frontiers in Neuroscience

ORIGINAL RESEARCH
published: 06 August 2019
doi: 10.3389/fnins.2019.00793

Streaming Batch Eigenupdates for Hardware Neural Networks

Brian D. Hoskins[1], Matthew W. Daniels[1], Siyuan Huang[2], Advait Madhavan[1,3], Gina C. Adam[2], Nikolai Zhitenev[1], Jabez J. McClelland[1] and Mark D. Stiles[1]*

[1] Physical Measurement Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, United States, [2] Electrical and Computer Engineering, George Washington University, Washington, DC, United States, [3] Institute for Research in Electronics and Applied Physics, University of Maryland, College Park, MD, United States

Neural networks based on nanodevices, such as metal oxide memristors, phase change memories, and flash memory cells, have generated considerable interest for their increased energy efficiency and density in comparison to graphics processing units (GPUs) and central processing units (CPUs). Though immense acceleration of the training process can be achieved by leveraging the fact that the time complexity of training does not scale with the network size, it is limited by the space complexity of stochastic gradient descent, which grows quadratically. The main objective of this work is to reduce this space complexity by using low-rank approximations of stochastic gradient descent. This low spatial complexity combined with streaming methods allows for significant reductions in memory and compute overhead, opening the door for improvements in area, time and energy efficiency of training. We refer to this algorithm and architecture to implement it as the streaming batch eigenupdate (SBE) approach.

Keywords: neuromorphic, memristor, network training, stochastic gradient descent, back propagation, singular value decomposition

OPEN ACCESS

Edited by:
Anton Civit,
University of Seville, Spain

Reviewed by:
Tayfun Gokmen,
IBM T. J. Watson Research Center, United States
Paul Honeine,
EA4108 Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes (LITIS), France

****Correspondence:***
Brian D. Hoskins
brian.hoskins@nist.gov

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *04 April 2019*
Accepted: *15 July 2019*
Published: *06 August 2019*

Citation:
Hoskins BD, Daniels MW, Huang S, Madhavan A, Adam GC, Zhitenev N, McClelland JJ and Stiles MD (2019) Streaming Batch Eigenupdates for Hardware Neural Networks. Front. Neurosci. 13:793. doi: 10.3389/fnins.2019.00793

INTRODUCTION

Deep neural networks (DNNs) have grown increasingly popular over the years in a wide range of fields from image recognition to natural language processing. These systems have enormous computational overhead, particularly on multiply and accumulate (MAC) operations, and specialized hardware has been developed to accelerate these tasks. As the networks are themselves tolerant to noise and low precision computing (4-bit and below), theoretical and experimental investigations have shown that analog implementations of DNNs using Ohm's and Kirchoff's laws to perform MAC operations can vastly accelerate the training and reduce the energy of inference by orders of magnitude.

Investigations regarding an appropriate nanodevice suitable for analog inference have focused on different families of 2-terminal memory devices (memristors, resistive random-access memory (ReRAM), phase change memories (PCM), etc.) as well as 3 terminal devices (flash memory, lithium insertion) (Haensch et al., 2019). These devices have the desirable properties of analog tunability, high endurance, and long-term memory needed for use in embedded inference applications. Applications based on these devices perform well when used for inference and have been well-studied, with intermediate scale systems having been built by integrating devices into crossbar arrays (Prezioso et al., 2015; Adam et al., 2017; Chakrabarti et al., 2017; Wang et al., 2018).

Though most of the effort has been focused on building inference engines, more recent work has begun to address difficulties in training such nanodevice arrays (Adam, 2018; Ambrogio et al., 2018). In crossbar architectures, there are two approaches to updating the weights. The first, which fits well with weights computed in software, is to either column wise or sequentially update each weight separately. The other, called an outer product update, is to update all the weights simultaneously with two vectors of voltages or voltage pulses. This latter approach is limited in the type of updates that can be applied, but its speed and energy advantage essentially preclude the use of the former in network training applications. The goal of the work presented here is to develop a technique based on outer product updates that approaches the training fidelity available for algorithms based on sequential weight updates, which are often employed in software-based platforms.

We focus on backpropagation-based learning in a layer in a deep neural network where the weights for that layer are stored in a memristor crossbar array. In the forward pass, the layer receives a series of input vectors $\mathbf{x}^j$, indexed by j, from the previous layer and multiplies it by the weight matrix $\mathbf{w}^j$. The resulting vector goes through a non-linear activation function and is passed to the next layer. In the backward pass, a similar operation occurs, except in the opposite direction, where the error from each consequent layer ($\boldsymbol{\delta}^j$) is pushed back into the crossbar, to compute the error for the previous layer with respect to each of the weights ($\mathbf{w}^j$). In the weight update step, learning proceeds by calculating the gradient of the loss function with respect to the weights in the layer of interest, by using input values, $\mathbf{x}^j$, and the error, $\boldsymbol{\delta}^j$, received from the forward and backward passes respectively. This gradient of the loss function is given by $-\boldsymbol{\delta}^j[\mathbf{x}^j]^T$. For a suitably parallelized architecture, the number of clock cycles needed for these operations is independent of the size of the memory arrays in each layer (Gokmen and Vlasov, 2016). This scaling is maintained for the weight matrix updates by using stochastic gradient descent (SGD), which uses outer product updates alone. In this case $\Delta\hat{\mathbf{w}}^j = -\eta\boldsymbol{\delta}^j[\mathbf{x}^j]^T$, with η the learning rate, and the outer product update is implemented by applying voltages representing $\mathbf{x}^j$ to one side of the crossbar and voltages representing $\boldsymbol{\delta}^j$ to the other. Independently updating each element would require a series of ab updates, where the inputs have length a and the errors have length b.

Though SGD is a powerful method for training, other methods, employed in software, such as momentum, Adagrad, or, most simply, batch update can sometimes be superior. However, these require additional memory overhead or explicit look-a-head updates of the memory which at the present time render them impracticable in the crossbar training setting (Gokmen et al., 2018).

The advantage of SGD is that it uses only outer product updates but it has the disadvantage that it requires an update of the weights for each input. A modification of this approach, minibatch gradient descent (MBGD), stores the n inputs $\mathbf{x}^{i,j}$ and the errors $\boldsymbol{\delta}^{i,j}$ for batch i and then uses them to sequentially update the weight matrix in the same way as done for SGD. On advanced processors, like graphical processing units (GPUs and TPUs), MBGD can take advantage of pipelining to significantly speed up the processing (Jouppi et al., 2017). If the n inputs and errors are stored separately, and then applied to a crossbar array with n outer-product updates, a strategy we call MBGD1, there is no apparent advantage to doing MBGD. However, a modification of MBGD1, we call MBGD2, in which the weight updates for each input are summed before they are used to update the cross-bar array, can reduce the number of times each device is written to, during the training process, hence reducing the stress on each device. Here, the weight matrix update matrix for batch

$$\Delta\hat{\mathbf{w}}^i = -\frac{\eta}{n}\sum_{j=1}^{n}\boldsymbol{\delta}^{i,j}[\mathbf{x}^{i,j}]^T,$$

is summed over all members of the batch, indexed by j and then each element of the cross-bar array is individually updated. MBGD1 uses n outer product updates, where n is the number of inputs in the minibatch, and each element of the cross-bar array is accessed n times. MBGD2, only updates each device once per batch, but requires $\min(a, b)$ operations if the updates are performed column wise or ab operations for sequential.

This latter form of mini-batch gradient descent (MBGD2) which is the focus of this work is of extreme interest for the case of nanodevice arrays. It has been suggested that it can increase tolerance with respect to device non-idealities, as well as be employed to minimize the number of device updates, which can be a problem in systems with low endurance or high energy to program (Kataeva et al., 2015). In PCM arrays, minimizing the number of updates is critical to preventing a hard reset when the device reaches its natural conductance limit (Boybat et al., 2017). Additionally, in cases where the energy of inference is significantly less than the energy of update, reducing the number of updates could result in a substantial decrease in the energy required to train the network, even if it occurs at the expense of training time. Given these advantages of mini-batch update, a dual memory approach, with trench capacitor based short term memory cells that linearly perform batch summation of the calculated gradients, have also been proposed (Kim et al., 2017; Ambrogio et al., 2018; Li et al., 2018). However, this update matrix is no longer in a form that is readily amenable to outer product updates so additional modifications are needed.

One straightforward way to convert the batch weight update to outer product form is to compute its singular value decomposition.

$$\Delta\hat{\mathbf{w}}^i = \sum_{j=1}^{\mathrm{k=min}(a,b)}\sigma^j\mathbf{u}^j[\mathbf{v}^j]^T.$$

In addition to allowing for rank one updates of the weight matrix, the update can be made to carry even more information about the weight update, by performing the update with a low rank approximation, that is, taking $k < min(a, b)$. We show in section Proposed methods for training and new algorithm that for this approach, taking a small number ranks which include the most important terms in this sum can provide a very good approximation with significant time savings. Unfortunately,

TABLE 1 | Comparison of asymptotic scaling for common methods of $a \times b$ crossbar training.

Training method	Update method	Updates per device	Crossbar primitive operations	Memory requirements	External computational load per datum
SGD	Outer product	nm	nm	$O\,(a+b)$	0
MBGD1	Outer Product	nm	nm	$O\,(na+nb)$	0
MBGD2	Sequential or column-wise	m	abm or ηm	$O\,(ab)$	$O(ab)$
Rank k SVD	Outer Product	km	km	$O\,(ab)$	$O\,(4\eta\mu^2+22\eta^3)$
Rank k SBE	Outer product	km	km	$O\,(ka+kb)$	$O\,(gka+gkb)$

We define m as the total number of minibatches, n as the amount of training data per minibatch, and k is the decomposition rank when appropriate. For the SVD method, we take scaling laws for the R-SVD algorithm, choosing $\mu = max(a, b)$ and $\eta = min(a, b)$ to get the best scaling (Golub and Van Loan, 2013). For the SBE approach, the complexity scales linearly with dimension and rank (Hua et al., 1999). The exact choice of algorithm will determine the value of the constant g. For the PASTd algorithm, g = 4 (Yang, 1995).

computing the singular value decomposition requires storing the entire update matrix and requires number of operations that scales like $\left[\min(a,b)\right]^3$. This would require significant overhead, most likely in the form of digital computation external to the crossbar, though analog embodiments could be implemented. To get the benefits of turning the batch into the smallest number of updates, an efficient means of calculation is needed.

Here, we combine the batch-update approach with a streaming method for computing low rank approximations to the singular value decomposition due to Oja (1982). This novel combination, which we refer to as streaming batch eigenupdate (SBE) computes the contribution of each input and error vector pair to the update as they arrive requires no further information about them for subsequent processing. This approach minimizes storage and computational requirements while preserving the advantages of batch updates. Such an approach is enabled by our finding that a low rank approximation works well for training a network. **Table 1** compares the necessary storage, computational overhead, and number of crossbar updates required for each of the approaches mentioned above. Section Materials and Methods describes the procedure for streaming batch eigenupdate. Section Results compares the performance of each of these approaches through simulations of a four-layer neural network and section Discussion discusses some of the remaining issues with implementation of this approach.

MATERIALS AND METHODS

Proposed Methods for Training and New Algorithm

The key idea behind our alternative approach is to estimate the most representative single outer product update for the batch. Not only is this approach fast, it also minimizes the amount of information that needs to be stored to make each update. We consider then, an arbitrary network layer being trained on batch i with an $a \times b$ weight matrix $\mathbf{w}^i$. The layer receives j activations $\mathbf{x}^{i,j}$ of dimension b and backpropagated errors $\boldsymbol{\delta}^{i,j}$ of dimension a per batch. In the ideal case, we would like the network to update according to

$$\mathbf{w}^{i+1} = \mathbf{w}^i + \Delta\hat{\mathbf{w}}^i,$$

where the batch average update $\Delta\hat{\mathbf{w}}^i$ is a sum of outer products,

$$\Delta\hat{\mathbf{w}}^i = -\frac{\eta}{n}\sum_{j=1}^{n}\boldsymbol{\delta}^{i,j}[\mathbf{x}^{i,j}]^T.$$

Each term in this sum is the gradient of the loss function of that input $\mathbf{x}^{i,j}$, which is a rank 1 matrix. The sum of the gradients, $\Delta\hat{\mathbf{w}}^i$, is the gradient of the batch loss function and is in general a rank $\min(n, a, b)$ matrix. Performing such an update with conventional SGD will require n outer product operations. An important observation here is that the outer product operation itself is a rank 1 operation, and hence an efficient alternative would entail using a low rank matrix approximation of $\Delta\hat{w}^i$ to reduce the total number of updates performed. More specifically, we perform $k < n$ outer product updates where k is the number of significant singular values of the true gradient $\Delta\mathbf{w}^i$.

The singular value decomposition (SVD) of $\Delta\mathbf{w}^i$ entails significant memory overhead and computational cost. We base our approach on streaming principal component analysis (PCA) developed to compute the k most significant singular vectors of a streaming covariance matrix (Balsubramani et al., 2013; Mitliagkas et al., 2013; Li et al., 2016; Yang et al., 2018), which is based on Oja's original algorithm for PCA that was developed to describe the evolution of neural network weights (Oja, 1982). By applying his formalism here on the weight *updates*, we can extract, on the fly, a set of k most representative vectors, of the original rank n update. This allows us to perform memory limited batch updates with $k(a + b)$ additional memory units instead of ab as used in previous studies. Oja's rule is sometimes considered a form of unsupervised learning, and employing it amounts to using a separate unsupervised neural network to train the network of interest. However, this network trains on the batch gradient and only needs very short-term memory as the gradient is constantly changing. Such short term memory arrays are already entering into use (Kim et al., 2017; Ambrogio et al., 2018).

If we consider the special case of $k = 1$, using only the single most important singular value, we can define an approximation for $\boldsymbol{\Delta}\mathbf{w}^i$, $\boldsymbol{\Delta}\hat{\mathbf{W}}^i$, in terms of left and right singular unit vectors $\mathbf{X}^i$ and $\boldsymbol{\Delta}^i$ corresponding to that largest singular value, σ^i. The rank

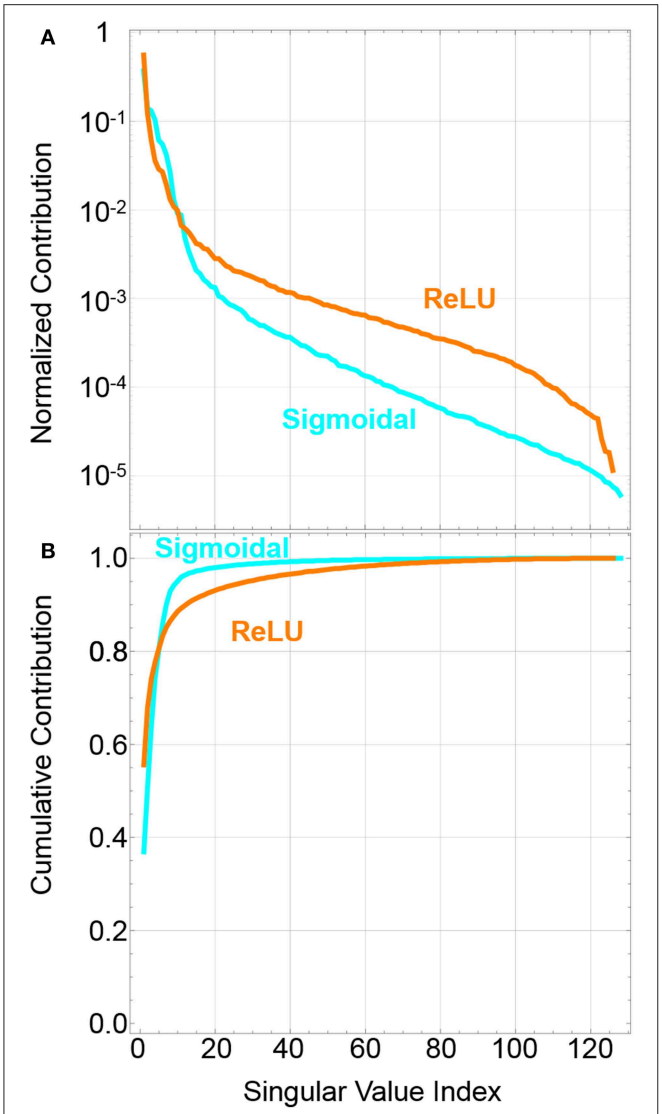

FIGURE 1 | (A) Example of the contribution of the normalized singular values to the batch update for the middle layer of a 728 × 256 × 128 × 10 network trained for MNIST with ReLU and sigmoidal activation. The batch size is 10,000. **(B)** Cumulative sum of the contribution of the first k singular values. The sum of the first few vectors approaches the total sum, one, showing that they contain most of the batch information.

1 approximation, which we call the principal eigenupdate of the batch, is then:

$$\Delta\hat{\mathbf{W}}^i \approx -\eta\sigma^i\Delta^i[\mathbf{X}^i]^T. \tag{1}$$

This represents the single best rank 1 approximation of the batch update, with η the traditional learning rate. These values can be estimated over a streaming batch of size n such that $\mathbf{X}^i \approx \frac{\mathbf{X}^{i,n}}{\|\mathbf{X}^{i,n}\|}$, $\Delta^i \approx \frac{\Delta^{i,n}}{\|\Delta^{i,n}\|}$, and $\sigma^i \approx \sigma^{i,n}$ using the following streaming update rules where j runs from 1 to n:

$$\mathbf{X}^{i,j+1} = \frac{j}{j+1}\mathbf{X}^{i,j} + \frac{1}{j+1}\mathbf{x}^{i,j}\frac{(\boldsymbol{\delta}^{i,j}\cdot\Delta^{i,j})}{\|\Delta^{i,j}\|}$$

$$\Delta^{i,j+1} = \frac{j}{j+1}\Delta^{i,j} + \frac{1}{j+1}\boldsymbol{\delta}^{i,j}\frac{(\mathbf{x}^{i,j}\cdot\mathbf{X}^{i,j+1})}{\|\mathbf{X}^{i,j+1}\|}$$

$$\sigma^{i,j+1} = \frac{j}{j+1}\sigma^{i,j} + \frac{1}{j+1}\frac{(\mathbf{x}^{i,j}\cdot\mathbf{X}^{i,j+1})}{\|\mathbf{X}^{i,j+1}\|}\frac{(\boldsymbol{\delta}^{i,j}\cdot\Delta^{i,j+1})}{\|\Delta^{i,j+1}\|}$$

Afterwards the weight matrix is updated with an outer product update using these rank 1 estimators of the singular values. The next batch is calculated from the end condition of the previous batch such that $\mathbf{X}^{i+1,1} = \mathbf{X}^{i,n}$, $\Delta^{i+1,1} = \Delta^{i,n}$, and $\sigma^{i+1,1} = \sigma^{i,n}$. The previous best estimate is presumed to approximate the subsequent best estimate, which is true if the learning rate is sufficiently small[1].

This algorithm falls within a general family of noisy power iterations (Hardt and Price, 2014), or power iterations performed on stochastic matrices, which are known to extract the eigenvectors of covariance matrixes. It is, additionally, a bi-iterative method for calculating both left and right eigenvectors (Clint and Jennings, 1971; Strobach, 1997).

Intuitively however, the algorithm can be interpreted as updating the "weighted average" activation and error based on the cross significance of its companion term. For example, the estimated left eigenvector, the "estimated activation," of the layer, $\mathbf{X}^{i,j}$, is modified significantly by $\mathbf{x}^{i,j}$ whenever the associated error is both large and directed along the estimated right eigenvector as measured by the factor $\frac{(\boldsymbol{\delta}^{i,j}\cdot\Delta^{i,j})}{\|\Delta^{i,j}\|}$. If the error of an input is small or pointing in an uncommon direction, the estimated activation does not change significantly. The same is true for the dependence of the changes in $\Delta^{i,j+1}$ on $\frac{(\mathbf{x}^{i,j}\cdot\mathbf{X}^{i,j+1})}{\|\mathbf{X}^{i,j+1}\|}$. This algorithm, in the context of using streaming data to estimate the important singular values of the SVD of a batch update matrix, we call this the streaming batch eigenupdate (SBE) algorithm.

This approach can be generalized to an arbitrary number of ranks using different approaches. The simplest and most pipelineable approach is deflation (Hyvärinen and Oja, 2000). During deflation, the first principal component is subtracted from the incoming data stream and the process for finding the subsequent principal component is identical to the one that came before. This would most easily allow for multiple ranks to be calculated without additional delay by simultaneously sending deflated data to lower ranks during the calculation of the principal ranks. Alternative approaches, based on QR factorization or Oja Subspace Networks, can involve feedback between ranks, reducing the number of iterations to achieve convergence but potentially increasing latency and the computational overhead (Oja, 1992; Allen-Zhu and Li, 2017).

[1]Sporadically, one or the other singular vector becomes anti-parallel to the calculated true vector, causing either the other vector to also become anti-parallel or for the singular value, $\sigma^{i,j}$, to become negative. These erroneous sign changes always occur in pairs during the calculation of the approximate eigenupdate, so that the net sign is always correct.

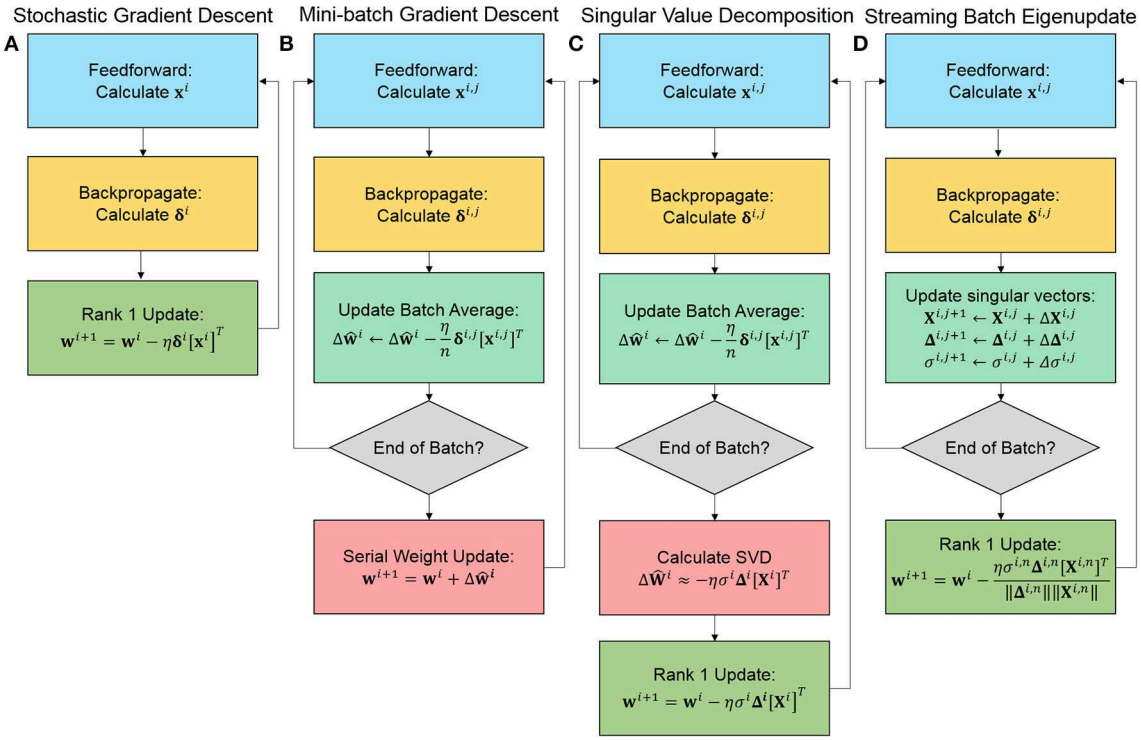

FIGURE 2 | Simplified comparison of the training algorithms for **(A)** stochastic gradient descent (SGD), **(B)** mini-batch gradient descent (MBGD), **(C)** the singular value decomposition (SVD) approximation of the batch, and **(D)** streaming batch eigenupdates (SBE). Both SGD and SBE are rank 1 and calculated on the fly, achieving the highest degree of acceleration.

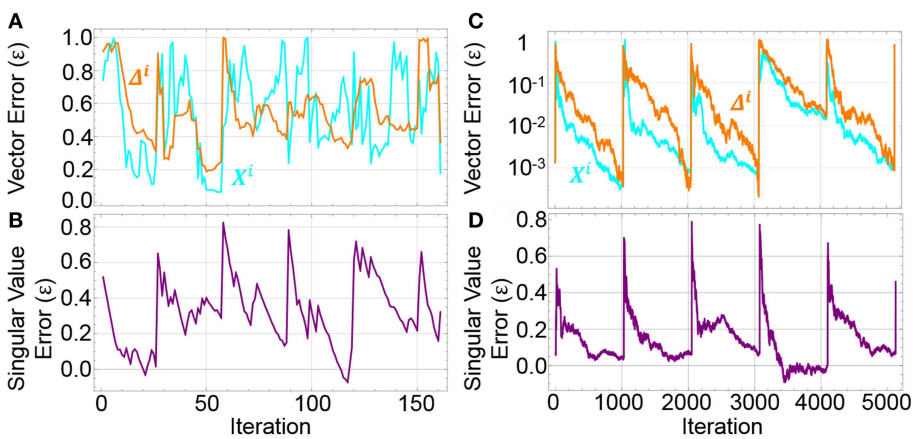

FIGURE 3 | Difference between SBE values and the full SVD values for **(A,C)** singular vectors $\mathbf{X}^i$ and $\boldsymbol{\delta}^i$ as calculated by $\varepsilon = 1 - abs(\frac{(\mathbf{X}^i \cdot \mathbf{X}^{i,j})}{\|\mathbf{X}^{i,j}\|})$ and **(B,D)** singular values as calculated by $\varepsilon = 1 - abs(\frac{\sigma^i}{\sigma^{i,j}})^1$. Batch sizes are 32 **(A,B)** and 1024 **(C,D)**. The larger batches show greater fidelity with more iterations. The sharp increases in the difference correspond to the update of the weight matrix and subsequent change in the gradient.

An important feature of this approach is that it opens a tradeoff space between the software and the hardware. On one hand, it necessarily throws away a significant amount of the information from the batch, which results in a low rank approximation. Hence, for updates with higher rank, larger eigenvalue matrixes would be less well-represented and therefore take a longer time to converge. On the other hand, this approximation, which is a form of compression, allows for a much more compact representation of the error, which has the potential to dramatically reduce hardware costs. One point to

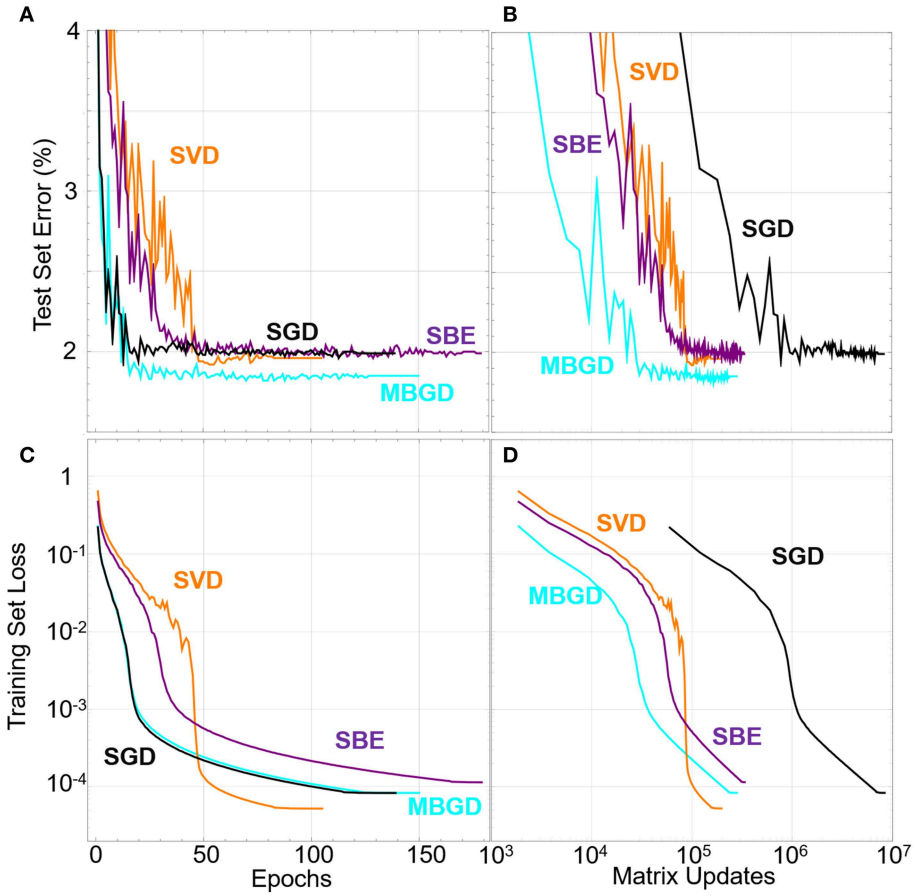

FIGURE 4 | Test set error rate vs. **(A)** the epochs and **(B)** the matrix updates. Training set loss functions under different SGD and batch learning rules (batch size is 32) vs. the number of **(C)** epochs and **(D)** matrix updates. The SVD and SBE algorithms required more epochs to train but fewer matrix updates.

note here is that the smaller the rank of the weight update, the more representative a low rank approximation would be. Consequently, we might expect the eigenupdate to perform better for activation functions that lead to sparse updates, such as for rectifying activation functions like rectified linear units (ReLU).

Figure 1 shows an example of the potential effectiveness of our approach prior to running network models. It shows the relative significance of different singular values, subject to the normalizing condition $\sum_{p=1}^{r} \sigma_p = 1$ for singular index value p up to rank r. The plots show a representative matrix decomposition for a particular batch update in the middle of a conventional $728 \times 256 \times 32 \times 10$ network trained on MNIST to 90% accuracy for the test set. Based on the relative magnitude of values for our example batch, ReLU activations can have as much as 60% of the batch update information contained in the first pair of singular vectors. From the cumulative contribution, we can see for sigmoidal activation, which squashes the outputs of the neurons, the first 10 pairs of singular vectors can capture as much as 95% of the information contained within our example batch. We attribute this fact ultimately to the observation that despite the large sizes of matrices in these networks, the complexity of the trajectories will ultimately be limited by the significantly smaller number of classes which are used to train the networks. Others have observed that the trajectories of networks during training can be reduced to a smaller number of principal components (Lorch, 2016; Antognini and Sohl-Dickstein, 2018).

Network Modeling and Experiments

For our experiments, we compare traditional approaches, stochastic gradient descent (SGD) and mini-batch gradient descent (MBGD), with our PCA based approaches, specifically doing the singular value decomposition (SVD) and the streaming batch eigenupdate (SBE) estimation of the batch between matrix updates. While MBGD and SVD cannot be efficiently implemented, SGD and SBE can. **Figure 2** outlines the key distinctions in the process execution of the algorithm.

To compare these approaches, we choose a very simple network architecture of $728 \times 100 \times 10$ neurons, using ReLU activation functions between layers and a cross-entropy loss function (LeCun et al., 1998). To control for the fact that using batches reduces the overall number of updates per epoch, we use a learning rate optimizer prior to network simulations, which minimizes the loss for 5 epochs. There is a hard cutoff terminating our simulations after 900 epochs. Batch sizes were

varied from 2^0 to 2^{13}. Networks are trained on the MNIST data set using the typical test-training partition. The exemplary series of networks trained below all began from the same randomly drawn starting condition.

RESULTS

To illustrate the convergence of the SBE algorithm during the batch training process, we calculate the error, ε, for the converging the singular vectors, $\mathbf{X}^{ij}$, to the true singular vectors, $\mathbf{X}^i$, as $\varepsilon = 1 - abs(\frac{(\mathbf{X}^i \cdot \mathbf{X}^{ij})}{\|\mathbf{X}^{ij}\|})$, and similarly for the singular value as $\varepsilon = 1 - abs(\frac{\sigma^i}{\sigma^{ij}})$[1]. **Figure 3** shows convergence curves of these errors during network training for batch sizes 32 and 1024. While 1024 shows strong periodic behavior between updates and convergence of the singular vectors down to an accuracy below 10^{-3}, the smaller batch size of 32 shows periodic behavior but no strong trends toward convergence of the approximate singular vector. Despite this weak convergence of the singular vector, the training of the network converges.

That fact that the convergence of the singular vectors is not necessary to demonstrate convergence of the network makes sense because convergence of vectors during power iterations is often determined by the eigengap, or the gap between the target eigenvalue and the next smallest eigenvalue of a matrix (Musco and Musco, 2015). A small eigengap leads to significant contamination of the target vector with other large eigenvalue vectors. This contamination complicates finding the eigenvector itself but still pushes a network to a lower value of the loss function.

Figure 4 shows that all the training algorithms reduce the training set loss function down to as low as 10^{-4}. We find that reducing the training set loss down to 10^{-2} is sufficient to achieve 100% accuracy on the training set and therefore about 97–98% accuracy on the test set. In these simulations, the SGD function is the fastest algorithm for training in terms of number of epochs, with MBGD, due to its parallelism, having significantly faster wall clock time. When re-plotting the data in terms of matrix updates, it's clear that the batch methods have an advantage in terms of minimizing the number of times the memory needs to be changed. However, these measures do not take into account the time that would be required to do the matrix updates in hardware. Since the SVD and SBE methods use only rank 1 updates, it takes less time for them to update the hardware by a factor of the number of elements in the crossbar.

These general trends can be seen in **Figure 5**, which shows the number of epochs and number of matrix updates needed to train the network to a training set loss of both 10^{-1} and 10^{-2}. For this example, MBGD is clearly the highest performing on all metrics, decreasing the number of updates needed to train the network vs. SGD by more than two orders of magnitude at a batch size of 4096. For the SBD and SBE algorithms, the epochs to train grows much faster, and the number of matrix updates needed to train only falls by a factor of 20 compared to SGD and does so at a much smaller batch size of 128. For very small batch sizes, the SBE algorithm performs worse than the SVD algorithm, which we attribute to poor qualities of the update vector, but at higher batch sizes it outperforms the SVD algorithm, which we attribute to a mixture of better update quality but with added stochasticity lacking in the SVD approach due to the random degree of convergence and sampling of lower significance eigenupdates and singular vectors.

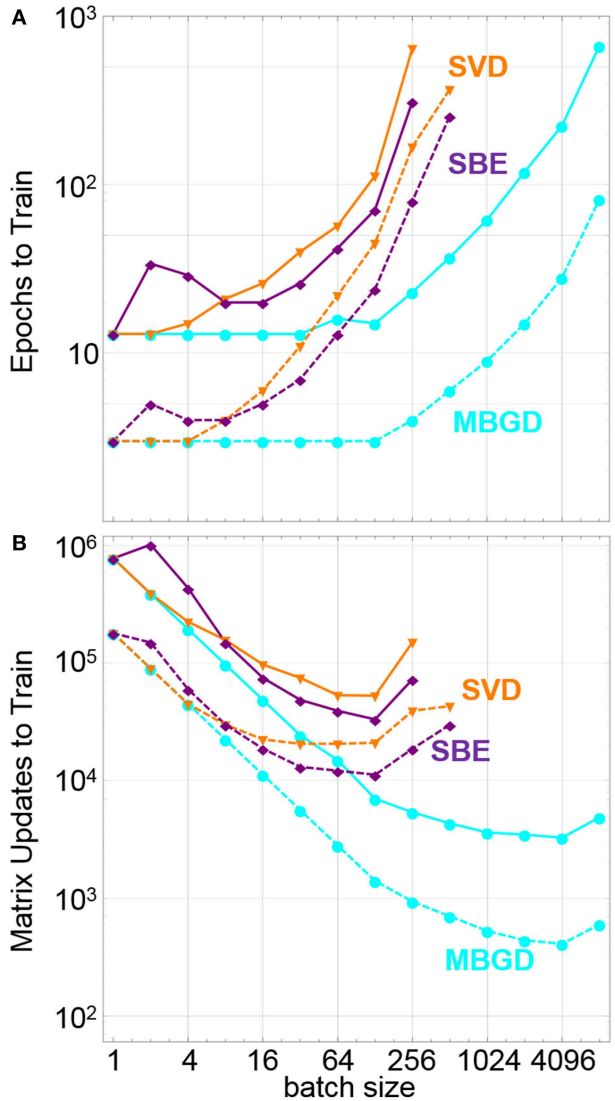

FIGURE 5 | Summary of impact of different training rules vs. batch size including **(A)** the number of epochs to train the training set loss function down to 0.1 (dashed lines) and 0.01 (solid lines), and **(B)** the number of matrix updates to set the loss function to 0.1(dashed lines) and to 0.01 (solid lines). The SVD and SBE training rules increase the update efficiency, but not as much as full batch update.

DISCUSSION

For the example below, the SBE approach is lower performing than the MBGD approach in terms of number of epochs to train and number of matrix updates. However, its use would vastly

accelerate the wall clock time of training in a hardware network since the transfer of the weights has the same complexity as the SGD approach, even in cases where the batches were stored in a local and parallel short-term memory array. Moreover, in the case of $k = 1$, calculating and storing the low rank versions of activations and error (left and right eigenvalues) take up significantly less area and compute ($O(a+b)$) as compared to the full rank ($O(a\times b)$) versions.

If a higher quality update were desired, the above algorithm could be extended to the calculation of multiple eigenupdates in parallel, similar to an Oja asymmetrical subspace network (Oja, 1992). The application of k eigenupdates would still be significantly faster than the time needed to transfer the point-wise or column-wise transfer for a full ranked batch update. Based on **Figure 1**, it is clear that a full rank transfer is unnecessary and possibly even detrimental if excess information leads to over fitting. Additionally, with high quality approximations of the gradient now available, it may be possible to implement as yet unknown algorithms which could accelerate training or provide a more efficient closed-loop transfer of the weight update to the devices. At a minimum, the SBE algorithm is able to reduce the memory and compute overhead required to do batch updates, which can reduce the writing and programming stress and latency on the devices by an order of magnitude.

The critical challenge is determining the most efficient hardware implementation of the SBE algorithm. The major operations required are the summation, multiplication and division respectively. Among them the most computationally intensive part is the normalization operation, $\frac{\mathbf{X}^{i,n}}{\|\mathbf{X}^{i,n}\|}$. Since we may only be working with low precision, such as 4-bit precision, and only dealing with a linear number of computations vs. problem size, the overhead of implementing these operations is significantly smaller when compared to their full rank counterparts. Digital implementations of such operations can be constructed with systolic array approaches, and if further energy efficiency is required, analog approaches can be used as well (Vanpoucke et al., 1994).

An alternate analog approach which gets rid of the division operation altogether is borrowed from the original Taylor series formulation of the Oja equations, which replaces division with a multiplication and subtraction (see **Supplementary Material**). Such a calculation, though, may run into issues with numerical stability. However, the physical constraints of the system along with the parallel calculation of additional singular vectors could stabilize the algorithm. Calculating multiple singular vectors is known to accelerate convergence of the dominant vectors (Balcan et al., 2016). Moreover, future hardware could likely use short term memory cells, such as trench capacitors and FETs, to perform resistive multiplication and dot product operations in combination with Gilbert cells to scale the outputted values properly (Li et al., 2018).

This is a rich tradeoff space which requires further exploration and is going to be part of follow-up work. Trading off the different system attributes, digital vs. analog, single vs. multi-rank, and batch size, can be used to build an optimal system for machine learning by efficiently calculating streaming batch eigenupdates.

DATA AVAILABILITY

The datasets generated for this study are available on request to the corresponding author.

AUTHOR CONTRIBUTIONS

BH conceived of the streaming batch eigenupdate concept for training networks. BH, MD, and MS developed the mathematical framework and algorithms implemented. BH and MD performed the network simulations. AM and BH analyzed implications of implementing the algorithm in hardware. All authors analyzed the results and wrote the manuscript.

FUNDING

AM acknowledges support from the Cooperative Research Agreement between the University of Maryland and the National Institute of Standards and Technology Center for Nanoscale Science and Technology, Award 70NANB14H209, through the University of Maryland.

ACKNOWLEDGMENTS

This manuscript has been released as a Pre-Print at arXiv.org (Hoskins et al., 2019).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2019.00793/full#supplementary-material

REFERENCES

Adam, G. C. (2018). Two artificial synapses are better than one. *Nature* 558:39. doi: 10.1038/d41586-018-05297-5

Adam, G. C., Hoskins, B. D., Prezioso, M., Merrikh-Bayat, F., Chakrabarti, B., and Strukov, D. B. (2017). 3-D memristor crossbars for analog and neuromorphic computing applications. *IEEE Transac. Electron Devices* 64, 312–318. doi: 10.1109/TED.2016.2630925

Allen-Zhu, Z., and Li, Y. (2017). "First efficient convergence for streaming K-Pca: a global, gap-free, and near-optimal rate," in *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, 487–92. Berkeley, CA: IEEE.

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

Antognini, J., and Sohl-Dickstein, J. (2018). "PCA of high dimensional random walks with comparison to neural network training," in *Advances in Neural Information Processing Systems 31*, eds S. Bengio, H. Wallach, H. Larochelle, K. Grauman N. Cesa-Bianchi, and R. Garnett (Curran Associates, Inc.), 10328–10337. Available online at: http://papers.nips.cc/paper/8232-pca-of-high-dimensional-random-walks-with-comparison-to-neural-network-training.pdf

Balcan, M. F., Du, S. S., Wang, Y., and Yu, A. W. (2016). "An improved gap-dependency analysis of the noisy power method," in *Conference on Learning Theory*, 284–309. Available online at: http://proceedings.mlr.press/v49/balcan16a.html

Balsubramani, A., Dasgupta, S., and Freund, Y. (2013). "The fast convergence of incremental PCA," in *Advances in Neural Information Processing Systems 26*, eds C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Curran Associates, Inc.), 3174–3182. Available online at: http://papers.nips.cc/paper/5132-the-fast-convergence-of-incremental-pca.pdf

Boybat, I., di Nolfo, C., Ambrogio, S., Bodini, M., Farinha, N. C. P., and Shelby, R. M., Narayanan, P., et al. (2017). "Improved deep neural network hardware-accelerators based on non-volatile-memory: the local gains technique," in *2017 IEEE International Conference on Rebooting Computing (ICRC)* (Washington, DC), 1–8.

Chakrabarti, B., Lastras-Montaño, M. A., Adam, G., Prezioso, M., Hoskins, B., Payvand, M., et al. (2017). A multiply-add engine with monolithically integrated 3D memristor crossbar/CMOS hybrid circuit. *Sci. Rep.* 7:42429. doi: 10.1038/srep42429

Clint, M., and Jennings, A. (1971). A simultaneous iteration method for the unsymmetric eigenvalue problem. *IMA J. Appl. Mathem.* 8, 111–121. doi: 10.1093/imamat/8.1.111

Gokmen, T., Rasch, M. J., and Haensch, W. (2018). Training LSTM networks with resistive cross-point devices. *Front. Neurosci.* 12:00745. 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.* 10:00333. doi: 10.3389/fnins.2016.00333

Golub, G. H., and Van Loan, C. F. (2013). *Matrix Computations*. Johns Hopkins University Press. Available online at: https://books.google.com/books?id=X5YfsuCWpxMC

Haensch, W., Gokmen, T., and Puri, R. (2019). The next generation of deep learning hardware: analog computing. *Proc. IEEE* 107, 108–122. doi: 10.1109/JPROC.2018.2871057

Hardt, M., and Price, E. (2014). "The noisy power method: a meta algorithm with applications," in *Advances in Neural Information Processing Systems 27*, eds Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Curran Associates, Inc.), 2861–2869. Available online at: http://papers.nips.cc/paper/5326-the-noisy-power-method-a-meta-algorithm-with-applications.pdf

Hoskins, B. D., Daniels, M. W., Huang, S., Madhavan, A., Adam, G. C., Zhitenev, N., et al. (2019). Streaming batch eigenupdates for hardware neuromorphic networks. *arXiv:1903.01635*.

Hua, Y., Xiang, Y., Chen, T., Abed-Meraim, K., and Miao, Y. (1999). A new look at the power method for fast subspace tracking. *Digital Signal Proc.* 9, 297–314. doi: 10.1006/dspr.1999.0348

Hyvärinen, A., and Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural Netw.* 13, 411–430. doi: 10.1016/S0893-6080(00)00026-5

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. *SIGARCH Comput. Archit. News* 45, 1–12. doi: 10.1145/3140659.3080246

Kataeva, I., Merrikh-Bayat, F., Zamanidoost, E., and Strukov, D. (2015). "Efficient training algorithms for neural networks based on memristive crossbar circuits," in *2015 International Joint Conference on Neural Networks (IJCNN)* (Killarney), 1–8.

Kim, S., Gokmen, T., Lee, H., and Haensch, W. E. (2017). "Analog CMOS-based resistive processing unit for deep neural network training," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 422–425. doi: 10.1109/MWSCAS.2017.8052950

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K. R. (1998). "Efficient BackProp," in *Neural Networks: Tricks of the Trade, This Book Is an Outgrowth of a 1996 NIPS Workshop* (London: Springer-Verlag), 9–50. Available online at: http://dl.acm.org/citation.cfm?id=645754.668382. doi: 10.1007/3-540-49430-8_2

Li, C.-L., Lin, H.-T., and Lu, C.-J. (2016). "Rivalry of two families of algorithms for memory-restricted streaming PCA," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, Vol. 51 (Cadiz: PMLR), 473–481.

Li, Y., Kim, S., Sun, X., Solomon, P., Gokmen, T., Tsai, H., Koswatta, S., et al. (2018). "Capacitor-based cross-point array for analog neural network with record symmetry and linearity," in *2018 IEEE Symposium on VLSI Technology* (Honolulu, HI), 25–26.

Lorch, E. (2016). "Visualizing Deep Network Training Trajectories with PCA," in *ICML Workship on Visualization for Deep Learning* (New York, NY).

Mitliagkas, I., Caramanis, C., and Jain, P. (2013). "Memory limited, streaming PCA," in *Advances in Neural Information Processing Systems 26*, eds C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Curran Associates, Inc.), 2886–2894. Available online at: http://papers.nips.cc/paper/5035-memory-limited-streaming-pca.pdf

Musco, C., and Musco, C. (2015). "Randomized block krylov methods for stronger and faster approximate singular value decomposition," in eds C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, R. Garnett (Curran Associates, Inc.), 1396–1404. Available online at: http://papers.nips.cc/paper/5735-randomized-block-krylov-methods-for-stronger-and-faster-approximate-singular-value-decomposition.pdf

Oja, E. (1982). Simplified neuron model as a principal component analyzer. *J. Mathem. Biol.* 15, 267–273. doi: 10.1007/BF00275687

Oja, E. (1992). Principal components, minor components, and linear neural networks. *Neural Netw.* 5, 927–935. doi: 10.1016/S0893-6080(05)80089-9

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

Strobach, P. (1997). Bi-Iteration SVD subspace tracking algorithms. *IEEE Transac. Signal Process.* 45, 1222–1240. doi: 10.1109/78.575696

Vanpoucke, F. J., Moonen, M., and Deprettere, E. F. A. (1994). "Numerically stable jacobi array for parallel Singular Value Decomposition (SVD) updating," in *Proc. SPIE*. Vol. 2296, (San Diego, CA).

Wang, Z., Joshi, S., Savel'ev, S., Song, W., Midya, R., Li, Y., Rao, M., et al. (2018). Fully memristive neural networks for pattern classification with unsupervised learning. *Nat. Electron.* 1:137. doi: 10.1038/s41928-018-0023-2

Yang, B. (1995). An Extension of the PASTd algorithm to both rank and subspace tracking. *IEEE Signal Process. Lett.* 2, 179–182. doi: 10.1109/97.410547

Yang, P., Hsieh, C.-J., and Wang, J.-L. (2018). *History PCA: A New Algorithm for Streaming PCA. arXiv[Preprint].arXiv:1802.05447* [Stat], February. Available online at: http://arxiv.org/abs/1802.05447

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.