock cycle 2:

Ctrl $= 1$ V, $v_{in_1^-} = \theta = \theta_1$

$v_{in_1^-} < input$

$v_{out_1^+} = 1$ V, $(v_{out_1^+} > v_{out_1^-})$

$v_{out_2^+} = 1$ V

For all further iterations, the output ($v_{out_1^+}$) does not flip and is held at 1 V.

The calculations above exhibit the same behavior expected of a hysteretic threshold for $w < 0$ where for $-|w| < input < |w|$, the output flips continually. And for *input* outside the range of the hysteretic width (e.g.: $-|w| < |w| < input$), the output does not flip and is held constant. In this case, θ_1 and θ_2 define the tunable width (w) of the hysteresis centered at 0.5 V. This can be scaled up or down depending on the voltages that are used to represent '1' in the mem-HNN.

A hysteretic threshold with $w > 0$ is essentially a Schmitt trigger, a commonly used circuit. Consider $\theta_1 < \theta_2$, with $\theta_1 = 0.2$ V and $\theta_2 = 0.8$ V, centered at 0.5 V.

Case A: Suppose that $input = 0.4$ V (such that $\theta_2 > input > \theta_1$) and initial state is $out_2^+ = 1$ V.

At clock cycle 1:

Ctrl $= 1$, $v_{in_1^-} = \theta = \theta_1$

$v_{in_1^-} < input$

$v_{out_1^+} = 1$ V, $(v_{out_1^+} > v_{out_1^-})$

$v_{out_2^+} = 1$ V

A similar analysis of all possible cases leads to the conclusion that for all further iterations, the output ($v_{out_1^+}$) does not continually flip. This is precisely the behavior expected from a hysteretic threshold for $w > 0$.

5 Investigation of the effect of different noise profiles for simulated annealing

In this section, we compare the effect of using different noise functions in HNNs and explain why we choose a quadratic superlinear decaying profile as our default noise decaying function in the main text of the article.

For this purpose, we choose four qualitatively different decaying profiles for our random noise function: linear decay, quadratic superlinear decay, quadratic sublinear decay and exponential decay, to optimize the effect of injecting noise. The mathematical expressions of the four different decay profiles are shown in Fig. S2. These four decaying noise functions are also compared with the fixed noise function used in Fig. 2b. In this comparison, all the initial noise levels of the four decaying profiles are set to 5, which we identified to be the optimal initial noise level for decaying noise of the 60 node Max-Cut problem. The comparison result is shown in Fig. S3.

Therefore, unless mentioned otherwise, to obtain the best energy performance when injecting noise, we select quadratic superlinear decaying noise in the HNN implementation in the main text of the article.

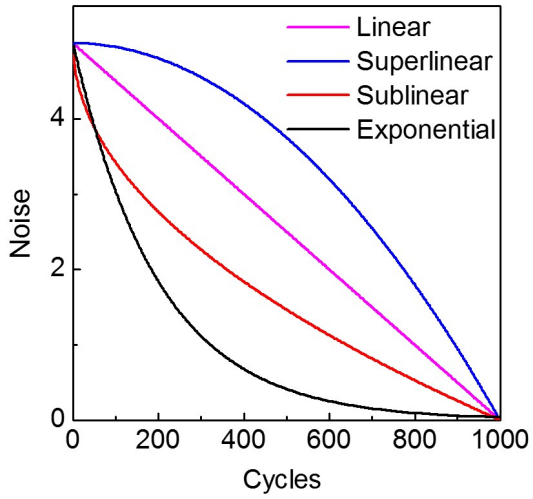

Figure S2. Different noise profiles for simulated annealing. Four different decaying profile illustrated for a given annealing time $T = 1000$, and initial noise level 5.

6 Function of noise term in the stochastic mem-HNN

In this section, we first give an overview of different ways noise have been algorithmically deployed in HNNs, followed by some context related to the specific design choice made in this paper.

6.1 Historic overview

The HNN has originally been proposed as an asynchronously updated discrete-time system with binary nodes without explicit noise terms[5]. Whereas it can be proven that this system will eventually converge to a stable fixed point[6], which fixpoint it relaxes into is both update scheme and initial condition dependent, and due to the limited set of neighboring states for a given state, it is easy for the network to get stuck in a local minima. Consequently, different versions of continuous-time HNN have been proposed, which should theoretically reach the global minima in a more efficient way than discrete-time HNNs, but suffer, amongst other things, from errors induced by the need to discretize the equations to solve them in software[7,8]. Using noise as a resource to avoid local minima was proposed using schemes that provide the theoretically optimal noise distribution[9], which is either computationally intensive or challenging to obtain in hardware. In other work, bounds on the noise have been derived that determine under which noise conditions the HNN will still converge to the desired limit set[10]. Boltzmann machines do not have explicit noise terms, but they are, similarly to Hopfield networks, discrete time systems and its updates are determined by asynchronously sampling the nodes of the network one-by-one following a sigmoid activation function[11]. Whereas Boltzmann machines have multiple applications, combining this system with simulated annealing allows to find ground states of the QUBO-system which is encoded in its Hamiltonian. As an alternative to this stochastic behavior, the mean-field annealing algorithm has been proposed, in which the output of the nodes is again determinstic, but instead of binary it is now real-valued, and follows the previously mentioned sigmoid activation function, of which the strength of the slope at the origin is adiabatically increased until it approximates the original binary heaviside (or, as in this work, a sign function when using a tanh as activation function with bipolar node states). A recent addition to that algorithm was to also explicitly include noise terms in the update equations, making the model again stochastic. This is called the noisy mean-field algorithm[12]. The latter approach is physics-inspired, based on the CIM results by Ref.[13], but so far, to the best of our knowledge, no published study has explicitly quantified how much this additional noise improves the solution quality compared to the original mean-field annealing algorithm. The noisy mean-field model was mainly proposed in a phenomenological way to explain the success probability scaling as a function of problem size for the CIM results.

In the world of physics-based hardware accelerators, the importance of noise has been discussed multiple times: recent proposals also show the benefit of having either classical or quantum fluctuations to enhance the chance to find the optimal solutions of NP-hard problems in Hopfield or Ising-based systems[13–20]. This is in line with other assessments that noise in electronics can be useful[21], as it plays an important role in some neural information processing schemes[22,23]. Based on the success of recent progress in physics-based accelerators, new software-algorithms for combinatorial optimization have been

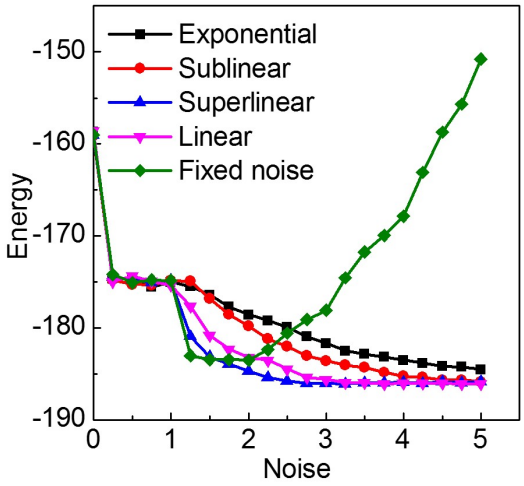

Figure S3. Simulated annealing noise profiles and Hopfield energy. We compare five different noise profiles: four different decaying noise profiles and one fixed noise profile. The initial noise level of all four decaying profiles is set to be 5. For this benchmark task, the quadratic superlinear decaying profile leads to the best energy result.

developed, sometimes explicitly based on the phenomenological behavior of the aforementioned physics-based accelerators, and these have been deployed in 'traditional' digital accelerators like GPUs[12] or FPGAs[24] - these algorithms all have an important stochastic or noise-induced component.

6.2 Link between the stochastic mem-HNN and a Boltzmann machine

In the simulations of this paper (e.g., Fig. 2), we are using decaying additive uniform noise in a discrete-time Hopfield Neural Network, an idea which has been proposed in the context of discrete-time HNNs for solving the shortest path problem[25], and some design criteria have theoretically been proposed in Ref.[26]. Neither of these two papers discloses that having a discrete-time HNN with binary states and additive uniform white noise can be seen as an approximated version of a Boltzmann machine, where the sigmoid function, i.e., the function typically used as the stochastic activation function, is now replaced by a (three-piece) piecewise linear approximation. In this paper, as intuitively expected, when performing simulated annealing by having the amplitude of the noise term gradually decay as a function of time, this can be interpreted as an approximation of a corresponding simulated annealing process in a traditional Boltzmann machine. Using such coarse piecewise linear approximations of the sigmoid function has been previously proposed to simplify the hardware implementation of a Boltzmann machine in an FPGA[27], but in that case the piecewise linear function was explicitly constructed in hardware, resulting in avoidable circuit overhead. In contrast, in our system we avoid the need to explicitly implement the piecewise linear system and restrict the hardware requirements to a simple comparator. More elaborate piecewise linear approximations have been successfully used in the context of FPGA-implementations of restricted Boltzmann machines for deep learning applications where the convergence degradation is claimed to be negligible[28,29] . One way to let the acceptance or rejection of a stochastic bit follow an activation function which follows the sigmoid more closely is by having a noise distribution which is Gaussian or a sech (the latter option being exact).

One recent implementation of simulated annealing in a mem-HNN proposed the use of hardware-generated noise in Cu-based CBRAM devices, as these devices have a tunable stochasticity[30]. In that scenario, a scheme similar to the one implemented in Fujutsi's digital annealer (DA)[24] was used in the sense that one spin (or a small set of spins) was selected at a time, to calculate the energy-difference ΔH which would be the result of a spin-flip, and this energy difference was then used to set the probability of the Cu-based CBRAM device by having an input pulse with length proportional to ΔH. Whereas it is clearly beneficial to use hardware as an intrinsic noise source over the usage of RNG implemented in digital electronics, the specific implementation is rather slow (*ms* timescale), and contains additional circuit complexity compaired to the one presented in this paper.

7 Design, area, and power for crossbar mem-HNN

The area and energy values of the circuits in Fig. 1a are summarized in Tables S1 and S2 for a 60×60 memristor array for two different technology nodes.

First, in Table S1, we use NeuroSim+[31] at 32 nm to model the area and energy for all the circuits required in Fig. 1. In Ref.[31], all the peripheral circuit modules have been validated with SPICE simulations at different tech nodes (including 32 nm) and physical layouts. The memristor cell size is assumed to be 12 F^2 ($F = 32$ nm). To estimate total energy or power values, we consider 50 cycles of operation, which may vary depending on the problem, accuracy of solution, etc. In the evaluation of this table, we assume that $R_{on} = 100\,\text{k}\Omega$ and $R_{off} = 1\,\text{M}\Omega$, with the percentage of ON/OFF cells being 50%, corresponding to a 50% density of the weights matrix of dense max-cut problems.

We performed simulations to determine the delay of the critical path during read or evaluation operation. Along with the comparator, we included energy and latency estimates for a transimpedance amplifier (TIA). Based on the delays of the different subcomponents of the proposed mem-HNN architecture, the critical path delay can be inferred to be approximately 0.9 ns. Since a full chip design will usually add a latency overhead, we included a latency overhead factor of 2 to arrive at a total latency of approximately 2 ns, corresponding to the clock frequency of 500 MHz.

Table S1. Area and energy (per clock cycle) breakdowns for circuits in Fig. 1a calculated based on 32 nm technology node.

Component	Area (μm^2)	Latency (ns)	Powered-on time (ns)	Energy (pJ)
I/O buffer	245.12	0.1	2	0.023
SW matrix & drivers	175.32	0.501	0.501	0.270
Memristor crossbar (60 cols.)	201.30	<0.3	0.3	0.338
MUX	939.52	0.0040	0.0040	0.013
MUX Decoder (6-bit)	340.57	0.1237	0.1237	0.110
Comparator +TIA (for 10 counts)	140	0.3	0.3	0.51
Total (seq.)	2041.836	0.92	2	31.74
Total (par.)	22460.196	0.92	2	349.14

From Table S1, using scaling protocols from leading foundry data, we estimated the performance metrics for a 16 nm design to be consistent with the GPU considered in Table 1. The results are provided in Table S2.

Table S2. Area and energy (per clock cycle) breakdowns for circuits in Fig. 1a scaled to 16 nm technology node.

Component	Area (μm^2)	Latency (ns)	Powered-on time (ns)	Energy (pJ)
I/O buffer	110.30	0.1	2	0.008
SW matrix & drivers	78.89	0.501	0.501	0.089
Memristor crossbar (60 cols.)	338	<0.3	0.3	0.338
MUX	153.26	0.0040	0.0040	0.004
MUX Decoder (6-bit)	153.26	0.1237	0.1237	0.036
Comparator + TIA (for 10 counts)	140	0.3	0.3	0.26
Total (seq.)	1243.24	0.92	2	10.9
Total (par.)	13675.65	0.92	2	120.1

In Table S1 and S2, the total energy (per clock cycle) was calculated by summing the energy consumption of the I/O buffer, SW matrix drivers, MUX and MUX Decoder across 60 effective columns of the memristor array, while the energy consumption of the memristor crossbar array and comparator were considered for only 10 columns of the array.

In Table 1, to calculate the power consumption for dense unweighted Max-Cut for $N = 60$, we used the value of the energy consumption per clock cycle provided in Table S2 for the $N = 60$ system, assuming a 500 MHz clock frequency and a 10 column batch update (the leakage power is negligible). In addition, we included a $2\times$ overhead, which includes a conservative estimate of the additional power needed to cool the memristor crossbar array. This $2\times$ overhead estimate originates from the notion that the power consumption for the CPU solution, 20 W/core, is based on the total power consumption of a data center[32], divided by its number of cores. Equivalently, for a HPC-type of infrastructure consisting of mem-HNN cores, we would need to include the overhead included in the power usage effectiveness (PUE) of the HPC infrastructure. On average, typical PUE values have recently been decreasing[33] to ~ 1.6. In addition, the overhead due to the power consumption of

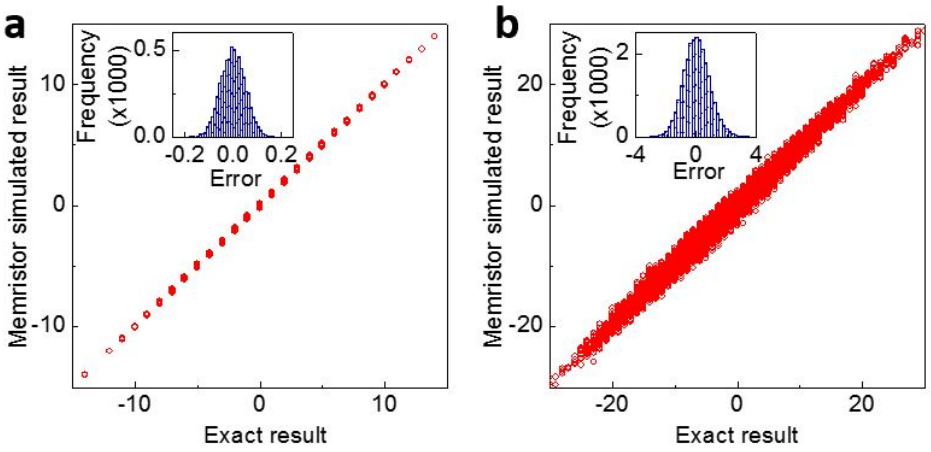

Figure S4. Errors and array sizes. a) Computations in arrays of size 32×32 have non-overlapping analogue results and error distributions (inset) lower than 0.5, matching digital results when rounded to the nearest integer. b) larger arrays of 128×128 increase these error distributions such that analogue output results can deviate more substantially from ideal integer results. All axes have common units as the elements of the associated binary weights matrix (which contains 0s and 1s). a) and b) do not scale proportionally to the matrix sizes because the inputs contain both positive and negative values, which partly balance each other.

cooling fans (not included in the PUE) needs to be accounted for as well, resulting in a total overhead of $2\times$. In contrast to the $20\,\mathrm{W/core}$ value for the CPUs, we have not explicitly incorporated the power consumption of the non-CPU compute infrastructure (i.e., network, storage, ...). Whereas this might result in a comparative overestimate of the power consumption of the CPU ($< 50\%$), this choice is justified given that the $N = 60$ benchmark task used in Table 1 is sufficiently small to run on a single accelerator-instance for all the technology entrees. Similar to what has been done for the AI-based matrix-vector product applications based on memristor-crossbars[34,35], in future work, it can be addressed which multi-tile architecture would result in an optimal power consumption for the mem-HNN system for large-scale problems.

Finally, in Table 1, annealing time T_{ann} was defined as the time required to update the entire output vector over 50 subsequent cycles. The time to solution was estimated as the product of annealing time and the number of repetitions (n_{rep}) of updating the entire output vector required to achieve 99% success probability. n_{rep} was calculated by solving the equation $0.99 = 1-(1-p_{sr})^{n_{rep}}$, where p_{sr} is the success probability for 50 cycles of annealing. $n_{rep} = 11$ for $N = 60$ and 50 cycles of annealing. Therefore, annealing time for parallel updates was 11 times smaller than the annealing time for sequential updates, whereas the energy consumption *per clock cycle* for a parallel run is 11 times higher. Consequently, the overall energy-to-solution number mentioned in Table 1 is identical for both scenarios.

8 Matrix-vector multiplication errors due to crossbar noise

Fig. S4a-b show more details of the simulation results used to support Fig. 5c. We performed simulations of a vast ensemble of analogue computations performed in memristor crossbar arrays of size 32×32 (Fig. 5a) and 128×128 (Fig. 5b). Each The analogue computations in the 32×32 array give results very close to ideal results in such a way that there is no overlap of neighboring distributions. In other words, the error distribution (see inset figure) has a standard deviation that is sufficiently below 0.5 that any result can be rounded to the nearest integer and will match the exact integer result. In contrast, 128×128 arrays have large enough errors that simple rounding will not match the exact results, the error standard deviation (inset) can be much greater than 0.5.

9 Additional details on the experimental results

9.1 Programming of the weights matrix

The memristor crossbar array used to program the weights of the target problem comprised of nanoscale (sub-100 nm) crossbar memristors (Fig. S5) made using tantalum oxide as the active material, thereby operating via the oxygen-vacancy migration mechanism. Since the weights matrix for the Max-Cut problem contains 0's and -1's, the weights matrix programmed into the memristor crossbar consisted of either low ('0') or high ('1') conductances, and the resulting current from the vector-matrix multiplications were flipped in sign to account for the negative sign on the high conductance states. Within a 128×64 array of

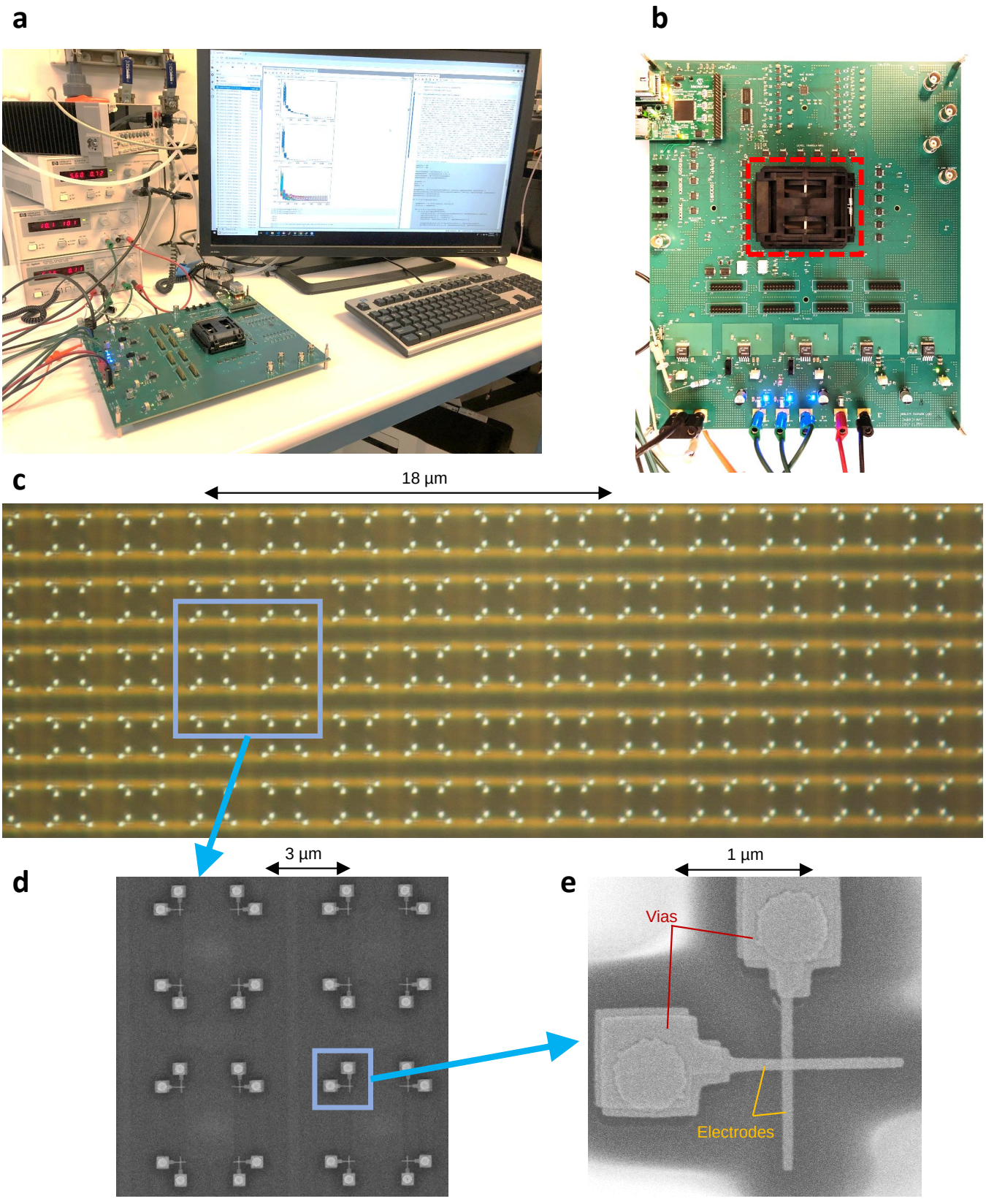

Figure S5. Images of the experimental setup, a) Photograph of the experimental setup. b) Photograph of the control board with peripheral discrete electronics and the highlighted (red dashed box) central package containing a wire-bonded memristor array. c) Optical micrograph of the chip containing the memristor crossbar array used for in-memory computing operations, d) scanning electron micrograph of a part of the array, and e) scanning electron micrograph of one device within the array. Electrodes connecting multiple devices and the access transisotors are beneath the surface, illustrated in Fig. 3d.

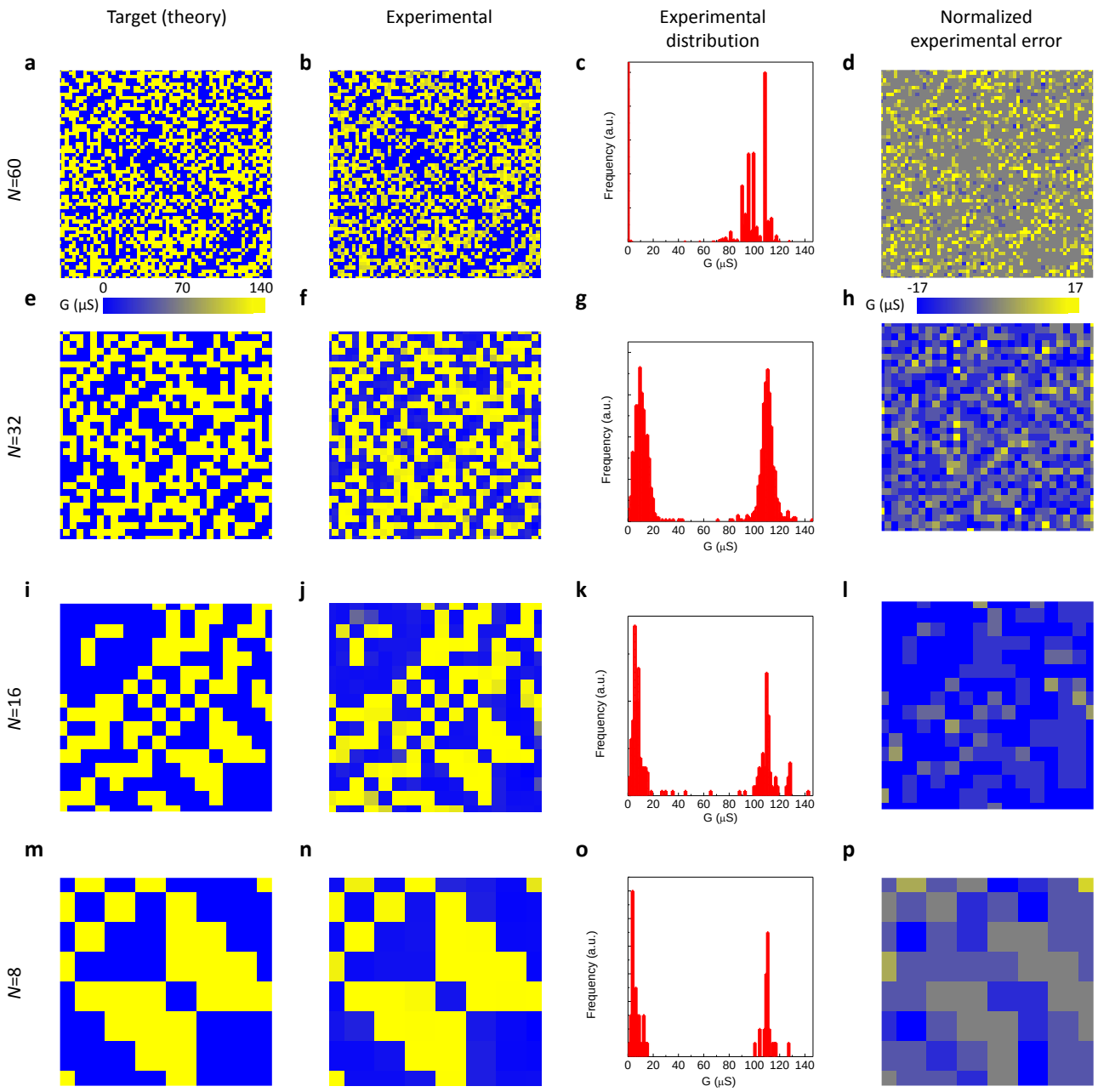

Figure S6. Experimental programming of problems of different sizes. a) Target conductance matrix, b) Experimentally implemented conductance matrix, distribution of conductances in the experimental implementation, and d) Normalized errors in the experimental conductance matrix relative to the target (normalized to 1), all for $N = 60$. e)-p) show corresponding data for $N = 32, 16$ and 8.

memristors, we programmed a chosen set of devices to represent the weights matrix by setting the target conductance for the ON or '1' state to be 1.1×10^{-4} S and for the OFF or '0' state to be 0.5×10^{-5} S. The target weights matrices, the programmed weights matrices, and the corresponding distribution of the conductances are shown in Fig. S6. Additional details of fabrication, programming and operation of the array are given elsewhere.[36] The layout of the hardware functions distributed across the memristor chip, external circuit boards and the control computer are provided in Fig. S7a. Following the programming of the weights matrix, in order to calculate the errors in the computations, we supplied hundreds of randomized binary input vectors and obtained the outputs corresponding to them. We obtained the differences between the experimentally obtained outputs and a set of outputs calculated by precise matrix multiplications using the same set of inputs as the experiments and the measured weights matrix programmed onto the memristor chip (Figs. S7b-d). This allowed for an estimation of the intrinsic noise of the system, which was used as a constant noise magnitude in our circuit simulations. We performed both experiments and experimentally-grounded circuit simulations to determine the optimal schedule of varying w during calculations across several cycles (Figs. S7e-g). Both simulations and experiments for different problem sizes confirmed that an optimal schedule was to sweep w from a negative to a positive value, which would mimic the process of simulated annealing. A read voltage of 0.2 V was used. Therefore the input voltage vector was determined by scaling the input vector by a factor of 0.2. The weights matrix was a binary matrix with the high conductance (1.1×10^{-4} S) representing '1'. The output currents were therefore divided by the high conductance value and the voltage scaling factor (0.2) to obtain the output vector.

9.2 Intrinsic noise improves the performance in a hysteretic threshold scheme

While Fig. 4 detailed experimental data demonstrating the presence of intrinsic noise and how it was being harnessed, we further evidence harnessing of intrinsic noise in Fig. S8. We performed circuit simulations of the mem-HNN similar to those in Fig. 4g, but without including the intrinsic noise estimated from the error distributions (e.g.: Fig. 4d). The results show clearly worse performance without the inclusion of intrinsic noise, and also a disagreement with experimental data in general. This reasserts the presence and utility of intrinsic noise by employing the hysteretic threshold. Indeed, the noise is not just 'present' in our experimental implementation, it is critical to reach superior performance.

9.3 Comparison of simulated annealing and hysteretic threshold

We compared the experiments based on the hysteretic threshold described in the main text with circuit simulations both using a hysteretic threshold model and using simulated annealing. In simulated annealing, we use a pseudo-random number generation circuit (as a part of the digital control computer) to externally inject controlled noise into the mem-HNN. In order to statistically compare simulated annealing and the utility of the hysteretic threshold, we simulate solutions to the problems shown in Fig. 4g using optimized simulated annealing schemes for each of the problems, which appear to compare very well with the experimental and simulated data that use the hysteretic threshold (Fig. S9). This reasserts the fidelity of our circuit simulations, especially in projecting performance metrics at larger scales, and the equivalence of the hysteretic threshold scheme and simulated annealing scheme. Further, in Fig. S10 we compare our experimental data using the hysteretic threshold on the problem of size $N = 60$ with ideal calculations using simulated annealing (without accounting for circuit nonidealities). It is apparent that (1) both techniques are able to achieve global optimization by escaping local energy minima, and (2) the experiments are as good as ideal calculations. In short, the idea of the hysteretic threshold performs at least as good as the well-known meta-heuristic of simulated annealing.

9.4 Experimental study of analogue crossbar and dynamical noise and interaction with hysteretic threshold

To analyse the effects of the combination of dynamic noise and cycle-to-cycle analogue circuit noise with the influence of the hysteretic threshold, we measured time traces based on the matrix-vector multiplication output of the crossbar array for fixed input vectors. To have a broad representative test set, we used 200 random bipolar trial input vectors. For each trial input, we performed matrix-vector multiplications with a programmed 60×60 Hopfield matrix for 250 iterations. We compared the output in this experiment with the ideal output obtained by performing the vector-matrix multiplication using the target Hopfield matrix. This data set was used to obtain the histograms shown in Fig. S11a-c. These histograms are obtained by calculating the deviation between the experimental output and the target output. In Fig. S11c, we sample over all trial vectors, all columns, and all time slices. In Fig. S11a, we show the deviation obtained by first calculating the time-averaged experimental output vectors, for all the trial outputs and subsequently sampling at all the columns and trial vectors. For a given input vector, the deviation of these time-averaged output vectors compared to the input reveals the errors in the vector-matrix multiplication performed by the memristor crossbar array, due to the various nonlinear effects described around Fig. 1. In Fig S11b, we then calculated for all time traces the deviation between the output at the current time-step and the corresponding time-averaged output vector, and sampled these values for all time slices, over all columns. Also, approximately, $\sigma^2_{\text{tot}} = \sigma^2_{\text{cir}} + \sigma^2_{\text{dyn}}$, which demonstrates that the total noise from the circuit comprises the two sources we have distinguished, with the analogue crossbar noise being dominant.

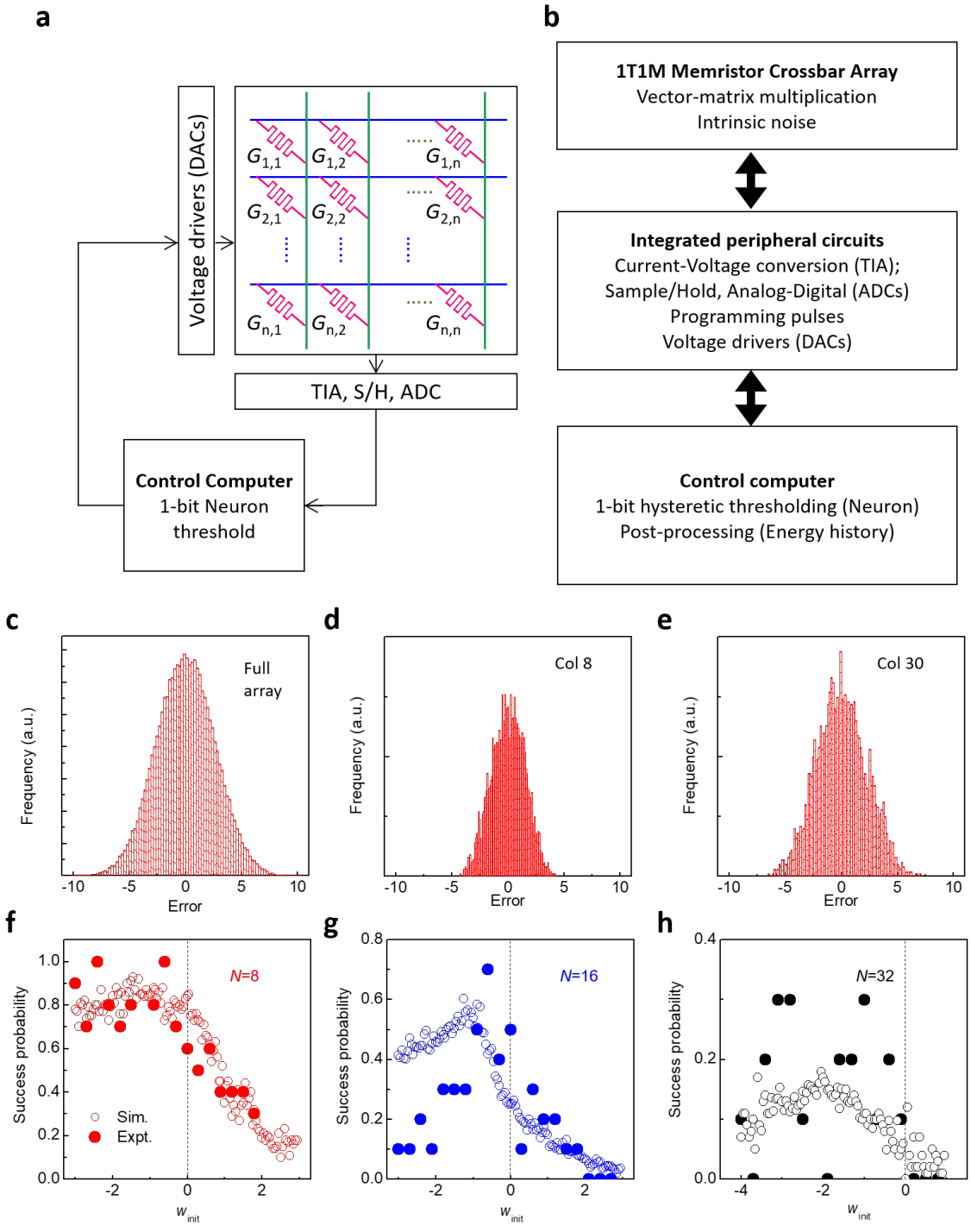

Figure S7. Experimental layout and calibration data. a) Schematic layout of the mem-HNN operating sequence, displaying the memristor crossbar array, peripheral components outside of the array, and control computer. Additional details, including operation and peripheral component specs are provided elsewhere.[36] b) Schematic layout of the hardware functions in the implementation of the mem-HNN. c)-e) Histograms of the errors in the calculations obtained from the experimental system (for N=32), for the full array and two randomly chosen columns of the array. f)-h) Success probability against initial w (for a fixed final positive w) shows that the highest success probability is achieved for a negative initial w for all three problem sizes ($N = 8, 16, 32$).

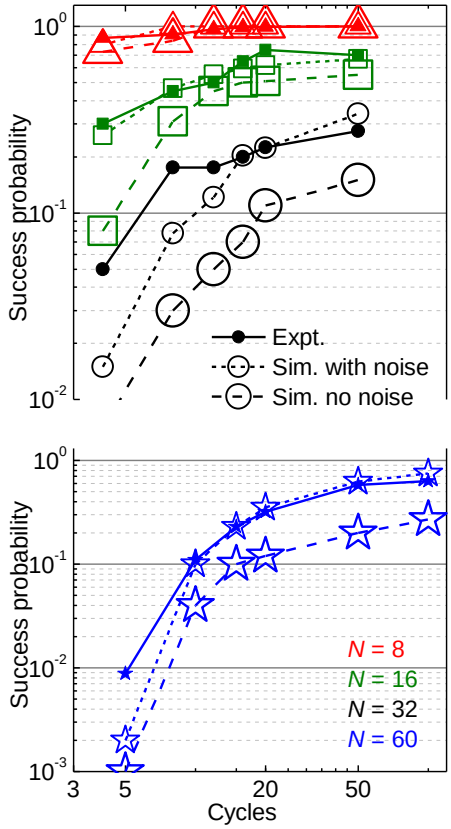

Figure S8. Additional data showing that the hysteretic threshold made use of intrinsic hardware noise. Data in Fig. 4g overlaid with similar simulated data except that the intrinsic noise was not included ('Sim. no noise'). The results show a clearly worse performance (lower success probabilities) and disagreement with experimental data when intrinsic noise is not included.

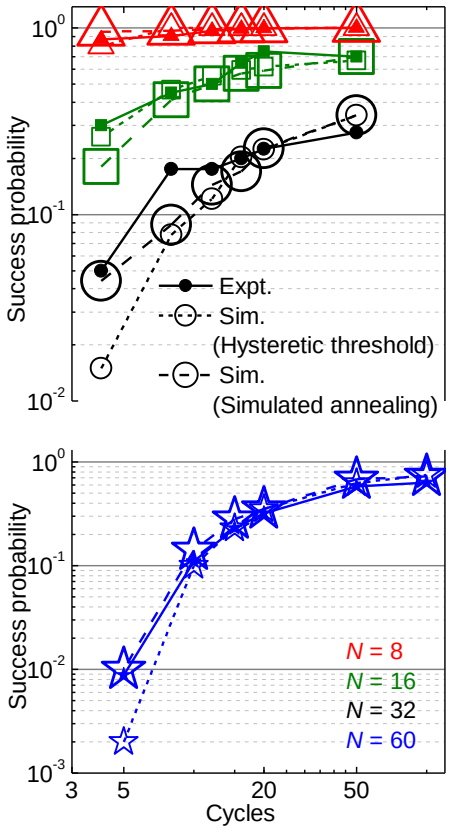

Figure S9. Comparison of the hysteretic threshold to simulated annealing. Data in Fig. 4g overlaid with simulated data similar to that in Fig. 4g, except that in this case we added simulations to solve the same problems using simulated annealing in addition to simulations in which the hysteretic threshold scheme was used.

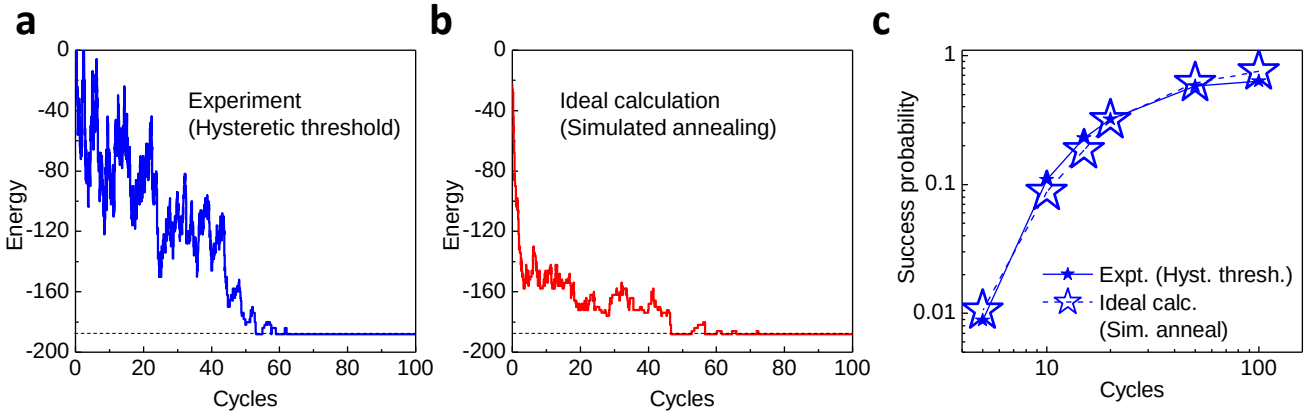

Figure S10. Comparison of experiments (hysteretic threshold) to ideal calculations (simulated annealing) a) Example of energy minimization for solving a problem of size $N = 60$, showing experimental data, which used a hysteretic threshold. b) An example of energy minimization for solving the same problem using ideal calculations (no circuit non-idealities) and employing simulated annealing (via a pseudo-random number generator). Dashed horizontal line in a) and b) indicate the energy representing the optimal solution. c) Success probabilities for the cases in a) and b) as a function of cycles. Experimental data using hysteretic threshold is as good as the best ideal Hopfield network calculations.

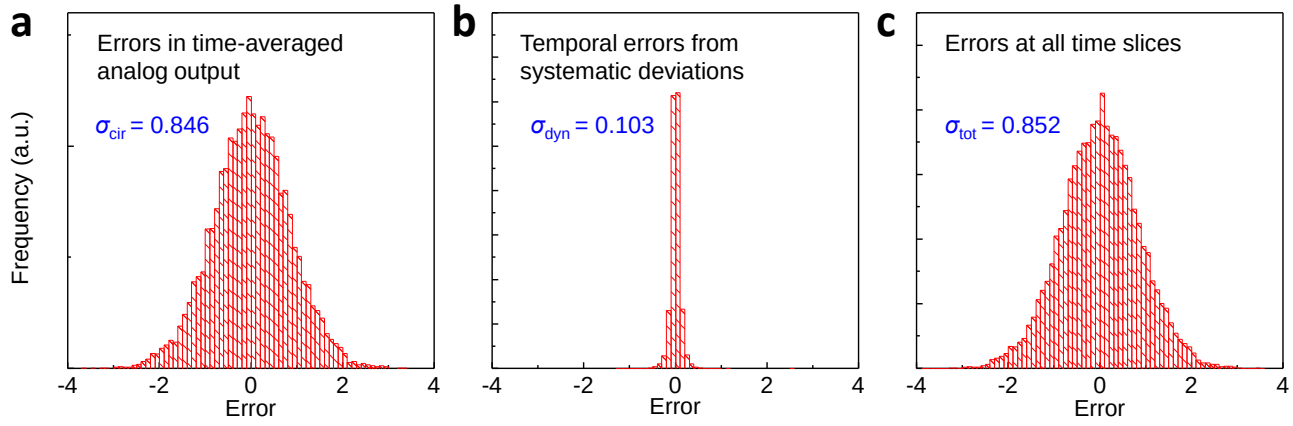

Figure S11. Histograms of noise. a) Histogram of analogue circuit noise (with standard deviation σ_{cir}). b) Histogram of dynamical noise (with standard deviation σ_{dyn}) c) Histogram of total noise (with standard deviation σ_{tot}).

Next, the experimental outputs of the vector-matrix multiplications described in the previous paragraph are input to the neuron update (Eqn. 1), using the hysteretic thresholding for each Hopfield node as in Eqn.3. In Fig. S12, to showcase the effect of dynamic noise for different hysteretic threshold widths, we show a node input fluctuating close to the nominal zero value (the same time trace of Fig. 1d). The Neuron output for the case of $2w = 0$ shows the case of an unmodified Hopfield network and the noisy input leads to fluctuating neuron states. Modulating the hysteretic threshold width to positive values suppresses these fluctuations, while moving to more negative hysteretic threshold widths amplifies them, as shown.

To characterize the analog crossbar noise, we experimentally performed thousands of steps of regular mem-HNN operations for different variants of w. We compared the experimental data (neuron input and output) to ideal calculations to obtain the deviations. An example time trace has been displayed in Fig. 1b , whereas we display a more comprehensive data set in Fig. S13. It is clear that a highly negative w indeed amplifies the intrinsic noise, while a positive w suppresses intrinsic noise. The case for $2w : -4 \rightarrow 1$, shows the operational mode similar to simulated annealing, where an initial amplification of noise is followed by a suppression of noise to stabilize the final convergence. The Fourier transforms of the input deviations (Fig. S13 are shown in the middle column, and display a white-noise-like distribution across the entire spectrum. These experimental results are as expected, as the analogue crossbar noise is fundamental to the computing operations performed in the crossbars themselves. The flat noise spectra shows an injection of noise at all frequencies of the mem-HNN operation, even up to the highest update frequency.

In Fig. S14, we generalize the effects of the hysteretic threshold width compared to the case study discussed in Fig. S12. Using the full dataset of Figs. S11 and S14, we calculated output time traces of the Hopfield node for the following different scenarios: (a) the ideal output assuming a Hopfield node with $w = 0$, and using the ideal target matrix, (b) the ideal output obtained by using the target matrix while using different w, (c) the output obtained for various w starting from the experimental time traces. For the time traces with non-zero w, we assumed identical initial conditions. Based on those four different timetraces, we then calculated the deviations plotted as a function of w for two cases: (i) Experimental vs. software with hysteretic threshold (i.e., scenario (c) vs (b)), and (ii) Experimental vs. software without hysteretic threshold (i.e., scenario (c) vs (a)). The y-axis counts the number of times the time traces are unequal and is normalized such that a deviation equal to one represents a deviation at every time step for all columns and all trials. Unsurprisingly, line (ii) approaches 0.5 for very negative w, which can be understood as the output without using a hysteretic threshold (i.e., time trace (a)) is either +1 or -1, whereas the signal with large negative w periodically oscillates between +1 and -1 (time trace (c)). In contrast, both deviation curves converge to zero for a broad range of positive w. The interplay of the errors due to analogue crossbar noise with the negative w as a controllable deviation source (functioning as 'computational noise' in the mem-HNN algorithm) can be seen in the behavior of curves (i) and (ii). In a finite region between $2w \approx -20...1$, the output of the Hopfield neurons has a tunable level of random deviations enabled by the experimental vector-matrix multiplications performed in analogue crossbar arrays. For larger values, the deviations go to zero. This controllable enhancement or suppression of the deviations is a core computational resource utilized in our proposed mem-HNN. It is worth noting that this is what enhances the success probability in the simulations with noise in Fig. S8 versus the no noise case.

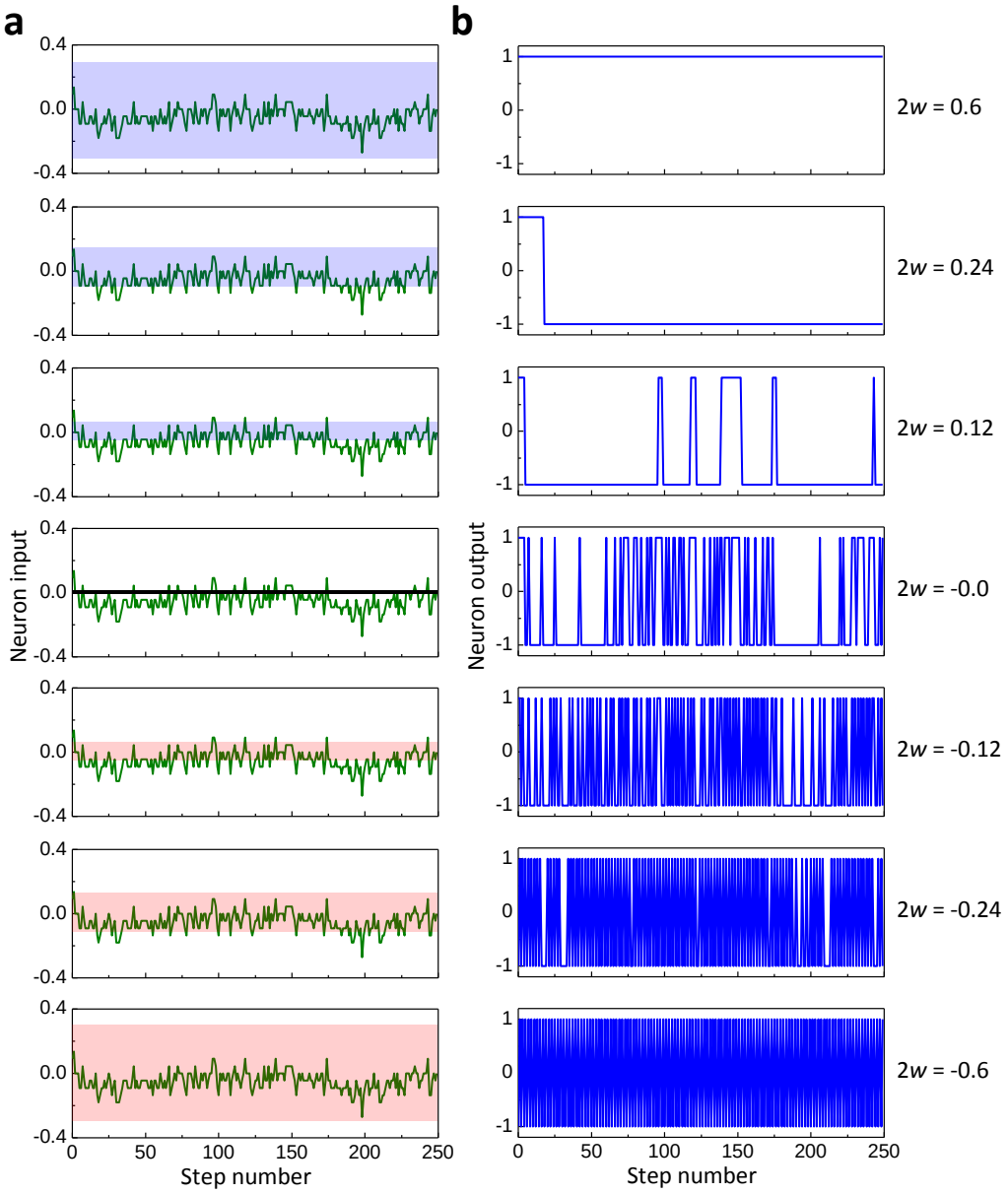

Figure S12. Dynamical noise and the interaction with a varying hysteretic threshold. a) Neuron input obtained via experimental measurements for the same input repeatedly applied to the memristor crossbar array. Hysteretic threshold widths ($2w$) are marked as bands over the data, with red bands representing $w < 0$ and blue bands representing $w > 0$. b) Neuron outputs calculated from experimental data for different hysteretic threshold widths $2w$, as annotated. The amplification of noise at negative w is apparent.

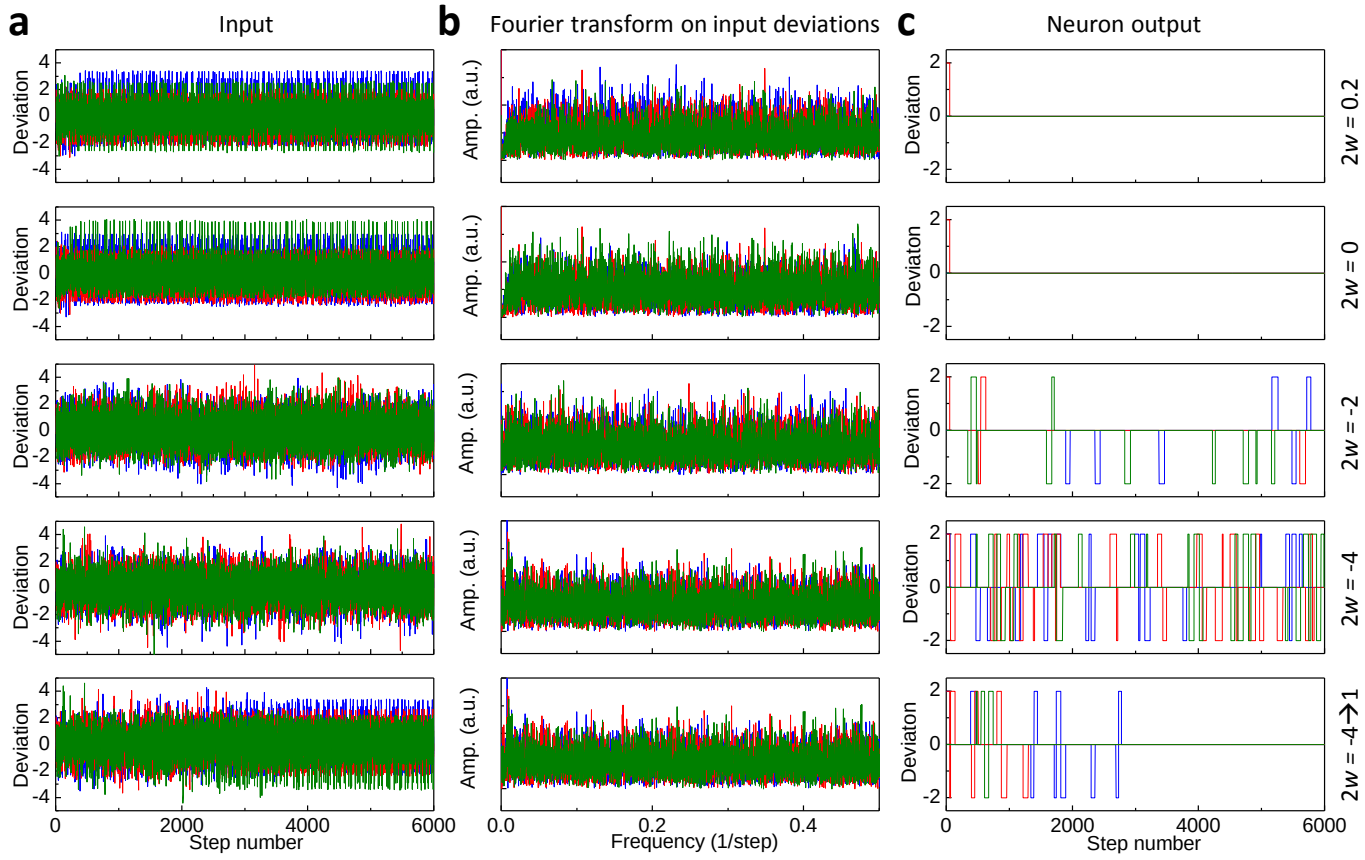

Figure S13. Analogue circuit noise a) Deviations in neuron input obtained via experimental measurements compared to ideal calculations during operation of a mem-HNN for three different trials (corresponding to three different initial conditions). b) Fourier transform of the neuron input in a), showing noise that is distributed uniformly across the spectrum. c) Deviations in neuron output obtained via experimental measurements compared to ideal calculations during operation of a mem-HNN. For the case of $2w : -4 \rightarrow 1$, the initial amplification of noise followed by a suppression of noise is apparent, while the input noise persists throughout the measurement.

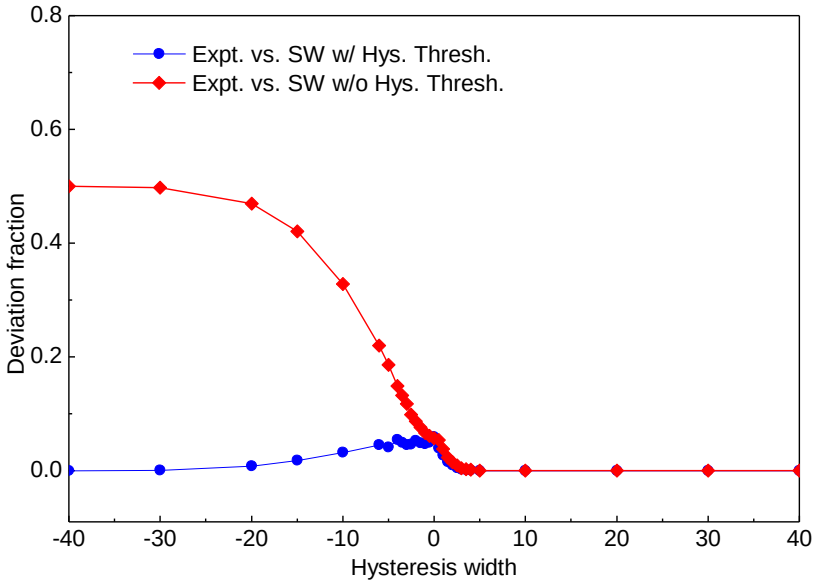

Figure S14. Deviations due to the hysteretic threshold. Deviations plotted against hysteresis width $2w$ for two cases: (i) Experimental vs. software with hysteretic threshold, and (ii) Experimental vs. software without hysteretic threshold.

9.5 Alternative sources of intrinsic noise

Our study focused on one source of intrinsic noise, namely noise from the analogue memristor array, complemented by a technique that enables generation and modulation of noise. However, in other implementations of the mem-HNN system, alternative sources of noise in analogue systems can be leveraged, many of which can be dynamically tuned to implement simulated annealing. To appreciate this insight and as extended material for the interested reader, we list three sources of noise for use in recurrent and other forms of neural networks.

Noise may originate from nonvolatile memristors used to store information. It is known that tantalum oxide memristors exhibit amplified random telegraph noise when biased or held at certain conductances[37]. A dedicated noise-generating row of memristors could be included in an array of memristors storing the weights matrix, and such memristors can be programmed at the start of every cycle to exhibit a certain degree of noise. This would allow simulated annealing since a controlled source of noise is added onto the currents at every row before they encounter a nonlinear filter. Noise may also originate from volatile memristors, such as controlled chaotic oscillations in nanoscale niobium oxide memristors[38]. Volatile memristors, especially those that exhibit threshold switching can form compact comparators that can provide the nonlinear filtering within a recurrent neural network. Thus by utilizing compact devices to produce nonlinear filtering, without any significant circuit overheads, it may be possible to achieve simulated annealing by controlling the magnitude of chaos injected at the filtering function. Analogue noise from the memristor array was discussed in detail in this study, which may originate form multiple sources including wire resistances, imperfect conductance programming, device-level fluctuations, etc. These may be static, such as finite conductance distributions, or these may be dynamic, such as the noise distributions arising from device- or circuit-level fluctuations. Both static and dynamic nature of such noise may have distinct behaviors when utilized for simulated annealing, which deserves further study of its own (cfr. e.g., Ref.[39] for a discussion on the consequences of static noise). Similar to the hysteretic threshold utilized in this study, there may be a variety of techniques to harness and modulate this noise.

10 Additional context for the comparison of the different accelerators

This section provides additional information about Table 1 that could not be included in the main text of the article due to space constraints.

10.1 Emerging technologies not included in Table 1

Interestingly, recently, another promising digital annealing technology, has been proposed, namely an FPGA implementation of an optimized Boltzmann sampling algorithm called a Digital Annealer[24]. Impressively, the DA has been applied to Max-Cut with non-binary weights with a $16-$bit resolution, even outperforming software algorithms that have a higher bit resolution[24].

We expect similar benefits for the mem-HNN system. Moreover, by embedding the crossbars inside the mem-HNN in an appropriate tiled architecture, the effective bit-resolution can be increased in future work[34]. Unfortunately, so far, the performance of the DA for the dense Max-Cut on $N = 60$ hasn't been reported and, as it behaves similar in power and speed to the CPU and GPU approach, it is not included in the benchmark table in the main text of the article. Similarly, the recent work on a CMOS annealer reported in[18,19] has not investigated its performance for the dense Max-Cut benchmark task and, as it only allows for sparse connectivity (hence bad scaling of the success probability and time-to-solution), we chose to not include it in the benchmark table due to place constraints.

10.2 Caveat about the power consumption numbers for quantum and CIM systems

In the case of D-wave's quantum annealing accelerator, 25 kW of power is needed to operate the system, of which the majority is used for cooling. This power envelope is expected to stay constant when the number of nodes scales into the tens of thousands, as the cooling is currently over-provisioned[40]. The idea is that, for some applications, a limited set of qubits might have a computational performance that can not be obtained with a traditional high performance computing (HPC) system, even though HPC systems use $100-1000\times$ more power. Due to this large fixed offset, when estimating the energy efficiency of quantum annealing based on the performance of current available small system sizes, the potential efficiency will be underestimated. Similarly, the current CIM implementations are just proof of concept implementations and are hence not yet optimized for energy efficiency.

10.3 Benchmark simulations on a CPU using parallel tempering

Previous research has demonstrated that running parallel tempering (PT) on an Intel Xeon CPU E5-1650 v2 (3.50GHz) as a digital benchmark for the dense Max-Cut problem, reaches comparable time-to-solution values (TTS) as the fiber-based CIM proposed by Stanford and NTT[13]. Even though the power budget (thermal design power 130 W, actual estimated power 20 W) is lower than the power budget of the GPU (250 W), due to the stronger reduction in time-to-solution when using a GPU, running the noisy mean field annealing algorithm proposed by D-wave on a GPU[12] is outperforming the parallel tempering approach when run on a CPU.

The Max-Cut problem instances used in this work, are either taken from the Biq-Mac Library $N = 60, 80, 100$, or randomly generated using the same problem definition as the dense Max-Cut problem proposed in the CIM analysis[13]. As an additional check on the difficulty of the specific Max-Cut instances used in Fig. 6, we reran these exact same problem instances on a CPU, and compared those with the previously published time-to-solution numbers for a CPU obtained[13] in the CPU-CIM comparison (Fig. S15). This CPU reference performance is comparable with the performance of NTT's parallelized version of the fiber-based CIM.

In the experiment for this paper, an Intel(R) Core(TM) i7-3720QM CPU @ 2.60GHz was used. Similar to the CIM-paper, all the CPU results included here are provided by Salvatore Mandrà, using the same implementation of parallel tempering in the NASA/TAMU Unified Framework for Optimization (UFO). The algorithm (PT@UFO) has been executed on a single core. Despite the lower clock speed, the run-time analysis is still consistent with the previous analysis published in the NTT/Stanford paper.

In Fig. S15, besides the reference data used in the Stanford/NTT paper, we include two different curves for the set of simulations ran on our own set of Max-Cut-problems: run-time without initialization time and run-time including the initialization time. Roughly speaking, the initialization time consists in the allocation of the arrays and the generation of all the random numbers that will be used in the simulation (all random numbers are cached and stored in memory). Hence, the run-time corresponds to the sole optimization of the Max-Cut problem. Typically, for larger systems, this initialization time is so fast that it is negligible compared to the run-time. However, for the rather small instances ($N < 100$) discussed in this paper, this is not the case. Importantly, random number generation happens intrinsically in the proposed hardware for the mem-HNN system, the CIM system or the quantum system, which is one of the advantages of using analogue hardware in accelerators. However, as the percentage of time spent initializing becomes negligible for larger systems, we report the TTS value *without* initialization for the CPU in Table 1. No description of initialization-time has been provided in the discussion of the GPU results[12].

A reasonable number for a single core energy consumption[32,41] is 20 W/core, this is the value used in Table 1. Additionally, the TTS for for the CPU benchmark can be improved by running independent processes in parallel, i.e., the TTS for PT@UFO can be reduced by the optimal number of repititions $n_{rep} = TTS/T_{\text{ann}}$ by running annealing instances on n_{rep} CPU cores (for the small $N = 60$ task, the median $n_{rep} = 1$, so parallelization is not beneficial yet). Importantly, if the overhead of parallelization is negligible, this should have no significant effect on the energy to solution value listed in Table 1.

The time-to-solution in Table 1 for PT@UFO is calculated by running 1000 simulations of PT, with random seeds. For each simulation, the algorithm stops after the required number of sweeps to obtain the optimal solution, which we call the 'run-time'. For a given problem size N and problem instance i, due to the different random seeds, every run-time is different, and by calculating the cumulative distribution of all the obtained run-times, combined with a bootstrapping procedure, the optimal

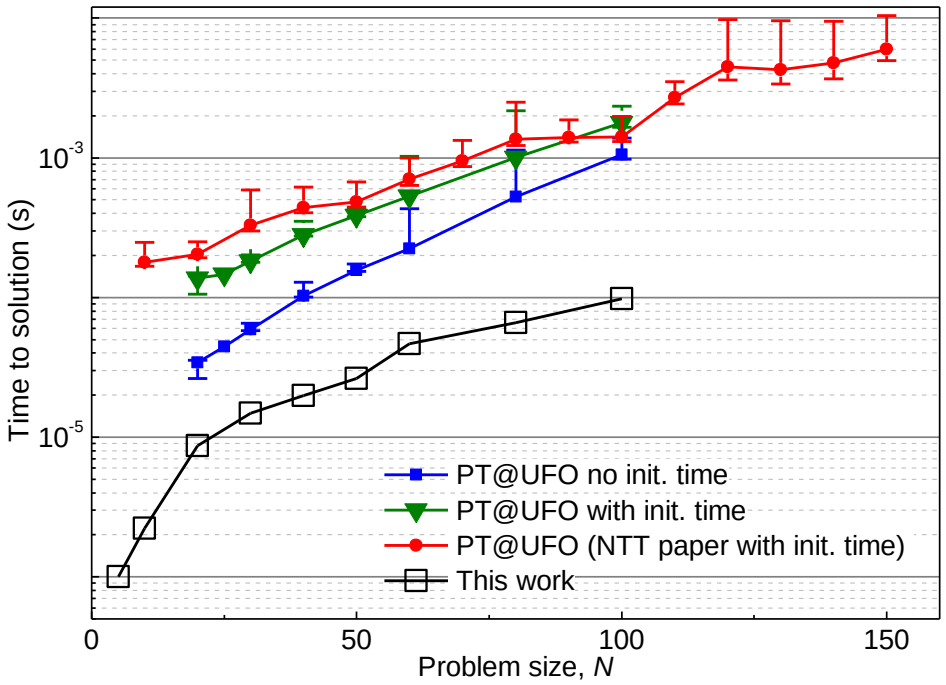

Figure S15. Comparison of mem-HNN to a CPU. Time-to-solution (TTS) benchmark results for dense Max-Cut obtained by running parallel tempering on a CPU: (red circle marker) reference data obtained from the Stanford/NTT-paper[13], using the same problem definition, but different instances as in this paper; (triangle marker) CPU run-time on the same problem instances as used in the mem-HNN simulations, including initialization time; (square marker) CPU run-time on our problem set, without including the initialization time. Black square markers are our results from Fig. 6b, for 1000 cycles, using sequential runs (i.e., the worst case scenario shown in Fig. 6b).

runtime $RT_{N,i}$ ($= T_{annealing}$) can be estimated that minimizes the time-to-solution $TTS_{N,i}$ for that task. The time-to-solution TTS_N for a given size N can then be obtained by using the same median value of $RT_N = \text{median}(RT_{N,i})$ for all the instances and by subsequently calculating for every task i (with size N) the success probability $p_{sr,i}$ that a simulation for this task can be been completed in less time than RT_N. Finally, these success probabilities $p_{sr,i}$ can be used to calculate for every size the median of the corresponding TTS-estimates (based on the number of repetitions $n_{rep,i} = \log(1-0.99)/\log(1-p_{sr,i})$) for all these instances. Specifically, for $N = 60$, $n_{rep} = 1$, consequently, as reflected in Table 1, $TTS_{60} = RT_{60} = T_{\text{ann}}$.

10.4 Power consumption estimate for the GPU data

In Table 1, when including the energy efficiency of the GPU, we are using the 250 W as an upper bound for the thermal power consumption. Similar to the CPU scenario, where the thermal design power is 130 W, but in practice the power consumption during operation can be assumed to be[32,41] $\sim 20\,\text{W}/\text{core}$, we assume the effective power of the GPU can be a factor $10\times$ lower than the power budget estimate.

10.5 Run-time analysis

Although the mem-HNN, the CIM and NMFA@GPU have a similar clock frequency (in the case of the CIM we pick the repetition rate of the pulsed laser as fundamental clock unit), and a similar time-to-solution scaling, there are still important differences in performance. Before we dive into the speed and energy efficiency specifics, it is worth noting the differences in modus operandi: the algorithm on the GPU can update the nodes synchronously, whereas the CIM system is purely asynchronous (one node gets updated per pump pulse), and, as explained in the discussion of Fig. 6, the mem-HNN is somewhere in-between: we notice better performance in success probability if we update the nodes in batches $N_b = 10$ instead of node-by-node or completely synchronous. For the CIM system, doing updates in batches would be possible by using multiple phase-sensitive amplifiers in parallel, but that would be a drastic increase in system complexity. Given that traditional discrete-time Hopfield algorithms are known to have better performance when using asynchronous updates, the ability for the noisy mean-field algorithm to work with in essence synchronous updates can be surprising. However, the asynchronous updates

were a mathematical trick to mimic noisy continuous-time Hopfield neural networks, where updates do occur 'synchronously'. The noisy mean field algorithm can be seen as another way of modeling noise-induced annealing effects in a discrete-time algorithm that allows for synchronous updates. These choices do have important consequences on the eventual time-to-solution. One annealing run will consist of a certain number of cycles, and, for the mem-HNN, one cycle will take $\frac{N}{N_b}T_{clock}$. For the CIM, $N_b = 1$ and there is an additional overhead on the order of $1.6 - 2.5$ related to additional pulses required for stabilizing the fiber feedback loop, increasing the annealing time. For the GPU the calculation time also dwells on the calculation of the matrix-vector products in the algorithm. In the DA implementation, only one node is updated per iteration, but due to its parallel-trial scheme the number of required cycles per run is reduced. Finally, despite the slightly higher clock cycle of the CPU, the parallel-tempering algorithm has a longer optimal annealing time than both the GPU-approach and the mem-HNN approach. This is probably related to its asynchronous update mechanism of the states of the Ising nodes.

As discussed in Sec. SM.10.3, there is a procedure to obtain the optimal run-time/annealing time for the parallel tempering algorithm on the CPU. Similarly, in principle, the number of cycles per annealing run can be optimized for different problem sizes for the other classical technology implementations as well. However, the number of cycles used to obtain Fig. S15 and Table 1 is fixed: 1000 for the GPU and CIM, 50 for mem-HNN (based on Fig 6b, resulting in a 20x difference). For a given annealing time, one run will then result in a success probability, and this success probability can be converted to a time-to-solution (i.e., the duration required to hit the optimal solution with 99% probability). Given the same clock frequency for GPU, CIM and mem-HNN, the shorter annealing time of the mem-HNN compared to the CIM can be solely explained by the stabilization pulse overhead for the CIM, the choice to use $20\times$ less cycles per run for the mem-HNN, and the CIM's current inability to update in batches. The GPU's scaling of the annealing time versus problem size is less straight forward to explain, as it also scales with the (dense) matrix-vector multiplication, hence, for large problem sizes, we expect a quadratic dependence (if the available CUDA-cores is assumed to be fixed).

10.6 Run-time versus success probability trade-off

The meta-parameters of the algorithm in the case of the noisy mean field annealing implementation, and the physics and operation regime of the hardware accelerators will then determine the success probability for a given annealing time. For instance, the success probability for the mem-HNN is lower than the success probability of the CIM, which results in the need to have more runs for the mem-HNN. However, due to the shorter annealing time, the absolute time-to-solution is still better. Obviously, a similar optimization routine based on the trade-off between allowing for a lower success probability to obtain an overall shorter time-to-solution could also be performed on the CIM system. Importantly, this wouldn't make up for the overhead time required for stabilization of the feedback loop and the current lack of possibilities to use batches in that technology. Also, whereas the mem-HNN benchmark numbers are just based on (experimentally grounded) simulation, the CIM is an actual experimental implementation. In these experiments, there is also overhead for readout and postprocessing of data[13], which is currently not included in the table.

10.7 Technology nodes for the electronic implementations

We assumed a 32 nm node for the simulations of the mem-HNN, while the CPU and GPU experiments used a 22 nm and 16 nm nodes, respectively. Using scaling protocols derived from leading foundry data, we estimated the performance metrics for a 16 nm technology node design, which is reported in Table 1.

10.8 Outlook for optical and quantum technologies

Finally, we want to give an outlook on the future prospects of the involved technologies.

and/or by allowing for batch updates using several parallel phase-sensitive amplifiers. One other promising alternative is to integrate the coherent Ising machine into a photonic integrated circuit[14,42,43]. This would result in relatively lower cost, potential for scalability, lower energy consumption and higher clock speeds.

As for potential improvements in quantum annealing, demonstrating larger node sizes will result in an increased energy efficiency, but without changes in the interconnectivity, we do not expect this technology to become competitive with the other listed approaches for the currently studied benchmark task. Consequently, as mentioned in Ref.[13], there are also efforts on allowing for all-to-all connectivity in quantum annealing implementations, which would result in similar scaling as the other classical approaches, but this technology development is non-trivial[17,44]. Besides increasing the number of qubits in a single D-wave system, to drastically improve the energy-efficiency of the quantum annealer far more efficient cryogenic cooling techniques need to be developed, but it is currently unclear which techniques might be suitable for this task[32].

References

1. Kochenberger, G. *et al.* The unconstrained binary quadratic programming problem : a survey. *J Comb Optim* **28**, 58–81 (2014).

2. MIPLIB 2017 (2018). Http://miplib.zib.de.

3. Lucas, A. Ising formulations of many np problems. *Frontiers in Physics* **2**, 5 (2014).

4. Wu, L.-Y., Zhang, X.-S. & Zhang, J.-L. Application of discrete hopfield-type neural network for max-cut problem. In *Proceedings of ICONIP*, 1439–44 (2001).

5. Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences* **79**, 2554–2558 (1982).

6. Rojas, R. *Neural networks: a systematic introduction* (Springer, 1996).

7. Joya, G., Atencia, M. A. & Sandoval, F. Hopfield neural networks for optimization : study of the different dynamics. *Neurocomputing* **43**, 219–237 (2002).

8. Atencia, M., Joya, G. & Sandoval, F. Two or three things that we (intend to) know about Hopfield and Tank networks. In *ESANN*, April, 27–29 (2005).

9. Schonfeld, D. & Friedman, G. Enhancement of Hopfield Neural Networks Using Stochatic Noise Processes. *Proceedings of the 2001 IEEE Signal Processing Society Workshop* 173–182 (2001).

10. Hu, S., Liao, X. & Mao, X. *Journal of Physics A: Mathematical and General* .

11. Ackley, D. H., Hinton, G. E. & Sejnowski, T. J. A learning algorithm for boltzmann machines. *Cognitive Science* **9**, 147 – 169 (1985). URL http://www.sciencedirect.com/science/article/pii/S0364021385800124.

12. King, A. D., Bernoudy, W., King, J., Berkley, A. J. & Lanting, T. Emulating the coherent Ising machine with a mean-field algorithm 1–5 (2018). arXiv:1806.08422v1.

13. Hamerly, R. *et al.* Experimental investigation of performance differences between coherent ising machines and a quantum annealer. *arXiv preprint arXiv:1805.05217* (2018).

14. Roques-Carmes, C. *et al.* Heuristic recurrent algorithms for photonic Ising machines. *Nature Communications* **11** (2020). 1811.02705.

15. Goto, H. Bifurcation-based adiabatic quantum computation with a nonlinear oscillator network. *Nature Publishing Group* 1–8 (2016). URL http://dx.doi.org/10.1038/srep21686.

16. Goto, H., Lin, Z. & Nakamura, Y. Boltzmann sampling from the Ising model using quantum heating of coupled nonlinear oscillators. *Scientific Reports* **8**, 1–9 (2018). URL http://arxiv.org/abs/1707.00986. 1707.00986.

17. Yamamoto, Y. & Aihara, K. Coherent Ising machines — optical neural networks operating at the quantum limit. *npj Quantum Information* 4–6 (2017). URL http://dx.doi.org/10.1038/s41534-017-0048-9.

18. Yamaoka, M. & Yoshimura, C. 20k-spin Ising chip for combinatorial optimization problem with CMOS annealing. *2015 IEEE International Solid-State Circuits Conference - (ISSCC)* 432–434 (2015). URL http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=7063111.

19. Yamaoka, M. *et al.* A 20k-spin Ising chip to solve combinatorial optimization problems with CMOS annealing. *IEEE Journal of Solid-State Circuits* **51**, 303–309 (2016).

20. Yoshimura, C. *et al.* Uncertain behaviours of integrated circuits improve computational performance. *Scientific reports* **5**, 16213 (2015). URL http://www.nature.com/srep/2015/151120/srep16213/full/srep16213.html.

21. Mcdonnell, M. D. Is Electrical Noise Useful ? *Proceedings of the IEEE* **99** (2011).

22. Faisal, A. A., Selen, L. P. J. & Wolpert, D. M. Noise in the nervous system. *Nature Review Neuroscience* **9**, 292–303 (2009).

23. Mcdonnell, M. D. & Ward, L. M. The benefits of noise in neural systems: bridging theory and experiment. *Nature reviews neuroscience* **12**, 415–425 (2011).

24. Aramon, M. *et al.* Physics-Inspired Optimization for Quadratic Unconstrained Problems Using a Digital Annealer 1–15 (2018). arXiv:1806.08815v2.

25. Liu, W. & Wang, L. Solving the Shortest Path Routing Problem Using Noisy Hopfield Neural Networks. In *2009 WRI International Conference on Communications and Mobile Computing*, 299–302 (2009).

26. Olurotimi, O. & Das, S. Noisy Recurrent Neural Networks the discrete time case. *IEEE Transactions on Neural Networks* **9**, 937–946 (1998).

27. Ueyoshi, K., Marukame, T., Asai, T., Motomura, M. & Schmid, A. FPGA Implementation of a Scalable and Highly Parallel Architecture for Restricted Boltzmann Machines. *Circuits and Systems* 2132–2141 (2016).

28. Kim, S. K., Mcafee, L. C., Mcmahon, P. L. & Ozukotun, K. A Highly Scalable Restricted Boltzmann Machine FPGA Implementation. In *2009 International Conference on Field Programmable Logic and Applications*, 367–372 (2009).

29. Amin, H., Curtis, K. & Hayes-Gill, B. Piecewise linear approximation applied to nonlinear function of a neural network. In *IEE Proceedings - Circuits, Devices and Systems*, 313–317 (1997).

30. Shin, J. H., Jeong, Y., Zidan, M. A., Wang, Q. & Lu, W. D. Hardware Acceleration of Simulated Annealing of Spin Glass by RRAM Crossbar Array. In *IEEE International Electron Devices Meeting (IEDM)*, 63–66 (2018).

31. Chen, P. Y., Peng, X. & Yu, S. NeuroSim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures. *Technical Digest - International Electron Devices Meeting, IEDM* 6.1.1–6.1.4 (2018).

32. Mandra, S. & Katzgraber, H. G. A deceptive step towards quantum speedup detection. *Quantum Science and Technology* **3**, 1–11 (2018). 1711.01368.

33. Uptime Institute's 2014 Data Center Survey. Tech. Rep. (2018).

34. Shafiee, A., Nag, A., Muralimanohar, N. & Balasubramonian, R. ISAAC : A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. *ACM SIGARCH Computer Architecture News* (2016).

35. Ankit, A. *et al.* Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 715–731 (ACM, 2019).

36. Hu, M. *et al.* Memristor-Based Analog Computation and Neural Network Classification with a Dot Product Engine. *Advanced Materials* **1705914**, 1–10 (2018).

37. Sheng, X. *et al.* Low-conductance and multilevel cmos-integrated nanoscale oxide memristors. *Advanced Electronic Materials* 1800876 (2019).

38. Kumar, S., Strachan, J. P. & Williams, R. S. Chaotic dynamics in nanoscale nbo2 mott memristors for analogue computing. *Nature* **548**, 318–321 (2017). URL http://dx.doi.org/10.1038/nature23307.

39. Albash, T., Martin-Mayor, V. & Hen, I. Analog errors in Ising machines. *Quantum Science and Technology* **4**, 02LT03 (2019).

40. D-Wave. The D-Wave 2X^{TM} Quantum Computer Technology Overview (2015). [Online; accessed 21-January-2019].

41. Villalonga, B. *et al.* A flexible high-performance simulator for verifying and benchmarking quantum circuits implemented on real hardware. *npj Quantum Information* **5**, 1–16 (2019). URL http://dx.doi.org/10.1038/s41534-019-0196-1. 1811.09599.

42. Kielpinski, D. *et al.* Information processing with large-scale optical integrated circuits. In *IEEE International Conference on Rebooting Computing (ICRC'16)* (2016).

43. Tezak, N. *et al.* Integrated coherent ising machines based on self-phase modulation in microring resonators. *IEEE Journal of Selected Topics in Quantum Electronics* **26**, 1–15 (2020).

44. Bunyk, P. I. *et al.* Architectural Considerations in the Design of a Superconducting Quantum Annealing Processor. *IEEE Transactions on Applied Superconductivity* **24**, 1–10 (2014).