RESEARCH ARTICLE | APRIL 11 2022

Desynchronous learning in a physics-driven learning network FREE

Special Collection: Memory Formation

J. F. Wycoff ; S. Dillavou ; M. Stern ; A. J. Liu ; D. J. Durian

J. Chem. Phys. 156, 144903 (2022)
https://doi.org/10.1063/5.0084631
CHORUS

The Journal of Chemical Physics ARTICLE scitation.org/journal/jcp

Desynchronous learning in a physics-driven learning network

Cite as: J. Chem. Phys. 156, 144903 (2022); doi: 10.1063/5.0084631
Submitted: 8 January 2022 • Accepted: 25 March 2022 •
Published Online: 11 April 2022

J. F. Wycoff, S. Dillavou,[a] M. Stern, A. J. Liu, and D. J. Durian

AFFILIATIONS

Department of Physics and Astronomy, University of Pennsylvania, Philadelphia, Pennsylvania 19104, USA

Note: This paper is part of the JCP Special Topic on Memory Formation.
[a]**Author to whom correspondence should be addressed:** dillavou@upenn.edu

ABSTRACT

In a neuron network, synapses update individually using local information, allowing for entirely decentralized learning. In contrast, elements in an artificial neural network are typically updated simultaneously using a central processor. Here, we investigate the feasibility and effect of desynchronous learning in a recently introduced decentralized, physics-driven learning network. We show that desynchronizing the learning process does not degrade the performance for a variety of tasks in an idealized simulation. In experiment, desynchronization actually *improves* the performance by allowing the system to better explore the discretized state space of solutions. We draw an analogy between desynchronization and mini-batching in stochastic gradient descent and show that they have similar effects on the learning process. Desynchronizing the learning process establishes physics-driven learning networks as truly fully distributed learning machines, promoting better performance and scalability in deployment.

INTRODUCTION

Learning is a special case of memory,[1,2] where the goal is to encode targeted functional responses in a network.[3–6] Artificial Neural Networks (ANNs) are complex functions designed to achieve such targeted responses. These networks are trained by using gradient descent on a cost function, which evolves the system's parameters until a local minimum is found.[7,8] Typically, this algorithm is modified such that subsections (batches) of data are used at each training step, effectively adding noise to the gradient calculation, known as Stochastic Gradient Descent (SGD).[9] This algorithm produces more generalizable results,[10–12] i.e., better retention of the underlying features of the dataset, by allowing the system to escape non-optimal fixed points.[13,14] This is reminiscent of noise-improving memory retention in physical systems, such as sheared suspensions,[15–17] where noise prevents the system from settling into equilibrium states where history-dependence is lost.

Recent work[18] has demonstrated the feasibility of entirely distributed, physics-driven learning in self-adjusting resistor networks. This system operates using coupled learning,[19] a theoretical framework for training physical systems using local rules[20–22] and physical processes[23–25] *in lieu* of gradient descent and a central processor. Because of its distributed nature, this system scales in speed and efficiency far better than ANNs and is robust to damage and may one day be a useful platform for machine learning applications, or robust smart sensors. However, just like computational machine learning algorithms, this system (as well as other proposed distributed machine learning systems, e.g., Refs. 26 and 27) relies on a global synchronization of the learning rule such that all elements change their resistance simultaneously. In contrast, the elements of the brain (neurons and synapses) evolve independently,[28,29] suggesting that global synchronization is not required for effective learning. Desynchronizing the updates in machine learning is a largely unexplored topic, as doing so would be computationally inefficient. However, in a distributed system, such as the brain or self-adjusting resistor networks, it is the less restrictive modality,[30] removing the need for a global communication across the network.

Here, we demonstrate that desynchronous implementation of coupled learning is effective in self-adjusting resistor networks, in both simulation and experiment. Furthermore, we show that desynchronous learning can actually *improve* the performance by allowing the system to evolve indefinitely, escaping local minima. We draw a direct analogy between stochastic gradient descent and desynchronous learning and show that they have similar effects on the

learning degrees of freedom in our system. Thus, we are able to remove the final vestige of non-locality from our physics-driven learning network, moving it closer to biological implementations of learning. The ability to learn with entirely independent learning elements is expected to greatly improve the scalability of such physical learning systems.

COUPLED LEARNING

Coupled learning[19] is a theoretical framework similar to equilibrium propagation[26,27] that specifies evolution equations that enable supervised, contrastive learning in physical networks. In the case of a resistor network, inputs and outputs are applied and measured voltages at designated nodes of the network, and the edges self-modify their resistance according to local rules. The learning algorithm is as follows: Input and output nodes are selected, and a set of inputs from the training set is applied as voltages on the input nodes, creating the "free" response of the network. Using the measured outputs from this state V_F^O, the output nodes are then clamped at voltages V_C^O given by

$$\vec{V}_C^O = \eta \vec{V}^D + (1-\eta)\vec{V}_F^O, \tag{1}$$

where V^D are the desired output voltages for this training example and $0 < \eta \leq 1$ is an adjustable global parameter ("hyper-parameter") that controls the strength of the nudge toward the clamped state. Thus, the output nodes are held at values closer to the desired outputs. When $\eta \ll 1$, this algorithm approaches gradient descent on a cost function.[19] This generates the "clamped" response of the network. The voltage drop across each edge in the free ΔV_i^F and clamped ΔV_i^C states then determine the coupled learning rule for changing the resistance of that edge,

$$\delta R_i = \frac{\gamma}{R_i^2}\left([\Delta V_i^C]^2 - [\Delta V_i^F]^2\right), \tag{2}$$

where R_i is the resistance of that edge and γ is a hyper-parameter that determines the learning rate of the system. In effect, this local learning rule lowers the power dissipation of the clamped state relative to the free state, nudging the entire system toward the (by definition) better clamped outputs. The system is then shown a new training example, and the process is repeated, iteratively improving the performance of the free state outputs. When a test set is given to the network to check its performance (by applying the input voltages appropriately), errors are calculated via the difference between the free state outputs and the desired outputs. A more detailed description of coupled learning is given in previous work.[19]

In the above algorithm, it is implicitly assumed that all edges update at the same time. Here, we relax this assumption, modifying the learning rule [Eq. (2)] with a probabilistic element,

$$\Delta R_i(p) = \begin{cases} \delta R_i & \text{with probability } p, \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where $0 < p \leq 1$ is the update probability and $p = 1$ recovers synchronized coupled learning. This modification, especially for low p, fundamentally changes how the system updates. Individual edges may spend long periods entirely static, while the system evolves around them, completely ignoring large changes along the way, that is, learning is desynchronized.

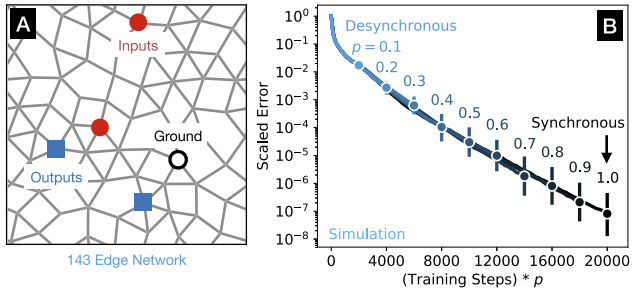

FIG. 1. Coupled learning is successful without synchronous updates. (a) Simulated 143 edge coupled learning network. (b) Test set scaled error [error/error$(t = 0)$] curves averaged over 50 distinct two-input two-output regression tasks as a function of training steps times update probability p. This x axis scaling collapses the curves as each training step causes $143p$ edge updates, on average, proportionally changing the learning rate. Colors denote differing values of p ranging from 0.1 to 1. Error bars at the terminus of each curve denote the range of final error values for a given p when run for 20 000 steps.

Using simulations of coupled learning, as per Ref. 19 but now with desynchronized updates, we find that the learning process is not hampered. In fact, the error as a function of training steps times p consistently collapses for all values of p for a variety of tasks and networks, as shown for a typical example in Fig. 1. This collapse occurs regardless of choice of hyper-parameters η (nudge amplitude) and γ (learning rate). Notably, when updates become more desynchronous (decreasing p), solutions increasingly drift in resistance space from those found for synchronous learning [Fig. 2(a)]. These behaviors suggest that desynchronization may aid in exploring an under-constrained resistance space, much like stochastic gradient descent (SGD) in machine learning, a connection we now formalize mathematically.

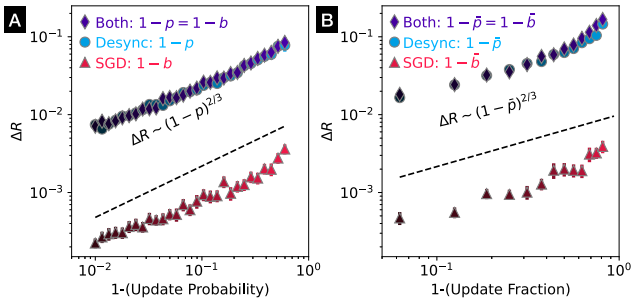

FIG. 2. Desynchronous learning behaves like stochastic gradient descent. (a) Distance in continuous resistor space from synchronized, full-batched solution as a function of $1 - p$ for a 16-edge simulated, continuous network. Each data point represents an average over 50 regression tasks, each with two inputs and two outputs. Note mini-batching (stochastic gradient descent) and desynchronization generate the same power law, as does their combined effect. The vertical shift results from an effective learning rate difference. (b) Same as (a) but with constant number of edges updating or batch size (or both) at each training step.

COMPARISON TO STOCHASTIC GRADIENT DESCENT

In computational machine learning, artificial neural networks can be trained using *batch* gradient descent. In this algorithm, the entire set of training data is run through the network, and a global gradient is taken with respect to each weight in the network, averaged over the training set. The weights are then modified based on this gradient until a local minimum is found. In practice, this method is inefficient at best and intractable at worst.[31] A typical modification to this algorithm is known as stochastic gradient descent (SGD), where instead of the entire training set, a randomly selected subset of training examples (mini-batch) is used to calculate the gradient at each training step.[9] This effectively adds noise to the gradient calculation, speeds processing, and boosts overall performance by allowing the system to continually evolve, escaping from local minima in the global cost function. Stochastic gradient descent has been shown to improve learning performance in different settings, specifically in obtaining lower generalization (test) errors compared to full batch gradient descent. It is, therefore, argued that SGD performs implicit regularization during training, finding minima in the cost landscape that are more likely to generalize to unseen input examples.[11]

This can be more clearly understood by describing training of a neural network as gradient descent dynamics of the learning degrees of freedom $\vec{w}$ (edge weights in a neural network) with an additional diffusion term, following Chaudhari and Soatto.[11] We define $\tilde{b}$ as the fraction of training data points used in a mini-batch. Full-batch ($b = 1$) training simply minimizes the cost function $C(\vec{w})$, and thus, the dynamics may be written as

$$\gamma^{-1} d\vec{w}(t) = -\vec{\nabla}_w C(\vec{w}) dt, \tag{4}$$

which yields solutions $\vec{w}_{\tilde{b}=1}$ that are minima of the cost function. When mini-batching, an additional diffusion term is added to the dynamics,

$$\begin{aligned} \gamma^{-1} d\vec{w}(t) &= -\vec{\nabla}_w C(\vec{w}) dt + \sqrt{2\gamma(\tilde{b}B)^{-1} D(\vec{w})}\, d\vec{W}(t), \\ D(\vec{w}) &= \left[B^{-1} \sum_i \vec{\nabla}_w C_i \otimes \vec{\nabla}_w C_i \right] - \vec{\nabla}_w C \otimes \vec{\nabla}_w C, \end{aligned} \tag{5}$$

where the diffusion matrix $D(\vec{w})$ is defined by outer products of the individual training example gradients, B is the total number of training examples, and dW is a Wiener process (random walk). These dynamics converge to critical points $\vec{w}_{\tilde{b}}$ that are different from the minima of the cost function, $\vec{w}_{\tilde{b}=1}$, by a factor that scales with the fraction of data points not included in each batch $(1 - \tilde{b})$. This difference is the hallmark of regularization, in this case performed implicitly by SGD.

In coupled learning, the desynchronization of edge updates is expected to yield a similar effect. Instead of having different training examples, learning stochastically uses the gradient at independent edges. Therefore, we can define an effective diffusion matrix for desynchronous coupled learning by

$$\gamma^2 D_{eff}(R) = \left[N^{-1} \sum_i \Delta R_i \otimes \Delta R_i \right] - \Delta\vec{R} \otimes \Delta\vec{R}, \tag{6}$$

where N is the total number of edges. Note the similar form to the second line of Eq. (5). With this definition, the analogy of desynchronous coupled learning and SGD is clear, with the edge update probability p playing the role of the batch fraction $\tilde{b}$, and thus, we expect similar results for the two methods. We verify the analogy between desynchronous coupled learning and SGD in simulation.

For simulations with continuously variable resistors, we observe no change in final error when learning is desynchronized. This is consistent with expectations from SGD when tasks have large, multi-dimensional zero-error basins that are always found by the system. However, the analogy between SGD and desynchronization can still be explored by observing the solutions in the resistor space. As a base case, we simulate a $N = 16$ edge network (the same structure we will use in our experimental setup) using the original coupled learning rule [Eq. (2)] with a *full batch* to solve a regression task with $B = 16$ training examples. That is, for a given edge i,

$$\Delta R_i = \sum_{j=1}^{B} \Delta R_{ij} = \sum_{j=1}^{B} \frac{\gamma}{R_i^2} \left([\Delta V_{ij}^C]^2 - [\Delta V_{ij}^F]^2 \right), \tag{7}$$

where j is the index of the training example, summed over all $B = 16$ elements of the training set. This is an entirely deterministic algorithm, given initial conditions of R_i and, thus, a good basis for comparison. Then, we compare two forms of stochasticity, randomly choosing edges (desynchronization) and randomly choosing training examples (SGD). With probability p, we update edges (i), and with probability b, we include each training example in the sum (j). For $b = 1$, we use a full batch, and for $p = 1$, we update every edge synchronously. Coupled learning as described in previous work[18,19] used $p = 1$ and $b \ll 1$ (a single training data point at a time). Decreasing p (desynchronizing) and decreasing b (stochastic mini-batching) do not meaningfully change the final error of the network's solutions in continuous coupled learning but do find *different* solutions than the full-batch synchronous case. In fact, we find they have the same relationships to the fully deterministic solutions,

$$b = 1: \quad L_2\left(\vec{R}(p = 1), \vec{R}(p)\right) \sim (1 - p)^{2/3}, \tag{8}$$

$$p = 1: \quad L_2\left(\vec{R}(b = 1), \vec{R}(b)\right) \sim (1 - b)^{2/3}. \tag{9}$$

Enforcing $p = b$ also gives the same power law, as shown in Fig. 2(a). We may also enforce a randomly selected but consistent fraction of edges ($\tilde{p}$) or of the training set ($\tilde{b}$) to be updated/included for each training step. This is the standard means of mini-batching in SGD, as mentioned previously. We find similar parallels between desynchronous and mini-batched learning in this condition, as shown in Fig. 2(b). The overall multiplicative factor separating the data can be explained by SGD and the desynchronous learning rule having a different effective learning rate. Matching these effective rates collapses all data shown in Figs. 2(a) and 2(b).

This robust analogy between desynchronization and SGD suggests that, in a system with a more disconnected cost landscape, we should expect error *improvements* when desynchronizing coupled learning. We now turn to such a system, our experimental realization of a 16-edge network, where the resistor values are discretized, which decreases the number of degrees of freedom and prevents

the system from settling into a minimum of exactly zero. As we will show, the experimental system successfully learns in the desynchronized regime, in some cases improving upon the synchronized solutions. Desynchronization thus allows a substantial simplification for implementation, especially in large networks, by removing the requirement for simultaneous updates across the entire system.

EXPERIMENTAL (DISCRETE) COUPLED LEARNING

We test desynchronous updates in an experimental realization of coupled learning. In recent work,[18] coupled learning was first implemented in a physical system. In this system, contrastive learning was performed in real time by using two identical twin networks to access the free and clamped states of the network simultaneously. The system was robust to real-world noise and successfully trained itself to perform a variety of tasks using a simplified version of the update rule that allowed only discrete values of R, specifically

$$\delta R_i = \begin{cases} +r_0 & \text{if } |\Delta V_i^C| + \sigma > |\Delta V_i^F|, \\ -r_0 & \text{otherwise.} \end{cases} \tag{10}$$

Note that we have explicitly added the measured bias of the comparators σ, which we find manifests as a random, uniformly distributed variable from 0 to 0.05 V. Previously, each edge in the network performed this update individually, but did so all at once, synchronized by a global clock. Here, we implement this learning rule[32] but incorporate a probabilistic element, such that with probability p each edge updates according to Eq. (10) on a given training step. Thus, we are able to tune the system from entirely synchronous ($p = 1$) to entirely desynchronous ($p \ll 1$).

We implement this probabilistic functionality via separate circuits housed locally with each twin edge of the network, as shown in Fig. 3(a). This circuit, when triggered by a global signal, compares its local oscillating voltage signal to a global "bias" voltage, as shown in Fig. 3(b). The components (comparators, capacitors, and resistors) used in each implementation of the oscillator vary slightly, changing the period and phase of oscillation; thus, the signals on each edge rapidly desynchronize. In experiment, we find a Pearson correlation between pairs of edges to be consistently of order 0.01 for an update probability of 50%, indicating that edges are updating independently. By changing the bias value, we can select a wide range of values of p for our experimental system.

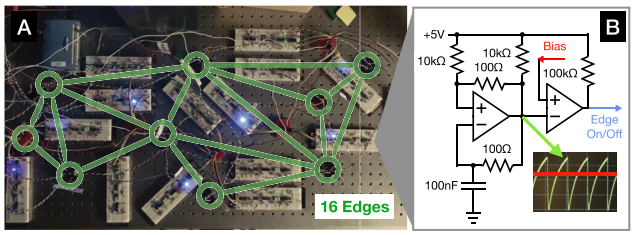

FIG. 3. Circuitry for realization of desynchronous coupled learning. (a) Image of the entire 16-edge network. Edges with LEDs on are active (updating) on this training step. (b) Diagram of the oscillator circuit in each edge in (a). A global bias voltage (red) determines p. Each edge compares the bias against a local oscillator signal (green) to determine if its resistance is updated.

As with the continuous version of coupled learning, desynchronization does not prohibit the discrete, experimental system from learning. In fact, desynchronized learning performs *better* on average than synchronous learning for "allosteric" (fixed input and output) tasks, as apparent in typical error curves as shown in Fig. 4(a). Why does this stochasticity improve final errors? In short, it is because randomness allows the network to explore resistance space. Edges continually evolve when $p < 1$ (desynchronous), whereas for $p = 1$ (synchronous), the system may find a local minimum and remain there indefinitely, as shown by the flat black resistor traces in Fig. 4(b). The ability to escape minima improves as the network becomes more desynchronized, leading to improved final error as p decreases for allosteric tasks in experiment, as shown in Fig. 4(c). As tasks become too difficult, the beneficial effects of desynchronization are diminished. For a two-output, two-input regression task, our 16-edge experimental network shows no benefit from desynchronization. However, as we now show in simulation, increasing the size of the network brings learning back into a regime where desynchronization confers an advantage.

To test the advantages of desynchronous learning for future larger realizations, we perform a simulation tailored to match our experimental system but with more edges. We use the discrete update rule [Eq. (10)], limit our resistance values to 128 linearly spaced values, and use $\sigma = U[0, 0.05]$ V (uniformly sampled between

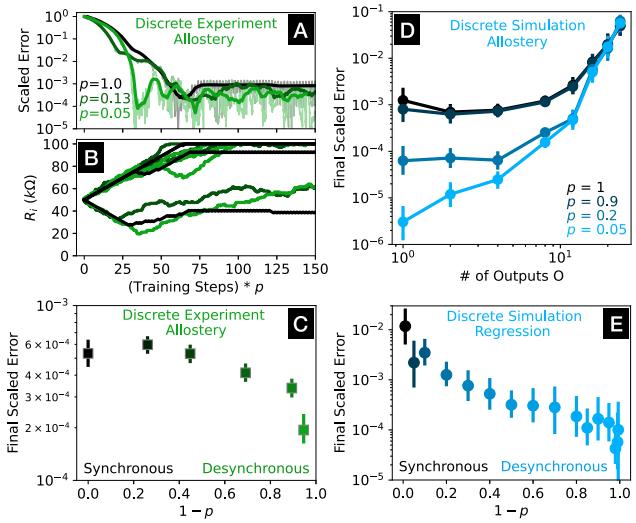

FIG. 4. Desynchronization improves discrete network solutions in experiment and simulation. (a) Scaled error [error/error($t = 0$)] vs training steps scaled by update probability p in experiment for an allosteric task with two inputs and two outputs. One typical raw (faded) and smoothed (color) curve is shown for each of the three values of p. (b) Three resistor values vs training steps scaled by update probability from the experiments shown in (a). (c) Scaled error at the end of training averaged over 25 allosteric tasks each with two inputs and two outputs as a function of p. (d) Scaled error at the end of training for allosteric tasks as a function of number of outputs O. Each data point is an average over 20 tasks, each with O outputs, O inputs, and $O/2$ ground nodes, increasingly constraining the network as O grows. Note the collapse of curves of varying p as the task complexity grows. (e) Scaled test set error at the end of training in simulation averaged over ten regression tasks with two inputs and two outputs. In (d) and (e), the same 143 edge simulated network from Fig. 1(a) is used with the discrete update rule [Eq. (11)].

0 and 0.05 V). As before, to desynchronize learning, we have edges that follow the update rule only with probability p on each training step,

$$\Delta R_i(p) = \begin{cases} \delta R_i & \text{with probability } p, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

That is, Eq. (10) was performed on each edge with probability p. The addition of σ leads to a tendency for the resistor values to drift upward, just like in the experiment, finding lower power solutions, and putting the resistors in a regime where they can take smaller steps relative to their magnitude. From simulations of a 143-edge discrete network, we find that as allostery task complexity (number of both inputs and outputs, O) increases, the beneficial effects of desynchronous learning diminish, as shown in Fig. 4(d). More complex tasks require more desynchronous (lower p) learning to confer an advantage over synchronous learning. For tasks with enough outputs, moderately desynchronous learning yields indistinguishable error from synchronous learning, as shown by the overlap of the blue and black curves on the right of Fig. 4(d).

Unlike the experimental 16-edge network, desynchronization does improve the error for our simulated 143-edge learning a two-input two-output regression task, as shown in Fig. 4(e). We believe that, for such a task, our 16-edge experimental network is in the "too-complex" regime, whereas our simulated 143-edge network is not and therefore shows a monotonic trend in final error with p.

Linear tasks such as allostery and linear regression do not have local minima when the parameters in the linear kernel are free to change continuously.[33] In our networks, the case is different, as the input–output relationship is always a linear function, but the linear kernel depends non-linearly on each resistance value, which are themselves the degrees of freedom. As a result, the cost landscape can have local minima. Even so, we see no evidence for local (non-zero) minima in our continuous simulations, likely because we have a very large number of degrees of freedom relative to the number of constraints. In the discrete case, however, the resistor space has fewer degrees of freedom, leading to more local minima that can trap the synchronous solution and preventing it from finding a global optimum. Thus, desynchronizing the edges ultimately helps find deeper minima in the discrete system (Fig. 4), but not in the continuous system (Fig. 1) where we find no evidence of non-zero minima.

DISCUSSION

In this work, we have demonstrated the feasibility of learning without globally synchronized updates in a physics-based learning network, both with a continuous state space of solutions and a discrete one, in simulation and experiment. In all cases, desynchronizing the learning process does not hamper the ability of the system to learn, and in the discrete resistor space with many local minima, it actually improves learning outcomes. We have shown that this improvement likely comes from a behavior analogous to stochastic gradient descent, namely, injecting noise into the learning process allows the system to escape local minima and find better overall solutions. We have mathematically formalized this analogy and showed that mini-batching and desynchronization produce the same scaling of distance in solution space compared to a fully deterministic (full batch, synchronous) algorithm.

The freedom to avoid global synchronization is an important step toward total decentralization of the learning process in a physical system; it is necessary to make a *learning material*. In this and previous[18] work, the experimental system is still run via a global clock and thus requires a one bit communication with every edge to trigger resistor updates. However, the success at all values of p demonstrates that edges with entirely self-triggered updates should also function well. For a larger, less precise, tighter packed, or three-dimensional learning systems, removing this connection to each edge may greatly simplify construction. Furthermore, allowing desynchronization opens the door for learning with new types of systems that cannot be synchronized, such as elements updating out of equilibrium,[34] or that include thermal noise[29] or other stochastic processes.

In discrete-valued coupled learning, mini-batching alone (the standard in coupled learning) gives inferior results to mini-batching plus desynchronous updates. This suggests that, in other learning problems with many local minima, including in artificial neural networks, desynchronous updates could benefit the learning process. While we are not aware of this desynchronization algorithm used in such a way, similar methods, such as dropout,[35] have been shown to be beneficial in improving generalizability of solutions,[36] similar to stochastic gradient descent. True desynchronization would be extremely inefficient in such a system, as then the entire gradient calculation is necessary for a single edge update. However, we have shown that benefits can be accrued by only moderate desynchronization, e.g., 80% update probability, which slows the learning process proportionately. The true test of the usefulness of this algorithm will be in larger, nonlinear networks solving problems on complex cost landscapes. This is a subject for future work.

ACKNOWLEDGMENTS

The authors thank Marc Z. Miskin for insightful discussions, including on the circuit design. This work was supported by the National Science Foundation [Grant Nos. UPenn MRSEC/DMR-1720530 (S.D. and D.J.D.) and DMR-2005749 (M.S.)] and the Simons Foundation [Investigator Award No. 327939 (A.J.L.)].

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

[1] R. G. Crowder, *Principles of Learning and Memory: Classic Edition* (Psychology Press, 2014).
[2] J. R. Anderson, *Learning and Memory: An Integrated Approach*, 2nd ed. (John Wiley & Sons, Inc., Hoboken, NJ, 2000), p. xviii, 487.
[3] J. J. Hopfield, Proc. Natl. Acad. Sci. U. S. A. **79**, 2554 (1982).
[4] R. McEliece, E. Posner, E. Rodemich, and S. Venkatesh, IEEE Trans. Inf. Theory **33**, 461 (1987).
[5] J. W. Rocks, N. Pashine, I. Bischofberger, C. P. Goodrich, A. J. Liu, and S. R. Nagel, Proc. Natl. Acad. Sci. U. S. A. **114**, 2520 (2017).

[6]M. Stern, M. B. Pinson, and A. Murugan, Phys. Rev. X **10**, 031044 (2020).
[7]Y. LeCun, Y. Bengio, and G. Hinton, Nature **521**, 436 (2015).
[8]P. Mehta, M. Bukov, C.-H. Wang, A. G. R. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, Phys. Rep. **810**, 1 (2019).
[9]S. Ruder, arXiv:1609.04747 (2017).
[10]N. S. Keskar and R. Socher, arXiv:1712.07628 (2017).
[11]P. Chaudhari and S. Soatto, in *2018 Information Theory and Applications Workshop (ITA)* (IEEE, San Diego, CA, 2018), pp. 1–10.
[12]S. L. Smith, B. Dherin, D. G. T. Barrett, and S. De, *On the Origin of Implicit Regularization in Stochastic Gradient Descent* (ICLR, 2021) available at https://iclr.cc/virtual/2021/poster/3157.
[13]Y. Feng and Y. Tu, Proc. Natl. Acad. Sci. U. S. A. **118**, e2015617118 (2021).
[14]M. Ruiz-Garcia, G. Zhang, S. S. Schoenholz, and A. J. Liu, in *Proceedings of the 38th International Conference on Machine Learning* (PMLR, 2021), Vol, 139, pp. 9157–9167.
[15]N. C. Keim and S. R. Nagel, Phys. Rev. Lett. **107**, 010603 (2011).
[16]J. D. Paulsen, N. C. Keim, and S. R. Nagel, Phys. Rev. Lett. **113**, 068301 (2014).
[17]N. C. Keim, J. D. Paulsen, Z. Zeravcic, S. Sastry, and S. R. Nagel, Rev. Mod. Phys. **91**, 035002 (2019).
[18]S. Dillavou, M. Stern, A. J. Liu, and D. J. Durian, arXiv:2108.00275 (2021).
[19]M. Stern, D. Hexner, J. W. Rocks, and A. J. Liu, Phys. Rev. X **11**, 021045 (2021).
[20]M. Stern, V. Jayaram, and A. Murugan, Nat. Commun. **9**, 4303 (2018).
[21]M. Stern, C. Arinze, L. Perez, S. E. Palmer, and A. Murugan, Proc. Natl. Acad. Sci. U. S. A. **117**, 14843 (2020).
[22]N. Pashine, Phys. Rev. Mater. **5**, 065607 (2021).
[23]N. Pashine, D. Hexner, A. J. Liu, and S. R. Nagel, Sci. Adv. **5**, eaax4215 (2019).
[24]D. Hexner, N. Pashine, A. J. Liu, and S. R. Nagel, Phys. Rev. Res. **2**, 043231 (2020).
[25]D. Hexner, A. J. Liu, and S. R. Nagel, Proc. Natl. Acad. Sci. U. S. A. **117**, 31690 (2020).
[26]B. Scellier and Y. Bengio, Front. Comput. Neurosci. **11**, 24 (2017).
[27]J. Kendall, R. Pantone, K. Manickavasagam, Y. Bengio, and B. Scellier, arXiv:2006.01981 (2020).
[28]L. F. Abbott and S. B. Nelson, Nat. Neurosci. **3**, 1178 (2000).
[29]D. Kappel, S. Habenschuss, R. Legenstein, and W. Maass, PLoS Comput. Biol. **11**, e1004485 (2015).
[30]D. Dolev, J. Y. Halpern, and H. R. Strong, J. Comput. Syst. Sci. **32**, 230 (1986).
[31]N. Golmant, N. Vemuri, Z. Yao, V. Feinberg, A. Gholami, K. Rothauge, M. W. Mahoney, and J. Gonzalez, arXiv:1811.12941 (2018).
[32]Specifically in this work we use comparators and an XOR gate to evaluate XOR $[(\Delta V_i^C > \Delta V_i^F), (\Delta V_i^C + \Delta V_i^C > 0)]$.
[33]A. C. Rencher and G. B. Schaalje, *Linear Models in Statistics*, 2nd ed. (Wiley-Interscience, Hoboken, NJ, 2008).
[34]M. Stern, S. Dillavou, M. Z. Miskin, D. J. Durian, and A. J. Liu, arXiv:2112.11399 (2021).
[35]In dropout, some fraction of edges in a layer of a neural network are removed for that training step. This is distinct from desynchronous learning, where all edges are present for calculating the outputs, but some simply do not update.
[36]N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, J. Mach. Learn. Res. **15**, 1929 (2014).