ORIGINAL RESEARCH
published: 09 May 2022
doi: 10.3389/frai.2022.891624

Neural Network Training With Asymmetric Crosspoint Elements

Murat Onen[1,2*‡], Tayfun Gokmen[1*‡], Teodor K. Todorov[1], Tomasz Nowicki[1], Jesús A. del Alamo[2], John Rozen[1], Wilfried Haensch[1] and Seyoung Kim[1*†]

[1] IBM Thomas J. Watson Research Center, Yorktown Heights, NY, United States, [2] Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, United States

Edited by:
Saban Öztürk,
Amasya University, Turkey

Reviewed by:
Mucahid Barstugan,
Konya Technical University, Turkey
Umut Özkaya,
Konya Technical University, Turkey
Bing Li,
Capital Normal University, China

***Correspondence:**
Murat Onen
monen@mit.edu
Tayfun Gokmen
tgokmen@us.ibm.com
Seyoung Kim
kimseyoung@postech.ac.kr

†Present address:
Seyoung Kim,
Department of Materials Science and Engineering, POSTECH, Pohang, South Korea

‡These authors have contributed equally to this work

Specialty section:
This article was submitted to Machine Learning and Artificial Intelligence, a section of the journal Frontiers in Artificial Intelligence

Received: 08 March 2022
Accepted: 01 April 2022
Published: 09 May 2022

Citation:
Onen M, Gokmen T, Todorov TK, Nowicki T, del Alamo JA, Rozen J, Haensch W and Kim S (2022) Neural Network Training With Asymmetric Crosspoint Elements. Front. Artif. Intell. 5:891624. doi: 10.3389/frai.2022.891624

Analog crossbar arrays comprising programmable non-volatile resistors are under intense investigation for acceleration of deep neural network training. However, the ubiquitous asymmetric conductance modulation of practical resistive devices critically degrades the classification performance of networks trained with conventional algorithms. Here we first describe the fundamental reasons behind this incompatibility. Then, we explain the theoretical underpinnings of a novel fully-parallel training algorithm that is compatible with asymmetric crosspoint elements. By establishing a powerful analogy with classical mechanics, we explain how device asymmetry can be exploited as a useful feature for analog deep learning processors. Instead of conventionally tuning weights in the direction of the error function gradient, network parameters can be programmed to successfully minimize the total energy (Hamiltonian) of the system that incorporates the effects of device asymmetry. Our technique enables immediate realization of analog deep learning accelerators based on readily available device technologies.

Keywords: analog computing, DNN training, hardware accelerator architecture, neuromorphic accelerator, learning algorithm

INTRODUCTION

Deep learning has caused a paradigm shift in domains such as object recognition, natural language processing, and bioinformatics which benefit from classifying and clustering representations of data at multiple levels of abstraction (Lecun et al., 2015). However, the computational workloads to train state-of-the-art deep neural networks (DNNs) demand enormous computation time and energy costs for data centers (Strubell et al., 2020). Since larger neural networks trained with bigger data sets generally provide better performance, this trend is expected to accelerate in the future. As a result, the necessity to provide fast and energy-efficient solutions for deep learning has invoked a massive collective research effort by industry and academia (Chen et al., 2016; Jouppi et al., 2017; Rajbhandari et al., 2020).

Highly optimized digital application-specific integrated circuit (ASIC) implementations have attempted to accelerate DNN workloads using reduced-precision arithmetic for the computationally intensive matrix operations. Although acceleration of inference tasks was achieved by using 2-bit resolution (Choi et al., 2019), learning tasks were found to require at least hybrid 8-bit floating-point formats (Sun et al., 2019) which still imposes considerable energy consumption and processing time for large networks. Therefore, beyond-digital approaches that can efficiently handle training workloads are actively sought for.

The concept of in-memory computation with analog resistive crossbar arrays is under intense study as a promising alternative. These frameworks were first designed to make use of Ohm's and Kirchhoff's Laws to perform parallel vector–matrix multiplications (see **Supplementary Sections 2.1**, **2.2** for details), which constitute $\approx 2/3$ of the overall computational load (Steinbuch, 1961). However, unless the remaining $\approx 1/3$ of computations during the update cycle is parallelized as well, the acceleration factors provided by analog arrays will be a mere $3\times$ at best with respect to conventional digital processors. It was much later discovered that rank-one outer products can also be achieved in parallel, using pulse-coincidence and incremental changes in device conductance (Burr et al., 2015; Gokmen and Vlasov, 2016). Using this method, an entire crossbar array can be updated in parallel, without explicitly computing the outer product[1] or having to read the value of any individual crosspoint element (Gokmen et al., 2017). As a result, all basic primitives for DNN training using the Stochastic Gradient Descent (SGD) algorithm can be performed in a fully-parallel fashion using analog crossbar architectures. However, this parallel update method imposes stringent device requirements since its performance is critically affected by the conductance modulation characteristics of the crosspoint elements. In particular, asymmetric conductance modulation characteristics (i.e., having mismatch between positive and negative conductance adjustments) are found to deteriorate classification accuracy by causing inaccurate gradient accumulation (Yu et al., 2015; Agarwal et al., 2016, 2017; Gokmen and Vlasov, 2016; Gokmen et al., 2017, 2018; Ambrogio et al., 2018). Unfortunately, all analog resistive devices to date have asymmetric characteristics, which poses a major technical barrier before the realization of analog deep learning processors.

In addition to widespread efforts to engineer ideal resistive devices (Woo and Yu, 2018; Fuller et al., 2019; Grollier et al., 2020; Yao et al., 2020), many high-level mitigation techniques have been proposed to remedy device asymmetry. Despite numerous published simulated and experimental demonstrations, none of these studies so far provides a solution for which the analog processor still achieves its original purpose: energy-efficient acceleration of deep learning. The critical issue with the existing techniques is the requirement of serial accessing to crosspoint elements one-by-one or row-by-row (Prezioso et al., 2015; Yu et al., 2015; Agarwal et al., 2017; Burr et al., 2017; Ambrogio et al., 2018; Li et al., 2018, 2019; Cai et al., 2019; Sebastian et al., 2020). Methods involving serial operations include reading conductance values individually, engineering update pulses to artificially force symmetric modulation, and carrying or resetting weights periodically. Furthermore, some approaches offload the gradient computation to digital processors, which not only requires consequent serial programming of the analog matrix, but also bears the cost of outer product calculation (Prezioso et al., 2015; Yu et al., 2015; Li et al., 2018, 2019; Cai et al., 2019; Sebastian et al., 2020). Updating an $N \times N$ crossbar array with these serial routines would require at least N or even N^2 operations. For practical array sizes, the update cycle would simply take too much computational time and energy. In conclusion, for implementations that compromise parallelism, whether or not the asymmetry issue is resolved becomes beside the point since computational throughput and energy efficiency benefits over conventional digital processors are lost for practical applications. It is therefore urgent to devise a method that deals with device asymmetry while employing only fully-parallel operations.

[1]The result of the outer product is not returned to the user, but implicitly applied to the network.

Recently, our group proposed a novel fully-parallel training method, *Tiki-Taka,* that can successfully train DNNs based on asymmetric resistive devices with asymmetric modulation characteristics (Gokmen and Haensch, 2020). This algorithm was empirically shown in simulation to deliver ideal-device-equivalent classification accuracy for a variety of network types and sizes emulated with asymmetric device models (Gokmen and Haensch, 2020). However, the missing theoretical underpinnings of the proposed algorithmic solution as well as the cost of doubling analog hardware previously limited the method described in Gokmen and Haensch (2020).

In this paper, we first theoretically explain why device asymmetry has been a fundamental problem for SGD-based training. By establishing a powerful analogy with classical mechanics., we further establish that the *Tiki-Taka* algorithm minimizes the total energy (Hamiltonian) of the system, incorporating the effects of device asymmetry. The present work formalizes this new method as Stochastic Hamiltonian Descent (SHD) and describes how device asymmetry can be exploited as a useful feature in a fully-parallel training. The advanced physical intuition allows us to enhance the original algorithm and achieve a reduction in hardware cost of 50%, improving its practical relevance. Using simulated training results for different device families, we conclude that SHD provides better classification accuracy and faster convergence with respect to SGD-based training in all applicable scenarios. The contents of this paper provide a guideline for the next generation of crosspoint elements as well as specialized algorithms for analog computing.

THEORY

Neural networks can be construed as many layers of matrices (i.e., weights, W) performing affine transformations followed by non-linear activation functions. Training (i.e., learning) process refers to the adjustment of W such that the network response to a given input produces the target output for a labeled dataset. The discrepancy between the network and target outputs is represented with a scalar error function, E, which the training algorithm seeks to minimize. In the case of the conventional SGD algorithm (Cauchy, 1847), values of W are incrementally modified by taking small steps (scaled by the learning rate, η) in the direction of the gradient of the error function sampled for each input. Computation of the gradients is performed by the backpropagation algorithm consisting of forward pass, backward pass, and update subroutines (Rumelhart et al., 1986) (**Figure 1A**). When the discrete nature of DNN training is analyzed in the continuum limit, the time evolution of W can

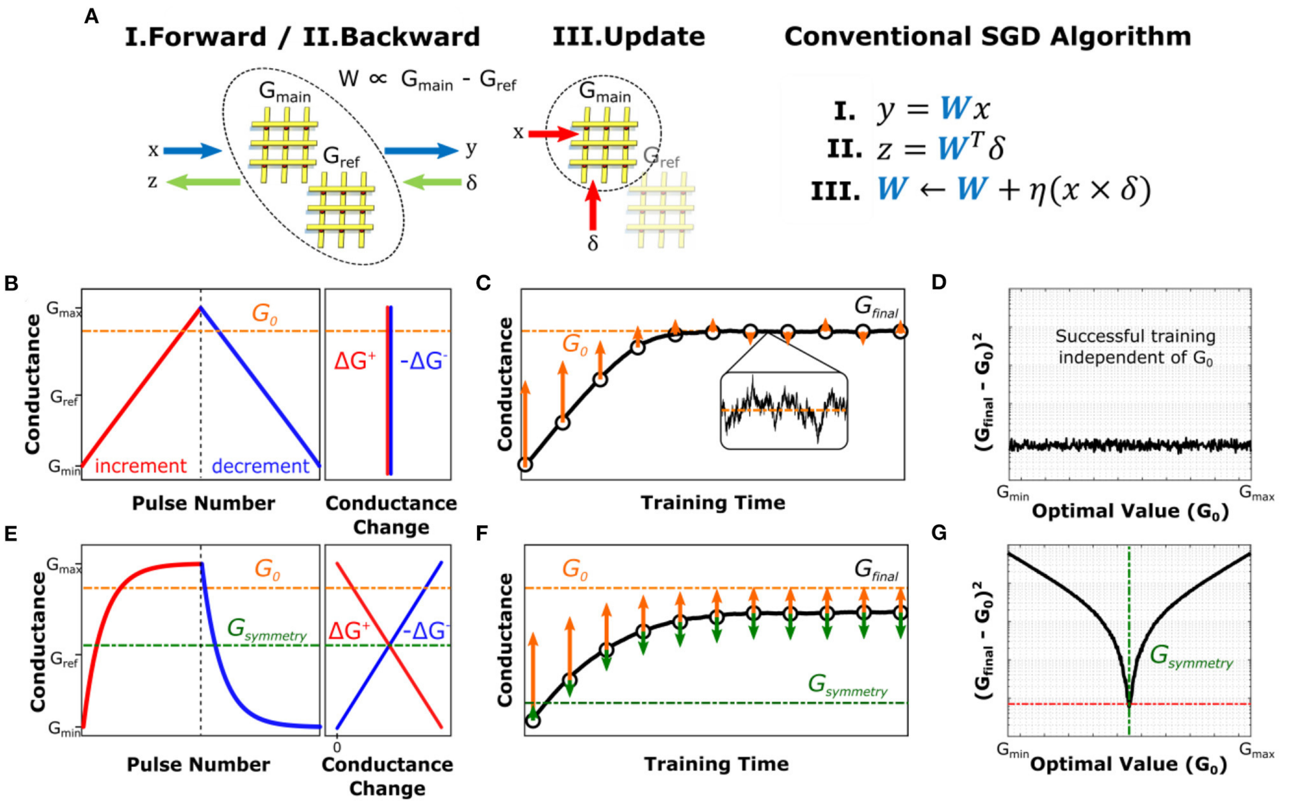

FIGURE 1 | Effect of asymmetric conductance modulation for SGD-based training. **(A)** Schematic and pseudocode of processes for conventional SGD algorithm (Cauchy, 1847). Vectors x, y, represent the input and output vectors in the forward pass whereas δ, z contain the backpropagated error information. The analog architecture schematic is only shown for a single layer, where all vectors are propagated between upper and lower network layers in general. The pseudocode only describes operations computed in the analog domain, whereas digital computations such as activation functions are not shown for simplicity. **(B)** Sketch of conductance modulation behavior of a symmetric crosspoint device. **(C)** Simulated single-parameter optimization result for the symmetric device shown in **(B)**. conductance successfully converges to the optimal value for the problem at hand, G_0. **(D)** Simulated residual distance between the final converged value, G_{final}, and G_0 for training the device with characteristics shown in **(B)** for datasets with different optimal values. **(E)** Sketch of conductance modulation behavior of an asymmetric crosspoint device. The point at which $\Delta G^+ = \Delta G^-$ is defined as the symmetry point of the device ($G_{symmetry}$) **(F)** Simulated training result for the same single-parameter optimization with the asymmetric device shown in **(E)**. Device conductance fails to converge to G_0, but instead settles at a level between G_0 and $G_{symmetry}$. **(G)** Simulated residual distance (in semilog scale) between the final value, G_{final}, and G_0 for training the device with characteristics shown in **(E)** for datasets with different optimal values.

be written as a Langevin equation:

$$\dot{W} = -\eta \left[\frac{\partial E}{\partial W} + \epsilon(t)\right] \quad (1)$$

where η is the learning rate and $\epsilon(t)$ is a fluctuating term with zero-mean, accounting for the inherent stochasticity of the training procedure (Feng and Tu, 2023). As a result of this training process, W converges to the vicinity of an optimum W_0, at which $\frac{\partial E}{\partial W} = 0$ but $\dot{W}$ is only on average 0 due to the presence of $\epsilon(t)$. For visualization, if the training dataset is a cluster of points in space, W_0 is the center of that cluster, where each individual point still exerts a force ($\epsilon(t)$) that averages out to 0 over the whole dataset.

In the case of analog crossbar-based architectures, the linear matrix operations are performed on arrays of physical devices, whereas all non-linear computations (e.g., activation and error functions) are handled at peripheral circuitry. The strictly positive nature of device conductance requires representation of each weight by means of the differential conductance of a pair of crosspoint elements (i.e., $W \propto G_{main} - G_{ref}$). Consequently, vector-matrix multiplications for the forward and backward passes are computed by using both the main and the reference arrays (**Figure 1A**). On the other hand, the gradient accumulation and updates are only performed on the main array using bidirectional conductance changes while the values of the reference array are kept constant[2]. In this section, to illustrate the basic dynamics of DNN training with analog architectures, we study a single-parameter optimization problem (linear regression) which can be considered as the simplest "neural network".

[2]For implementations using devices showing unidirectional conductance modulation characteristics, both the main and the reference array are updated. When SGD is used as the training algorithm, values of G_{ref} are not critical as long as they fall in the midrange of G_{main}'s conductance span (Gokmen and Vlasov, 2016).

The weight updates in analog implementations are carried out through modulation of the conductance values of the crosspoint elements, which are often applied by means of pulses. These pulses cause incremental changes in device conductance ($\Delta G^{+,-}$). In an ideal device, these modulation increments are of equal magnitude in both directions and independent of the device conductance, as shown in **Figure 1B**. It should be noted that the series of modulations in the training process is inherently non-monotonic as different input samples in the training set create gradients with different magnitudes and signs in general. Furthermore, as stated above, even when an optimum conductance, G_0, is reached ($W_0 \propto G_0 - G_{ref}$), continuing the training operation would continue modifying the conductance in the vicinity of G_0, as shown in **Figure 1C**. Consequently, G_0 can be considered as a dynamic equilibrium point of the device conductance from the training algorithm point of view.

Despite considerable technological efforts in the last decade, analog resistive devices with the ideal characteristics illustrated in **Figure 1B** have yet to be realized. Instead, practical analog resistive devices display asymmetric conductance modulation characteristics such that unitary (i.e., single-pulse) modulations in opposite directions do not cancel each other in general, i.e., $\Delta G^{+}(G) \neq -\Delta G^{-}(G)$. However, with the exception of some device technologies such as Phase Change Memory (PCM) which reset abruptly (Burr et al., 2017; Sebastian et al., 2017; Ambrogio et al., 2018), many crosspoint elements can be modeled by a smooth, monotonic, non-linear function that shows saturating behavior at its extrema as shown in **Figure 1E** (Kim et al., 2019b, 2020; Yao et al., 2020). For such devices, there exists a unique conductance point, $G_{symmetry}$, at which the magnitude of an incremental conductance change is equal to that of a decremental one. As a result, the time evolution of G during training can be rewritten as:

$$\dot{G} = -\eta\left[\frac{\partial E}{\partial G} + \epsilon(t)\right] - \eta\kappa\left|\frac{\partial E}{\partial G} + \epsilon(t)\right| f_{hardware} \qquad (2)$$

where κ is the asymmetry factor and $f_{hardware}$ is the functional form of the device asymmetry (see **Supplementary Section 1.1** for derivation). In this expression, the term $-\eta\left|\frac{\partial E}{\partial G} + \epsilon(t)\right|$ signifies that the direction of the change related to asymmetric behavior is solely determined by $f_{hardware}$, irrespective of the direction of the intended modulation. For the exponentially saturating device model shown in **Figure 1E**, $f_{hardware} = G - G_{symmetry}$, which indicates that each and every update event has a component that drifts the device conductance toward its symmetry point. A simple observation of this effect is when enough equal number of incremental and decremental changes are applied to these devices in a random order, the conductance value converges to the vicinity of $G_{symmetry}$ (Kim et al., 2020). Therefore, this point can be viewed as the physical equilibrium point for the device, as it is the only conductance value that is dynamically stable.

It is essential to realize that there is in general no relation between $G_{symmetry}$ and G_0, as the former is entirely device-dependent while the latter is problem-dependent. As a result, for an asymmetric device, two equilibria of hardware and software create a competing system, such that the conductance value converges to a particular conductance somewhere between $G_{symmetry}$ and G_0, for which the driving forces of the training algorithm and device asymmetry are balanced out (**Figure 1F**). In examples shown in **Figures 1C,F**, G_0 of the problem is purposefully designed to be far away from $G_{symmetry}$, so as to depict a case for which the effect of asymmetry is pronounced. Indeed, it can be seen that the discrepancy between the final converged value, G_{final}, and G_0 strongly depends on the relative position of G_0 with respect to the $G_{symmetry}$ (**Figure 1G**), unlike that of ideal devices (**Figure 1D**). Detailed derivation of these dynamics can be found in **Supplementary Section 1.2**.

In contrast to SGD, our new training algorithm, illustrated in **Figure 2A**, separates both the forward path and error backpropagation from the update function. For this purpose, two array pairs (instead of a single pair), namely A_{main}, A_{ref}, C_{main}, C_{ref} are utilized to represent each layer (Gokmen and Haensch, 2020). In this representation, $A = A_{main} - A_{ref}$ stands for the auxiliary array and $C = C_{main} - C_{ref}$ stands for the core array.

The new training algorithm operates as follows. At the beginning of the training process, A_{ref} and C_{ref} are initialized to $A_{main,symmetry}$ and $C_{main,symmetry}$, respectively [reasons will be clarified later, see Section M1. Array Initialization (Zero-Shifting) for details] following the method described in Kim et al. (2020). As illustrated in **Figure 2A**, first, forward and backward pass cycles are performed on the array-pair C (Steps I and II), and corresponding updates are performed on A_{main} (scaled by the learning rate η_A) using the parallel update scheme discussed in Gokmen and Vlasov (2016) (Step III). In other words, the updates that would have been applied to C in a conventional SGD scheme are directed to A instead.

Then, every τ cycles, another forward pass is performed on A, with a vector u, which produces $v = Au$ (Step IV). In its simplest form, u can be a vector of all "0"s but one "1", which then makes v equal to the row of A corresponding to the location of "1" in u. Finally, the vectors u and v are used to update C_{main} with the same parallel update scheme (scaled by the learning rate η_c) (Step V). These steps (IV and V shown in **Figure 2A**) essentially partially add the information stored in A to C_{main}. The complete pseudocode for the algorithm can be found in Section M2. Pseudocode for SHD Algorithm.

At the end of the training procedure C alone contains the optimized network, to be later used in inference operations (hence the name core). Since A receives updates computed over $\frac{\partial E}{\partial C}$, which have zero-mean once C is optimized, its active component, A_{main}, will be driven toward $A_{main,symmetry}$. The choice to initialize the stationary reference array, A_{ref}, at $A_{main,symmetry}$ ensures that $A = 0$ at this point (i.e., when C is optimized), thus generating no updates to C in return.

With the choice of u vectors made above, every time steps IV and V are performed, the location of the "1" for the u vector would change in a cyclic fashion, whereas in general any set of orthogonal u vectors can be used for this purpose (Gokmen and Haensch, 2020). We emphasize that these steps should not be confused with weight carrying (Agarwal et al., 2017; Ambrogio et al., 2018), as C is updated by only a fractional amount in the direction of A as $\eta_C << 1$ and at no point information stored

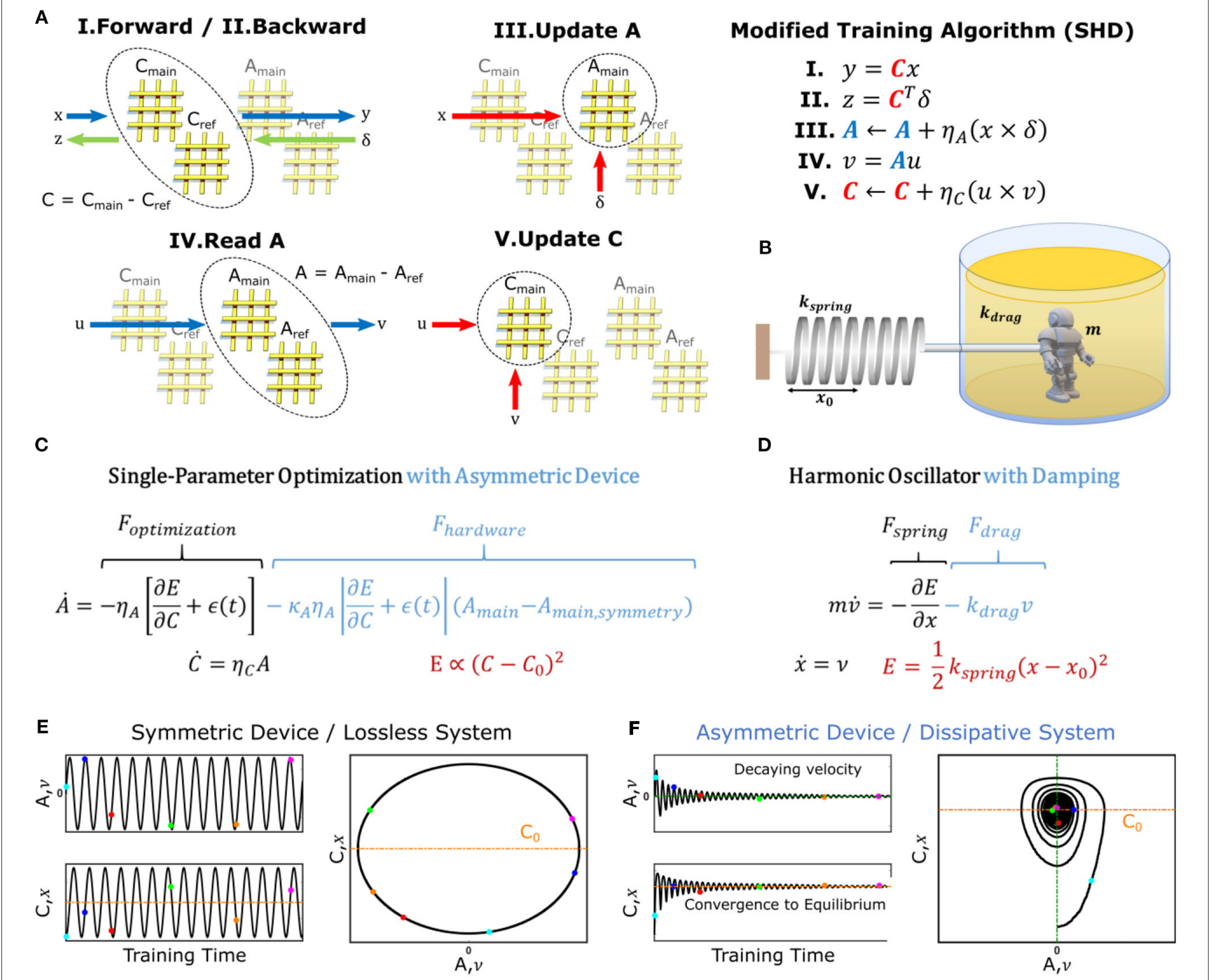
A
I.Forward / II.Backward
C = C_main - C_ref
III.Update A
IV.Read A
A = A_main - A_ref
V.Update C
Modified Training Algorithm (SHD)
I. $y = Cx$
II. $z = C^T\delta$
III. $A \leftarrow A + \eta_A(x \times \delta)$
IV. $v = Au$
V. $C \leftarrow C + \eta_C(u \times v)$
B
k_{spring}
k_{drag}
m
x_0
C
Single-Parameter Optimization with Asymmetric Device
$F_{optimization}$
$F_{hardware}$
$\dot{A} = -\eta_A\left[\frac{\partial E}{\partial C} + \epsilon(t)\right] - \kappa_A\eta_A\left|\frac{\partial E}{\partial C} + \epsilon(t)\right|(A_{main} - A_{main,symmetry})$
$\dot{C} = \eta_C A$
$E \propto (C - C_0)^2$
D
Harmonic Oscillator with Damping
F_{spring}
F_{drag}
$m\dot{v} = -\frac{\partial E}{\partial x} - k_{drag}v$
$\dot{x} = v$
$E = \frac{1}{2}k_{spring}(x - x_0)^2$
E
Symmetric Device / Lossless System
Training Time
A,v
C,x
C0
F
Asymmetric Device / Dissipative System
Decaying velocity
Convergence to Equilibrium
Training Time

FIGURE 2 | DNN training with Stochastic Hamiltonian Descent (SHD) algorithm and dynamics of a dissipative harmonic oscillator. **(A)** Schematic and pseudocode of training process using the SHD algorithm. The pseudocode only describes operations computed in the analog domain, whereas digital computations such as non-linear error functions are not shown for simplicity. **(B)** Illustration of a damped harmonic oscillator system. **(C)** Differential equations describing the evolution of the parameters with the SHD training algorithm in the continuum limit. **(D)** Equations of motion describing the dynamics of a harmonic oscillator. **(E)** Simulated results for a single-parameter optimization task using the SHD algorithm with symmetric devices described in **Figure 1B**. **(F)** Simulated results for a single-parameter optimization task using the SHD algorithm with asymmetric devices described in **Figure 1E**.

in A is externally erased (i.e., A is never reset). Instead, A and C create a coupled-dynamical-system, as the changes performed on both are determined by the values of one another.

Furthermore, it is critical to realize that the algorithm shown in **Figure 2** consists of only fully-parallel operations. Similar to steps *I* and *II* (forward and backward pass on C), steps *IV* is yet another matrix-vector multiplication that is performed by means of Ohm's and Kirchhoff's Laws. On the other hand, the update steps *III* and *V* are performed by the stochastic update scheme (Gokmen and Vlasov, 2016). This update method does not explicitly compute the outer products ($x \times \delta$ and $u \times v$), but instead uses a statistical method to modify all weights in parallel proportional to those outer products. As a result, no serial operations are required at any point throughout the training operation, enabling high throughput and energy efficiency benefits in deep learning computations.

For the same linear regression problem studied above, the discrete-time update rules given in **Figure 2A** can be rewritten as a pair of differential equations in the continuum limit that describe the time evolution of subsystems A and C (**Figure 2C**) as:

$$\dot{A} = -\eta_A\left[\frac{\partial E}{\partial C} + \epsilon(t)\right] - \eta_A\kappa_A\left|\frac{\partial E}{\partial C} + \epsilon(t)\right| \times (A_{main} - A_{main,\ symmetry}) \quad (3)$$

$$\dot{C} = \eta_C A + \eta_C\kappa_C|A| \times (C_{main} - C_{main,symmetry}) \quad (4)$$

It can be noticed that this description of the coupled system has the same arrangement as the equations governing the motion of a damped harmonic oscillator (**Figures 2B,D**). In this analogy, subsystem A corresponds to velocity, ν, while subsystem C maps to position, x, allowing the scalar error function of the optimization problem[3], $(C - C_0)^2$, to map onto the scalar potential energy of the physical framework, $\frac{1}{2}k_{spring}\,(x - x_0)^2$. Moreover, for implementations with asymmetric devices, an additional force term, $F_{hardware}$, needs to be included in the differential equations to reflect the hardware-induced effects on the conductance modulation. As discussed earlier, for the device model shown in **Figure 1E** this term is proportional to $A_{main} - A_{main,symmetry}$. If we assume $A_{ref} = A_{main,symmetry}$ (this assumption will be explained later), we can rewrite $F_{hardware}$ as a function of $A_{main} - A_{ref}$, which then resembles a drag force, F_{drag}, that is linearly proportional to velocity ($\nu \propto A = A_{main} - A_{ref}$) with a variable (but strictly non-negative) drag coefficient k_{drag}. In general, the $F_{hardware}$ term can have various functional forms for devices with different conductance modulation characteristics but is completely absent for ideal devices. Note that, only to simplify the physical analogy, we ignore the effect of asymmetry in subsystem C, which yields the equation shown in **Figure 2C** (instead of Equation 4). This decision will be justified in the Section Discussions.

Analogous to the motion of a lossless harmonic oscillator, the steady-state solution for this modified optimization problem with ideal devices (i.e., $F_{hardware} = 0$) has an oscillatory behavior (**Figure 2E**). This result is expected, as in the absence of any dissipation mechanism, the total energy of the system cannot be minimized (it is constant) but can only be continuously transformed between its potential and kinetic components. On the other hand, for asymmetric devices, the dissipative force term $F_{hardware}$ gradually annihilates all energy in the system, allowing $A \propto \nu$ to converge to 0 ($E_{kinetic} \rightarrow 0$) while $C \propto x$ converges to $C_0 \propto x_0$ ($E_{potential} \rightarrow 0$). Based on these observations, we rename the new training algorithm as *Stochastic Hamiltonian Descent (SHD)* to highlight the evolution of the system parameters in the direction of reducing the system's total energy (Hamiltonian). These dynamics can be visualized by plotting the time evolution of A vs. that of C, which yields a spiraling path representing decaying oscillations for the optimization process with asymmetric devices (**Figure 2F**), in contrast to elliptical trajectories observed for ideal lossless systems (**Figure 2E**).

Following the establishment of the necessity to have dissipative characteristics, here we analyze conditions at which device asymmetry provides this behavior. It is well-understood in mechanics that for a force to be considered dissipative, its product with velocity (i.e., power) should be negative (otherwise it would imply energy injection into the system). In other words, the hardware-induced force term $F_{hardware} = -\kappa_A \eta_A \left|\frac{\partial E}{\partial C} + \epsilon(t)\right| (A_{main} - A_{main,symmetry})$ and the velocity, $\nu = A_{main} - A_{ref}$, should always have opposite signs. Furthermore, from the steady-state analysis, for the system to be stationary ($\nu = 0$) at the point with minimum potential energy ($x = x_0$), there should be no net force ($F = 0$). Both of these arguments indicate that, for the SHD algorithm to function properly, A_{ref} must be set to $A_{main,\,symmetry}$. Note that as long as the crosspoint elements are realized with asymmetric devices (opposite to SGD requirement) and a symmetry point exists for each device, the shape of their modulation characteristics is not critical for successful DNN training with the SHD algorithm. Importantly, while a technologically viable solution for symmetric devices has not yet been found over decades of investigation, asymmetric devices that satisfy the aforementioned properties are abundant.

A critical aspect to note is that the SGD and the SHD algorithms are fundamentally disjunct methods governed by completely different dynamics. The SGD algorithm attempts to optimize the system parameters while disregarding the effect of device asymmetry and thus converges to the minimum of a wrong energy function. On the other, the system variables in an SHD-based training do not conventionally evolve in directions of the error function gradient, but instead, are tuned to minimize the total energy incorporating the hardware-induced terms. The most obvious manifestation of these properties can be observed when the training is initialized from the optimal point (i.e., the very lucky guess scenario) since any "training" algorithm should at least be able to maintain this optimal state. For the conventional SGD, when $W = W_0$, the zero-mean updates applied to the network were shown above to drift W away from W_0 toward $W_{symmetry}$. On the other hand, for the SHD method, when $A = 0$ and $C = C_0$, the zero-mean updates applied on A do not have any adverse effect since A_{main} is already at $A_{main,\,symmetry}$ for $A = 0$. Consequently, no updates are applied to C either as $\dot{C} = A = 0$. Therefore, it is clear that SGD is fundamentally incompatible with asymmetric devices, even when the solution is guessed correctly from the beginning, whereas the SHD does not suffer from this problem. Note that the propositions made for SGD can be further generalized to other crossbar-compatible training methods such as equilibrium propagation (Scellier and Bengio, 2017) and deep Boltzmann machines (Salakhutdinov and Hinton, 2009), which can also be adapted to be used with asymmetric devices following the approach discussed in this paper.

Finally, we appreciate that large-scale neural networks are much more complicated systems with respect to the problem analyzed here. Similarly, different analog devices show a wide range of conductance modulation behaviors, as well as bearing other non-idealities such as analog noise, imperfect retention, and limited endurance. However, the theory we provide here finally provides an intuitive explanation for: (1) why device asymmetry is fundamentally incompatible with SGD-based training and (2) how to ensure accurate optimization while only using fully-parallel operations. We conclude that asymmetry-related issues within SGD should be analyzed in the context of competing equilibria, where the optimum for the classification problem is not even a stable solution at steady-state. In addition to this simple stability analysis, the insight to modify the optimization landscape

[3]Conventionally error functions are written in terms of the difference between the network response and the target output and gradients are computed accordingly. However, in the absence of any stochasticity, ϵ, it can instead be written in terms of the network weights and their optimal values as well for notational purposes.

to include non-ideal hardware effects allows other fully-parallel solutions to be designed in the future using advanced concepts from optimal control theory. As a result, these parallel methods enable analog processors to provide high computational throughput and energy efficiency benefits over their conventional digital counterparts.

EXPERIMENTAL DEMONSTRATION

In order to validate the SHD dynamics theorized above, we carried out an experimental demonstration of the SHD algorithm using metal-oxide based electrochemical devices reported in Sebastian et al. (2017) (**Figure 3A**). These devices are three-terminal[4], voltage-controlled crosspoint elements, absent of any compliance circuits or serial-access devices. The modulation characteristics obtained for one of the devices is shown in **Figure 3B**, where "crossed-swords" behavior is observed with a well-defined symmetry point.

To capture the essence of SHD-based training, we have chosen a 2-parameter optimization problem with a synthetic dataset $x_{1,2}$ and y generated of form $y = G_{0_1}x_1 + G_{0_2}x_2 + \gamma$, where $G_{01,2}$ are the unknowns searched for and γ is the Gaussian noise. During the forward and backward pass cycles, input values (from the training set) were represented with different voltage levels and output results were obtained *via* measuring the line currents. We note that in an actual implementation representing input values with different pulse widths rather than amplitudes might be beneficial, avoiding the impact of the non-linear conductance of the crosspoint elements for accurate vector-matrix multiplication. Following the generation of the update vectors, x and δ, the array is programmed in parallel using stochastic updating with half-bias voltage scheme, as explained in Gokmen and Vlasov (2016). Therefore, we neither computed the outer product explicitly nor accessed the devices serially at any point (**Figure 3C**).

The array training results using the SHD algorithm are shown in **Figure 3D**. It can be seen that both A_1 and A_2 converges to 0, while C_1 and C_2 successfully converge to the optimal values. Moreover, the distinctive spiraling behavior (i.e., decaying oscillations) was observed for both variables, displaying analogous dynamics to dissipative mechanical systems. We found that the success of the training operation strongly depends on the stability of the devices' symmetry points. As discussed earlier, any discrepancy between the symmetry point and the reference point (initialized to the symmetry point at the beginning of training) of a device indicates a non-zero steady-state velocity. Therefore, future crosspoint device technologies should exhibit a well-defined symmetry point that is at least quasi-static throughout the training operation.

[4]SHD algorithm is compatible with various configurations of resistive device, such as 2-terminal devices, as well as 3-terminal devices we show here.

DISCUSSION AND SIMULATED TRAINING RESULTS

In this section, we first discuss how to implement the SHD algorithm with 3 arrays (instead of 4) using the intuition obtained from the theoretical analysis of the coupled-system. Then we provide simulated results for a large-scale neural network for different asymmetry characteristics to benchmark our method against SGD-based training.

Considering a sequence of $m + n$ incremental and n decremental changes at random order, the net modulation obtained for a symmetric device is on average m. On the other hand, we have shown above that for asymmetric devices the conductance value eventually converges to the symmetry point for increasing n (irrespective of m or the initial conductance). It can be seen by inspection that for increasing statistical variation present in the training data (causing more directional changes for updates), the effect of device asymmetry gets further pronounced, leading to heavier degradation of classification accuracy for networks trained with conventional SGD (see **Supplementary Figure S1**). However, this behavior can alternatively be viewed as non-linear filtering, where only signals with persistent sign information, $\frac{m}{m+2n}$, are passed. Indeed, the SHD algorithm exploits this property within the auxiliary array, A, which filters the gradient information that is used to train the core array, C. As a result, C is updated with less frequency and only in directions with a high confidence level of minimizing the error function of the problem at hand. A direct implication of this statement is that the asymmetric modulation behavior of C is much less critical than that of A (see **Supplementary Figure S2**) for successful optimization as its update signal contains less amount of statistical variation. Therefore, symmetry point information of C_{main} is not relevant either. Using these results and intuition, we modified the original algorithm by discarding C_{ref} and using A_{ref} (set to $A_{main,\ symmetry}$) as a common reference array for differential readout. This modification reduces the hardware cost of SHD implementations by 50% to significantly improve their practicality.

Our description of asymmetry as the mechanism of dissipation indicates that it is a necessary and useful device property for convergence within the SHD framework (**Figure 2E**). However, this argument does not imply that the convergence speed would be determined by the magnitude of device asymmetry for practical-sized applications. Unlike the single-parameter regression problem considered above, the exploration space for DNN training is immensely large, causing optimization to take place over many iterations of the dataset. In return, the level of asymmetry required to balance (i.e., damp) the system evolution is very small and can be readily achieved by any practical level of asymmetry.

To prove these assertions, we show simulated results in **Figure 4** for a Long Short-Term Memory (LSTM) network, using device models with increasing levels of asymmetry, trained with both the SGD and SHD algorithms. The network was trained on Leo Tolstoy's War and Peace novel, to predict the next character for a given text string (Karpathy et al., 2015).

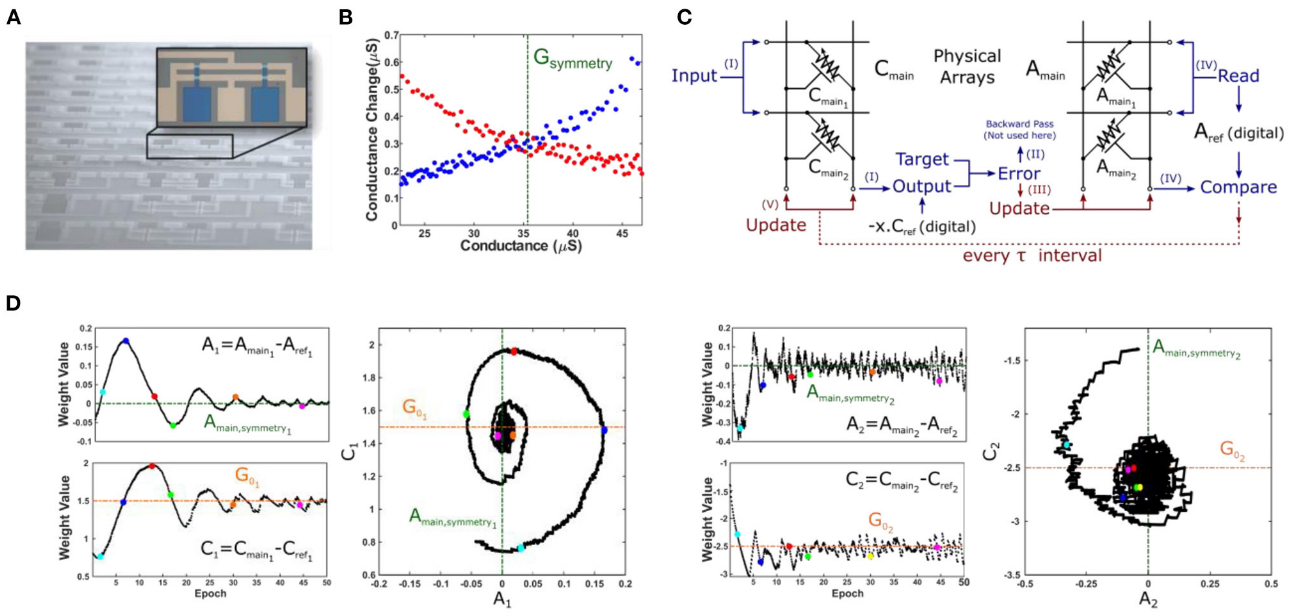

FIGURE 3 | Experimental demonstration of SHD training algorithm. **(A)** Optical micrograph of metal-oxide based electrochemical devices (Sebastian et al., 2017). Note that the image shows an integrated array whereas experiments were conducted with individual devices connected externally. **(B)** Conductance modulation characteristics obtained for one of the devices, showing "crossed-swords" behavior with a well-defined symmetry point. **(C)** Schematic for array configuration used in 2-parameter optimization with SHD algorithm. All steps are shown using the same notation used in **Figure 2** except for the backward pass (Step *II*) which is not required for a single layer network. For training, sum of squared errors is used to calculate the scalar error and vector δ, C_{main} is updated once every 10 samples (i.e., $\tau = 10$) whereas $[1, 0]$ and $[0, 1]$ were used in Step *IV* (as *u* vectors). The reference arrays containing symmetry point information are stored in digital (as they remain unchanged throughout the training) for simplicity. **(D)** Evolution of device conductances for the first (A_1, C_1) and the second (A_2, C_2) parameters. Plotting the values of *A* vs. *C* produces the distinctive spiraling image, as expected from the theoretical analysis.

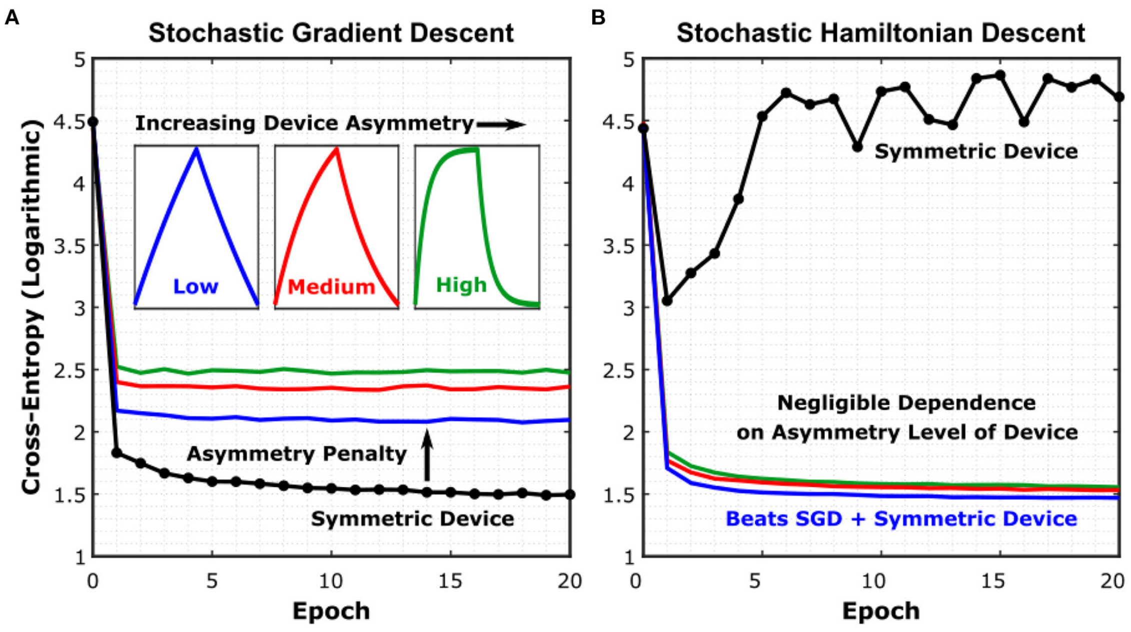

FIGURE 4 | Simulated training results for different resistive device technologies. **(A)** Simulated learning curves of a Long Short-Term Memory (LSTM) network trained on Leo Tolstoy's War and Peace novel, using different crosspoint device models under the SGD algorithm. Details of the network can be found in Karpathy et al. (2015). **(B)** Simulated learning curves for the same network using the SHD algorithm. All simulation details can be found in Section M3. Training Simulator and LSTM Network. See **Supplementary Figure S4** for device-to-device variation included in the simulations and **Supplementary Figure S6** for floating-point baseline comparison.

For reference, training the same network with a 32-bit digital floating-point architecture yields a cross-entropy level of 1.33 (complete learning curve shown in **Supplementary Figure S6**). We have particularly chosen this network as LSTM's are known for being particularly vulnerable to device asymmetry (Gokmen et al., 2018).

The insets in **Figure 4** show the average conductance modulation characteristics representative for each asymmetry level. The simulations further included device-to-device variation, cycle-to-cycle variation, analog read noise, and stochastic updating similar to the work conducted in Gokmen and Vlasov (2016). The learning curves show the evolution of the cross-entropy error, which measures the performance of a classification model, with respect to the epochs of training. First, **Figure 4A** shows that even for minimally asymmetric devices (blue trace) trained with SGD, the penalty in classification performance is already severe. This result also demonstrates once more the difficulty of engineering a device that is symmetric-enough to be trained accurately with SGD. On the other hand, for SHD (**Figure 4B**), all depicted devices are trained successfully, with the sole exception being the perfectly symmetric devices (black trace), as expected (see **Supplementary Figure S3** for devices with abrupt modulation characteristics). Furthermore, **Figure 4B** demonstrates that SHD can even provide training results with higher accuracy and faster convergence than those for perfectly symmetric devices trained with SGD. As a result, we conclude that SHD is generically superior to SGD for analog deep learning architectures.

Finally, although we present SHD in the context of analog computing specifically, it can also be potentially useful on conventional processors (with simulated asymmetry). The filtering dynamics described above allows SHD to guide its core component selectively in directions with high statistical persistence. Therefore, at the expense of increasing the overall memory and number of operations, SHD might outperform conventional training algorithms by providing faster convergence, better classification accuracy, and/or superior generalization performance.

CONCLUSION

In this paper, we described a fully-parallel neural network training algorithm for analog crossbar-based architectures, Stochastic Hamiltonian Descent (SHD), based on resistive devices with asymmetric conductance modulation characteristics, as is the case for all practical technologies. In contrast to previous work that resorted to serial operations to mitigate asymmetry, SHD is a fully-parallel and scalable method that can enable high throughput and energy-efficiency deep learning computations with analog hardware. Our new method uses an auxiliary array to successfully tune the system variables in order to minimize the total energy (Hamiltonian) of the system that includes the effect of device asymmetry. Standard techniques, such as Stochastic Gradient Descent, perform optimization without accounting for the effect of device asymmetry and thus converge to the minimum of a wrong energy function. Therefore, our theoretical framework describes the inherent fundamental incompatibility of asymmetric devices with conventional training algorithms. The SHD framework further enables the exploitation of device asymmetry as a useful feature to selectively filter and apply the updates only in directions with high confidence. The new insights shown here have allowed a 50% reduction in the hardware cost of the algorithm. This method is immediately applicable to a variety of existing device technologies, and complex neural network architectures, enabling the realization of analog training accelerators to tackle the ever-growing computational demand of deep learning applications.

METHODS

M1. Array Initialization (Zero-Shifting)

Initialization of the reference array requires identification of the conductance values of each and every element in A_{main}, and programming the reference array conductances (A_{ref}) to those values. Given that under those conditions $A = A_{main} - A_{ref}$ becomes 0, the method is also referred to as zero-shifting (Kim et al., 2019a). To identify $A_{main,\ symmetry}$, a sufficiently long sequence of increment-decrement pulses is applied to A_{main}. Given the asymmetric nature of the devices, each pair results in a residual conductance modulation toward each device's respective symmetry point. Following this step, the resultant $A_{main} = A_{main,\ symmetry}$ is then copied to the reference array. Since these steps only occur once per training, the time and energy costs are negligible with respect to the rest of the operation (even for serial copying).

M2. Pseudocode for SHD Algorithm

```
Initialize
    $k: iteration\ step \leftarrow 1, l: layer\ index$
    Set $\tau, \eta$
    For each layer
    $A^l_{main}[k] = A^l_{main,symmetry}$ $(m \times n\ matrix,\ dynamic)$
    $A^l_{ref} = A^l_{main,symmetry}$ $(m \times n\ matrix,\ static)$
    $C^l_{main}[k] = C^l_{main,symmetry}$ $(m \times n\ matrix,\ dynamic)$
For each labeled data pair $[x_i,\ t_i]$
        Convert input $x_i$ to time encoded voltage pulse for the first layer $(x^1_i)$
        $MAC\ O^1[k] = x^1_i[k].[C^1_{main}[k] - A^1_{ref}]$
        Convert analog output to digital to store and apply non-linear functions (activations, pooling etc.)
        Forward propagate $O^1[k]$ as the input for next layer (always using $C^l_{main}$ arrays) for all layers
        Compute error (cost) using network output $O^{final}[k]$ and target output $t[k]$
        Backward propagate using the same dynamics (again using $C^l_{main}$ arrays) to compute all error matrices $\delta^l$[k]
        Update $A^l_{main}[k+1] \leftarrow A^l_{main}[k] - \eta.x^l[k] \otimes \delta^l[k]$ using stochastic update scheme
        If mod (k,$\tau$) = 0
            $u^l[k] = [0, 0, 0 \ldots 1, \ldots 0, 0]$, where "1" is at $k^{th}$ location
            MAC $v^l[k] = u^l[k].[A^l[k] - A^l_{ref}]$
```

$$\text{Update } C^{l}_{main}\left[k+1\right] \leftarrow C^{l}_{main}\left[k\right] - \eta . u^{l}[k] \bigotimes v^{l}[\,k]$$

M3. Training Simulator and LSTM Network

The simulation framework used here is the same that was used in Gokmen and Vlasov (2016), Gokmen et al. (2017, 2018), and Gokmen and Haensch (2020). The simulations start with instantiating 3 devices per weight. Each device parameter (e.g., number of states, asymmetry factor, and symmetry point) is generated with a given mean and standard variation, such that no two devices are the same. Moreover, these device parameters also bear cycle-to-cycle variation, defined by another parameter, to make the operation more realistic. An open access version of the simulator we used in this work can be found in github.com/ibm/aihwkit for reproduction of the results.

The incremental changes are set such that devices have on average 1,200 programmable states within their dynamic range. Through setting the gain factors at the integrator terminals appropriately, the average full conductance range of devices are adjusted to be equivalent $\pm$ 2 arbitrary units. Consistent with this notation, the integrators are set to saturate at $\pm$ 40 arbitrary units. We have used 9-bit resolution for the ADCs and 7-bit resolution for the DACs where the output-referred noise level was set at 0.02 arbitrary units. This selection was made in order not to be limited by noise-related performance degradation, as studied by Gokmen et al. (2017). In the update cycle, the maximum allowed number of pulses (i.e., bit length, BL) was set to be 100. However, as update management determines this number on-the-go depending on certain characteristics of the update vectors and device parameters, real BL was $<$10 for the most of the training.

The War and Peace dataset consists of $3,258,246$ characters, which we split into training and test sets as $2,933,246$ and $325,000$ characters, respectively. The network is trained to have a vocabulary of 87 distinct characters. We have selected to use hidden vectors of 64-cell size, which corresponds to $\sim$77K weights for the complete network. Full details of the network architecture can be found in Karpathy et al. (2015).

The selection of the LSTM problem studied here in detail is found to be optimal, which is complex enough to validate the training algorithm, while it still is trainable with limited number of conductance states, analog noise, variations, and limited resolution. Given that SHD only resolves asymmetry related issues, whereas other imperfections related with analog processors such as device-to-device variability, cycle-to-cycle variability, noise, and resolution can still deteriorate the training performance significantly, we recommend future studies to explore larger problems, once there are additional solutions for these other non-idealities related to analog crossbar architectures.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/**Supplementary Material**, further inquiries can be directed to the corresponding author/s.

AUTHOR CONTRIBUTIONS

MO and TG conceived the original idea and performed software experiments. TT fabricated devices. MO and SK performed hardware experiments. All authors contributed to the theory development and contributed to the preparation of the manuscript. All authors contributed to the article and approved the submitted version.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/frai.2022.891624/full#supplementary-material

REFERENCES

Agarwal, S., Gedrim, R. B. J., Hsia, A. H., Hughart, D. R., Fuller, E. J., Talin, A. A., et al. (2017). Achieving ideal accuracies in analog neuromorphic computing using periodic carry. *Symp. VLSI Technol.* 174–175. doi: 10.23919/VLSIT.2017.7998164

Agarwal, S., Plimpton, S. J., Hughart, D. R., Hsia, A. H., Richter, I., Cox, J. A., et al. (2016). Resistive memory device requirements for a neural algorithm accelerator. *Proc. Int. Jt. Conf. Neural Networks.* 929–938. doi: 10.1109/IJCNN.2016.7727298

Ambrogio, S., Narayanan, P., Tsai, H., Shelby, R. M., Boybat, I., Nolfo, C., et al. (2018). Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature.* 558, 60–67. doi: 10.1038/s41586-018-0180-5

Burr, G. W., Shelby, R. M., Sebastian, A., Kim, S., Sidler, S., Virwani, K., et al. (2017). Neuromorphic computing using non-volatile memory. *Adv. Phys.* 2, 89–124. doi: 10.1080/23746149.2016.1259585

Burr, G. W., Shelby, R. M., Sidler, S., Di Nolfo, C., Jang, J., Boybat, I., et al. (2015). Experimental demonstration and tolerancing of a large-scale neural network (165 000 Synapses) using phase-change memory as the synaptic weight element. *IEEE Trans. Electron Devices.* 62, 3498–3507. doi: 10.1109/TED.2015.2439635

Cai, F., Correll, J. M., Lee, S. H., Lim, Y., Bothra, V., Zhang, Z., et al. (2019). A fully integrated reprogrammable memristor– CMOS system for efficient multiply–accumulate operations. *Nat. Electron.* 2, 1. doi: 10.1038/s41928-019-0270-x

Cauchy, A. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris.* 25, 536–538.

Chen, Y., Member, S., Krishna, T., Emer, J. S., and Sze, V. (2016). Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid-State Circuits.* 52, 127–138. doi: 10.1109/JSSC.2016.2616357

Choi, J., Venkataramani, S., Srinivasan, V., Gopalakrishnan, K., Wang, Z., and Chuang, P. (2019). Accurate and efficient 2-bit quantized neural networks. *Proc. 2nd SysML Conf.* 348–359.

Feng, Y., and Tu, Y. (2023). *How Neural Networks Find Generalizable Solutions: Self-Tuned Annealing in Deep Learning.* Available online at: https://arxiv.org/abs/2001.01678 (accessed March 01, 2022).

Fuller, E. J., Keene, S. T., Melianas, A., Wang, Z., Agarwal, S., Li, Y., et al. (2019). Parallel programming of an ionic floating-gate memory array for scalable neuromorphic computing. *Science* 364, 570–574. doi: 10.1126/science.aaw5581

Gokmen, T., and Haensch, W. (2020). Algorithm for training neural networks on resistive device arrays. *Front. Neurosci.* 14, e00103. doi: 10.3389/fnins.2020.00103

Gokmen, T., Onen, M., and Haensch, W. (2017). Training deep convolutional neural networks with resistive cross-point devices. *Front. Neurosci.* 11, 538. doi: 10.3389/fnins.2017.00538

Gokmen, T., Rasch, M. J., and Haensch, W. (2018). Training LSTM networks with resistive cross-point devices. *Front. Neurosci.* 12, 745. doi: 10.3389/fnins.2018.00745

Gokmen, T., and Vlasov, Y. (2016). Acceleration of deep neural network training with resistive cross-point devices: design considerations. *Front. Neurosci.* 10, 333. doi: 10.3389/fnins.2016.00333

Grollier, J., Querlioz, D., Camsari, K. Y., Everschor-Sitte, K., Fukami, S., and Stiles, M. D. (2020). Neuromorphic spintronics. *Nat. Electron.* 3, 360–370. doi: 10.1038/s41928-019-0360-9

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., et al. (2017). In - datacenter performance analysis of a tensor processing unit. *Proc. 44th Annu. Int. Symp. Comput. Archit.* 1–12. doi: 10.1145/3079856.3080246

Karpathy, A., Johnson, J., and Fei-Fei, L. (2015). "Visualizing and understanding recurrent networks", in *ICLR* 2016 (San Juan), 1–12.

Kim, H., Rasch, M., Gokmen, T., Ando, T., Miyazoe, H., Kim, J.-J., et al. (2020). *Zero-Shifting Technique for Deep Neural Network Training on Resistive Cross-point Arrays*. Available online at: https://arxiv.org/abs/1907.10228 (accessed March 01, 2022).

Kim, H., Rasch, M., Gokmen, T., Ando, T., Miyazoe, H., Kim, J. J., et al. (2019a). Zero-shifting Technique for deep neural network training on resistive cross-point arrays. *arXiv* 2019–2022.

Kim, S., Todorov, T., Onen, M., Gokmen, T., Bishop, D., Solomon, P., et al. (2019b). Oxide based, CMOS-compatible ECRAM for deep learning accelerator. *IEEE Int. Electron Devices Meet.* 847–850. doi: 10.1109/IEDM19573.2019.8993463

Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*. 521, 436–444. doi: 10.1038/nature14539

Li, C., Hu, M., Li, Y., Jiang, H., Ge, N., Montgomery, E., et al. (2018). Analogue signal and image processing with large memristor crossbars. *Nat. Electron.* 1, 52–59. doi: 10.1038/s41928-017-0002-z

Li, C., Wang, Z., Rao, M., Belkin, D., Song, W., Jiang, H., et al. (2019). Long short-term memory networks in memristor crossbar arrays. *Nat. Mach. Intell.* 1, 49–57. doi: 10.1038/s42256-018-0001-4

Prezioso, M., Merrikh-Bayat, F., Hoskins, B. D., Adam, G. C., Likharev, K. K., and Strukov, D. B. (2015). Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*. 521, 61–64. doi: 10.1038/nature14441

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. (2020). *Zero: Memory Optimizations Toward Training Trillion Parameter Models*. Atlanta, GA: IEEE Press.

Rumelhart, D. E., Hinton, G. E., and Willams, R. J. (1986). Learning representations by back-propagating errors. *Nature*. 323, 533–536. doi: 10.1038/323533a0

Salakhutdinov, R., and Hinton, G. (2009). Deep Boltzmann machines. *J. Mach. Learn. Res.* 5, 448–455.

Scellier, B., and Bengio, Y. (2017). Equilibrium propagation: bridging the gap between energy-based models and backpropagation. *Front. Comput. Neurosci.* 11, e00024. doi: 10.3389/fncom.2017.00024

Sebastian, A., Le Gallo, M., Khaddam-Aljameh, R., and Eleftheriou, E. (2020). Memory devices and applications for in-memory computing. *Nat. Nanotechnol.* 15, 246–253. doi: 10.1038/s41565-020-0655-z

Sebastian, A., Tuma, T., Papandreou, N., Le Gallo, M., Kull, L., Parnell, T., et al. (2017). Temporal correlation detection using computational phase-change memory. *Nat. Commun.* 8. 1–10. doi: 10.1038/s41467-017-01481-9

Steinbuch, K. (1961). Die lernmatrix. *Kybernetik*. 1, 36–45. doi: 10.1007/BF00293853

Strubell, E., Ganesh, A., and McCallum, A. (2020). Energy and policy considerations for deep learning in NLP. *ACL 2019 - 57th Annu. Meet. Assoc. Comput. Linguist. Proc. Conf.* 3645–3650. doi: 10.18653/v1/P19-1355

Sun, X., Choi, J., Chen, C.-Y., Wang, N., Venkataramani, S., Srinivasan, V., et al. (2019). Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks. *Adv. Neural Inf. Process. Syst.*

Woo, J., and Yu, S. (2018). Resistive memory-based analog synapse: the pursuit for linear and symmetric weight update. *IEEE Nanotechnol. Mag.* 12, 36–44. doi: 10.1109/MNANO.2018.2844902

Yao, X., Klyukin, K., Lu, W., Onen, M., Ryu, S., Kim, D., et al. (2020). Protonic solid-state electrochemical synapse for physical neural networks. *Nat. Commun.* 11, 1–10. doi: 10.1038/s41467-020-16866-6

Yu, S., Chen, P. Y., Cao, Y., Xia, L., Wang, Y., and Wu, H. (2015). Scaling-up resistive synaptic arrays for neuro-inspired architecture: challenges and prospect. *Tech. Dig. - Int. Electron Devices Meet. IEDM*. 17–3. doi: 10.1109/IEDM.2015.7409718

Conflict of Interest: MO, TG, TT, TN, JR, WH, and SK were employed by IBM Thomas J. Watson Research Center.

The remaining author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.