frontiers in Neuroscience

ORIGINAL RESEARCH
published: 21 July 2016
doi: 10.3389/fnins.2016.00333

Acceleration of Deep Neural Network Training with Resistive Cross-Point Devices: Design Considerations

Tayfun Gokmen * *and Yurii Vlasov* †

IBM T. J. Watson Research Center, Yorktown Heights, NY, USA

In recent years, deep neural networks (DNN) have demonstrated significant business impact in large scale analysis and classification tasks such as speech recognition, visual object detection, pattern extraction, etc. Training of large DNNs, however, is universally considered as time consuming and computationally intensive task that demands datacenter-scale computational resources recruited for many days. Here we propose a concept of resistive processing unit (RPU) devices that can potentially accelerate DNN training by orders of magnitude while using much less power. The proposed RPU device can store and update the weight values locally thus minimizing data movement during training and allowing to fully exploit the locality and the parallelism of the training algorithm. We evaluate the effect of various RPU device features/non-idealities and system parameters on performance in order to derive the device and system level specifications for implementation of an accelerator chip for DNN training in a realistic CMOS-compatible technology. For large DNNs with about 1 billion weights this massively parallel RPU architecture can achieve acceleration factors of 30, 000× compared to state-of-the-art microprocessors while providing power efficiency of 84, 000 GigaOps/s/W. Problems that currently require days of training on a datacenter-size cluster with thousands of machines can be addressed within hours on a single RPU accelerator. A system consisting of a cluster of RPU accelerators will be able to tackle Big Data problems with trillions of parameters that is impossible to address today like, for example, natural speech recognition and translation between all world languages, real-time analytics on large streams of business and scientific data, integration, and analysis of multimodal sensory data flows from a massive number of IoT (Internet of Things) sensors.

Keywords: deep neural network training, synaptic device, machine learning, artificial neural networks, nanotechnology, materials engineering, electronic devices, memristive devices

OPEN ACCESS

Edited by:
Themis Prodromakis, University of Southampton, UK

Reviewed by:
Robert Legenstein, Graz University of Technology, Austria
Hesham Mostafa, University of Zurich and ETH Zurich, Switzerland

***Correspondence:**
Tayfun Gokmen
tgokmen@us.ibm.com

†***Present Address:***
Yurii Vlasov, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA

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

Received: *04 April 2016*
Accepted: *01 July 2016*
Published: *21 July 2016*

Citation:
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

INTRODUCTION

Deep Neural Networks (DNNs; LeCun et al., 2015) demonstrated significant commercial success in the last years with performance exceeding sophisticated prior methods in speech (Hinton et al., 2012) and object recognition (Krizhevsky et al., 2012; Simonyan and Zisserman, 2015; Szegedy et al., 2015). However, training the DNNs is an extremely computationally intensive task that requires massive computational resources and enormous training time that hinders their further application. For example, a 70% relative improvement has been demonstrated for a DNN with 1 billion connections that was trained on a cluster with 1000 machines for three days

(Le et al., 2012). Training the DNNs relies in general on the backpropagation algorithm that is intrinsically local and parallel (Rumelhart et al., 1986). Various hardware approaches to accelerate DNN training that are exploiting this locality and parallelism have been explored with a different level of success starting from the early 90s (Arima et al., 1991; Lehmann et al., 1993) to current developments with GPU (Coates et al., 2013; Wu et al., 2015), FPGA (Gupta et al., 2015) or specially designed ASIC (Chen Y. et al., 2014). Further acceleration is possible by fully utilizing the locality and parallelism of the algorithm. For a fully connected DNN layer that maps N neurons to N neurons significant acceleration can be achieved by minimizing data movement using local storage and processing of the weight values on the same node and connecting nodes together into a massive $N \times N$ systolic array (Lehmann et al., 1993) where the whole DNN can fit in. Instead of a usual time complexity of $O(N^2)$ the problem can be reduced therefore to a constant time $O(1)$ independent of the array size. However, the addressable problem size is limited to the number of nodes in the array that is challenging to scale up to billions even with the most advanced CMOS technologies.

Novel nano-electronic device concepts based on non-volatile memory (NVM) technologies, such as phase change memory (PCM; Jackson et al., 2013; Kuzum et al., 2013) and resistive random access memory (RRAM; Jo et al., 2010; Indiveri et al., 2013; Kuzum et al., 2013; Yu et al., 2013; Saïghi et al., 2015), have been explored recently for implementing neural networks with a learning rule inspired by spike-timing-dependent plasticity (STDP) observed in biological systems (Bi and Poo, 1998). Only recently, their implementation for acceleration of DNN training using backpropagation algorithm have been considered (Burr et al., 2014; Li et al., 2014; Xu et al., 2014; Prezioso et al., 2015; Soudry et al., 2015) with reported acceleration factors ranging from 27× (Burr et al., 2015) to 900× (Xu et al., 2014), and even 2140× (Seo et al., 2015) and significant reduction in power and area. All of these bottom-up approaches of using previously developed memory technologies looks very promising, however the estimated acceleration factors are limited by device specifications intrinsic to their application as NVM cells. Device characteristics usually considered beneficial or irrelevant for memory applications such as high on/off ratio, digital bit-wise storage, and asymmetrical set and reset operations, are becoming limitations for acceleration of DNN training (Burr et al., 2015; Yu et al., 2015). These non-ideal device characteristics can potentially be compensated with a proper design of peripheral circuits and a whole system, but only partially and with a cost of significantly increased operational time (Burr et al., 2015). In contrast, here we propose a top-down approach where ultimate acceleration of DNN training is achieved by design of a system and CMOS circuitry that imposes specific requirements for resistive devices. We propose and analyze a concept of Resistive Processing Unit (RPU) devices that can simultaneously store and process weights and are potentially scalable to billions of nodes with foundry CMOS technologies. As opposed to other approaches in the literature (Burr et al., 2014, 2015; Li et al., 2014; Xu et al., 2014; Prezioso et al., 2015; Seo et al., 2015; Soudry et al., 2015; Yu et al., 2015) the proposed final device characteristic that come out of this analysis allow a single device to perform all the operations required by the algorithm without additional circuit components. Our estimates indicate that acceleration factors close to 30,000× are achievable on a single chip with realistic power and area constraints.

MATERIALS AND METHODS

Definition of the RPU Device Concept

The backpropagation algorithm is composed of three cycles, forward, backward, and weight update that are repeated many times until a convergence criterion is met. The forward and backward cycles mainly involve computing vector-matrix multiplication in forward and backward directions. This operation can be performed on a 2D crossbar array of two-terminal resistive devices as it was proposed more than 50 years ago (Steinbuch, 1961). In forward cycle, stored conductance values in the crossbar array form a matrix, whereas the input vector is transmitted as voltage pulses through each of the input rows. In a backward cycle, when voltage pulses are supplied from columns as an input, then the vector-matrix product is computed on the transpose of a matrix. These operations achieve the required $O(1)$ time complexity, but only for two out of three cycles of the training algorithm.

In contrast to forward and backward cycles, implementing the weight update on a 2D crossbar array of resistive devices locally and all in parallel, independent of the array size, is challenging. It requires calculating a vector-vector outer product which consists of a multiplication operation and an incremental weight update to be performed locally at each cross-point as illustrated in **Figure 1A**. The corresponding update rule is usually expressed as (Rumelhart et al., 1986).

$$w_{ij} \leftarrow w_{ij} + \eta x_i \delta_j \tag{1}$$

where w_{ij} represents the weight value for the i^{th} row and the j^{th} column (for simplicity layer index is omitted) and x_i is the activity at the input neuron, δ_j is the error computed by the output neuron and η is the global learning rate.

In order to implement a **local and parallel** update on an array of two-terminal devices that can perform both weight storage and processing (RPU) we first propose to significantly simplify the multiplication operation itself by using stochastic computing techniques (Gaines, 1967; Poppelbaum et al., 1967; Alaghi and Hayes, 2013; Merkel and Kudithipudi, 2014). It has been shown that by using two stochastic streams the multiplication operation can be reduced to a simple AND operation (Gaines, 1967; Poppelbaum et al., 1967; Alaghi and Hayes, 2013). **Figure 1B** illustrates the stochastic update rule where numbers that are encoded from neurons (x_i and δ_j) are translated to stochastic bit streams using stochastic translators (STR). Then they are sent to the crossbar array where each RPU device changes its conductance (g_{ij}) slightly when bits from x_i and δ_j coincide. In this scheme we can write the update rule as follows.

$$w_{ij} \leftarrow w_{ij} \pm \triangle w_{min} \sum_{n=1}^{BL} A_i^n \wedge B_j^n \tag{2}$$

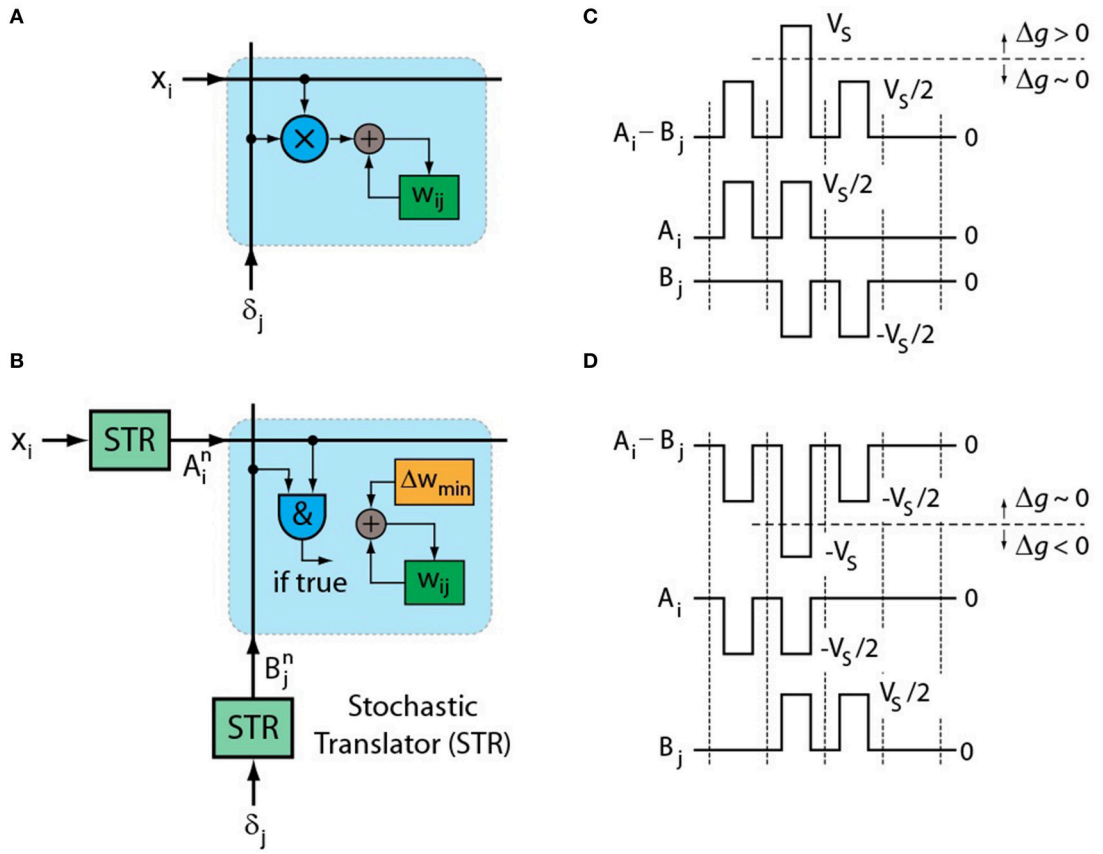

FIGURE 1 | (A) Schematics of original weight update rule of Equation (1) performed at each cross-point. **(B)** Schematics of stochastic update rule of Equation (2) that uses simple AND operation at each cross-point. Pulsing scheme that enables the implementation of stochastic updates rule by RPU devices for **(C)** up and **(D)** down conductance changes.

where BL is the length of the stochastic bit stream at the output of STRs that is used during the update cycle, Δw_{min} is the change in the weight value due to a single coincidence event, A_i^n and B_j^n are random variables that are characterized by a Bernoulli process, and the superscript n represents the bit position in the trial sequence. The probabilities that A_i^n and B_j^n are equal to unity are given by Cx_i and $C\delta_j$, respectively, where C is a gain factor in the STR.

One possible pulsing scheme that enables the stochastic update rule of Equation (2) is presented in **Figure 1C**. The voltage pulses with positive and negative amplitudes are sent from corresponding STRs on rows (A_i) and columns (B_j), respectively. As opposed to a floating point number encoded into a binary stream, the corresponding number translated into a stochastic stream is represented by a whole population of such pulses. In order for a two-terminal RPU device to distinguish coincidence events at a cross-point, its conductance value should not change significantly when a single pulse amplitude is half of the switching voltage of the device (V_S). However, when two pulses coincide and the RPU device sees the full voltage (V_S) the conductance should change by a nonzero amount Δg_{min}. The parameter Δg_{min} is proportional to Δw_{min} through the amplification factor defined by peripheral circuitry. To enable both up and down changes in conductance the polarity of the pulses can be switched during the update cycle as shown in **Figure 1D**. The sign of the multiplication is determined by the polarity of the pulses that are used during the update cycle. Therefore, for $x_i > 0$ cases the signed multiplication can be performed by populating the rows corresponding to $x_i > 0$ during both up and down cycles while the columns are populated selectively either at the up or the down cycle depending on the sign of δ_j. Similar operation can be repeated if there exists negative values ($x_i < 0$) for some of the rows. The proposed pulsing scheme allows all the RPU devices in an array to work in parallel and perform the multiplication operation locally by simply relying on the statistics of the coincidence events, thus achieving the $O(1)$ time complexity for the weight update cycle of the training algorithm.

RESULTS

Network Training with RPU Array Using a Stochastic Update Rule

To test the validity of this approach, we compare classification accuracies achieved with a deep neural network composed of fully

connected layers with 784, 256, 128, and 10 neurons, respectively. This network is trained with a standard MNIST training dataset of 60,000 examples of images of handwritten digits (LeCun et al., 1998) using the cross-entropy objective function and the backpropagation algorithm (Rumelhart et al., 1986). Raw pixel values of each 28 × 28 pixel image are given as inputs, while logistic sigmoid and softmax activation functions are used in hidden and output layers, respectively. The temperature parameter for both activation functions is assumed to be unity. **Figure 2** shows a set of classification error curves for the MNIST test dataset of 10,000 images. The curve marked with open circles in **Figure 2A** corresponds to a baseline model where the network is trained using the conventional update rule as defined by Equation (1) with a floating point multiplication operation. Here, the mini-batch size of unity is chosen throughout the following experiments. Training is performed repeatedly for all 60,000 images in the training dataset which constitutes a single training epoch. Learning rates of $\eta = 0.01$, 0.005, and 0.0025 for epochs 0–10, 11–20, and 21–30, respectively, are used. The baseline model reaches classification error of 2.0% on the test data in 30 epochs.

In order to make a fair comparison between the baseline model and the stochastic model in which the training uses the stochastic update rule of Equation (2), the learning rates need to match. In the most general form the average change in the weight value for the stochastic model can be written as.

$$\mathbb{E}\left(\Delta w_{ij}\right) = BL\,\Delta w_{min} C^2 x_i \delta_j \tag{3}$$

Therefore the learning rate for the stochastic model is controlled by three parameters BL, Δw_{min}, and C that should be adjusted to match the learning rates that are used in the baseline model.

Although the stochastic update rule allows to substitute the multiplication operation with a simple AND operation, the result of the operation, however, is no longer exact, but probabilistic with a standard deviation to mean ratio that scales with $1/\sqrt{BL}$. Increasing the stochastic bit stream length BL would decrease the error, but in turn would increase the update time. In order to find an acceptable range of BL values that allow to reach classification errors similar to the baseline model, we performed training using different BL values while setting $\Delta w_{min} = \eta/BL$ and $C = 1$ in order to match the learning rates used for the baseline model as discussed above. As it is shown in **Figure 2A**, BL as small as 10 is sufficient for the stochastic model to become indistinguishable from the baseline model.

In addition, for the stochastic update rule the change in the weight value for a single update cycle is bounded by $BL\ \Delta w_{min}$ and this condition may happen if the probabilities of generating pulses from STRs (Cx_i and $C\delta_j$) are close to unity or larger. The effect of this clipping in the weight update is also taken into account in our simulations and does not degrade the performance as shown in **Figure 2A** for BL as small as 10.

To determine how strong non-linearity in the device switching characteristics is required for the algorithm to converge to classification errors comparable to the baseline model, a non-linearity factor is varied as shown **Figure 2B**. The non-linearity factor is defined as the ratio of two conductance changes at half and full voltages as $k = \frac{\Delta g(V_S/2)}{\Delta g(V_S)}$. As shown in **Figure 2C**, the values of $k \approx 1$ correspond to a saturating type non-linear response, when $k = 0.5$ the response is linear as typically considered for an ideal memristor (Chua, 1971; Strukov et al., 2008), and values of $k \approx 0$ corresponds to a rectifying type non-linear response. As it is shown in **Figure 2B** the algorithm fails to converge for the linear response, however, a non-linearity factor k below 0.1 is enough to achieve classification errors comparable to the baseline model.

These results validate that although the updates in the stochastic model are probabilistic, classification errors can become indistinguishable from those achieved with the baseline model. The implementation of the stochastic update rule on an array of analog RPU devices with non-linear switching characteristics effectively utilizes the locality and the parallelism of the algorithm. As a result the update time is becoming

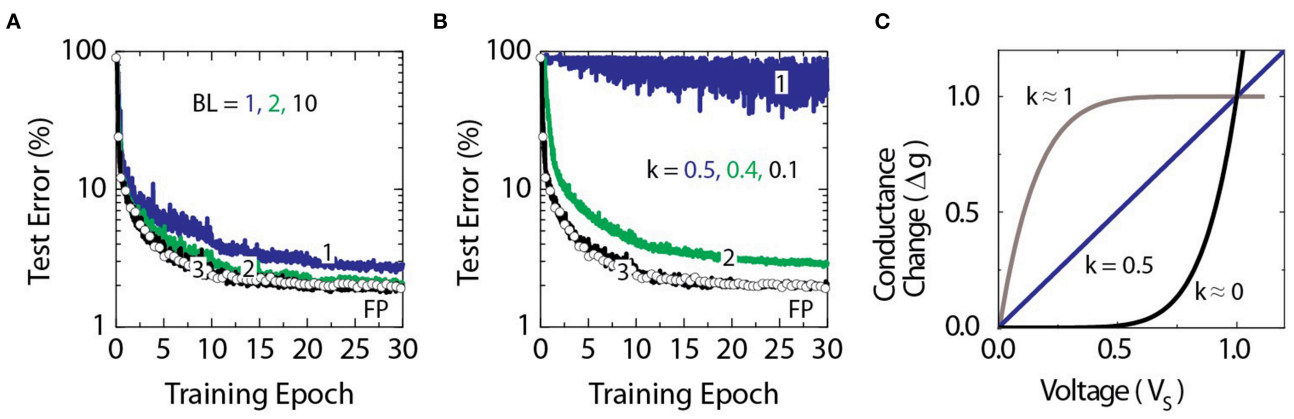

FIGURE 2 | Test error of DNN with the MNIST dataset. Open white circles correspond to the baseline model with the training performed using the conventional update rule of Equation (1). **(A)** Lines marked as 1, 2, and 3 correspond to the stochastic model of Equation (2) with stochastic bit lengths $BL = 1$, 2, and 10, respectively. **(B)** Lines marked as 1, 2, and 3 correspond to the stochastic model with $BL = 10$ and the non-linearity ratio $k = 0.5$, 0.4, and 0.1, respectively. **(C)** Illustration of various non-linear responses of RPU device with $k = 0$, 0.5, and 1.

independent of the array size, and is a constant value proportional to BL, thus achieving the required $O(1)$ time complexity.

Derivation of RPU Device Specifications

Various materials, physical mechanisms, and device concepts have been analyzed in view of their potential implementation as cross-bar arrays for neural network training (Burr et al., 2014, 2015; Li et al., 2014; Xu et al., 2014; Prezioso et al., 2015; Soudry et al., 2015). These technologies have been initially developed for storage class memory applications. It is not clear beforehand, however, whether intrinsic limitations of these technologies, when applied to realization of the proposed RPU concept, would result in a significant acceleration, or, in contrast, might limit the performance. For example, PCM devices can only increase the conductance during training, thus resulting in network saturation after a number of updates. This problem can be mitigated by a periodic serial reset of weights, however with a price of lengthening the training time (Burr et al., 2014, 2015) as it violates the $O(1)$ time complexity. In order to determine the device specifications required to achieve the ultimate acceleration when $O(1)$ time complexity is reached, we performed a series of trainings summarized in **Figure 3**. Each figure corresponds to a specific "stress test" where a single parameter is scanned while all the others are fixed allowing to explore the acceptable RPU device parameters that the algorithm can tolerate without significant error penalty. This includes variations in RPU device switching characteristics, such as, incremental conductance change due to a single coincidence event, asymmetry in up and down conductance changes, tunable range of the conductance values, and various types of noise in the system. For all of the stochastic models illustrated in **Figure 3**, $k = 0$ and $BL = 10$ is used. In order to match the learning rates used for the baseline model the x_i and δ_j are translated to stochastic streams with C defined as

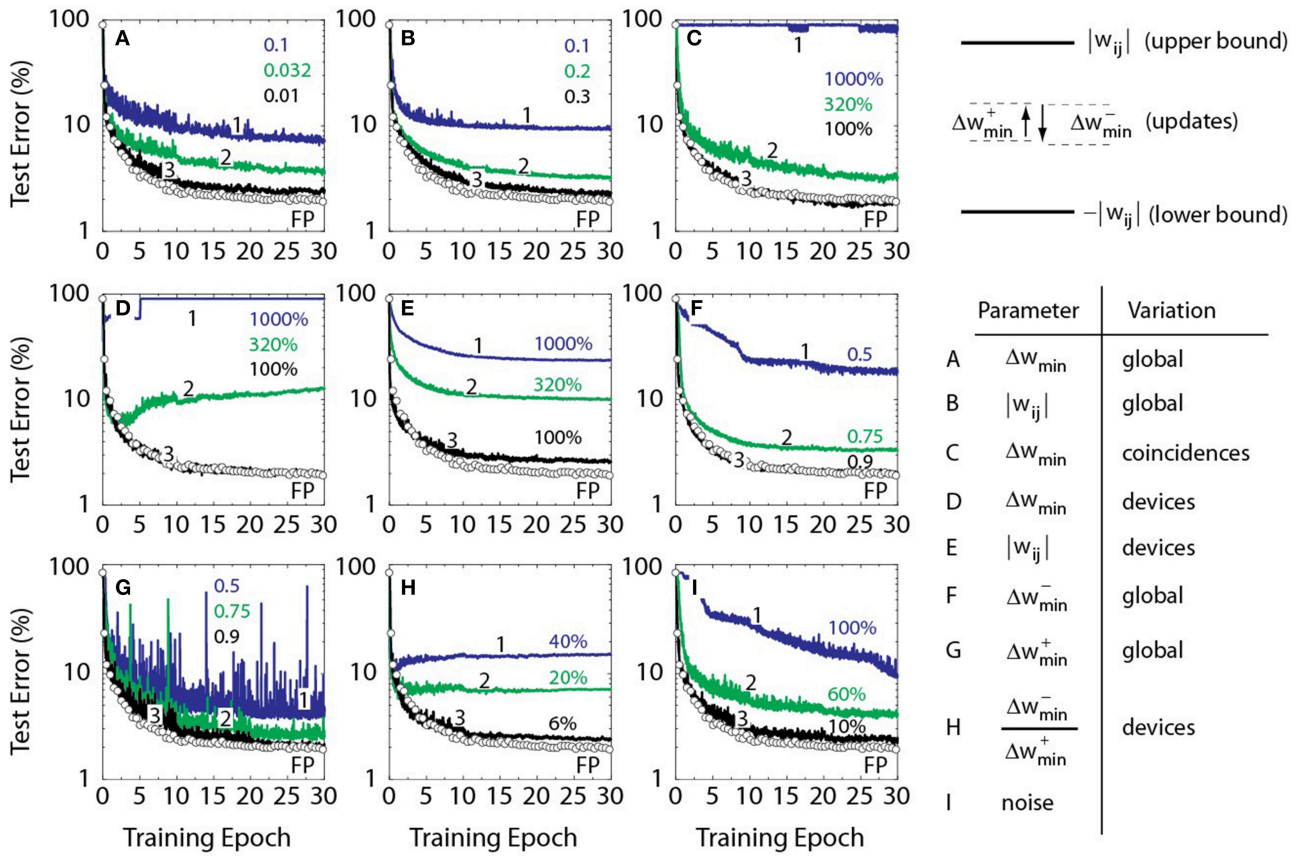

FIGURE 3 | Test error of DNN with the MNIST dataset. Open white circles correspond to a baseline model where the training is performed using the conventional update rule of Equation (1). All solid lines assume a stochastic model with $BL = 10$ and $k = 0$. **(A)** Lines 1, 2, and 3 correspond to a stochastic model with $\Delta w_{min} = 0.1$, 0.032, and 0.01, respectively. All curves in **(B–I)** use $\Delta w_{min} = 0.001$. **(B)** Lines 1, 2, and 3 correspond to a stochastic model with weights bounded to 0.1, 0.2, and 0.3, respectively. **(C)** Lines 1, 2, and 3 correspond to a stochastic model with a coincidence-to-coincidence variation in Δw_{min} of 1000, 320, and 100%, respectively. **(D)** Lines 1, 2, and 3 correspond to a stochastic model with device-to-device variation in Δw_{min} of 1000, 320 and 100%, respectively. **(E)** Lines 1, 2, and 3 correspond to a stochastic model with device-to-device variation in the upper and lower bounds of 1000, 320 and 100%, respectively. All solid lines in E have a mean value of 1.0 for upper bound (and −1.0 for lower bound). **(F)** Lines 1, 2, and 3 correspond to a stochastic model, where down changes are weaker by 0.5, 0.75, and 0.9, respectively. **(G)** Lines 1, 2, and 3 correspond to a stochastic model, where up changes are weaker by 0.5, 0.75, and 0.9, respectively. **(H)** Lines 1, 2, and 3 correspond to a stochastic model with device-to-device variation in the up and down changes by 40, 20 and 6%, respectively. **(I)** Lines 1, 2, and 3 correspond to a stochastic model with a noise in vector-matrix multiplication of 100, 60, and 10%, respectively, normalized on activation function temperature which is unity.

$C = \sqrt{\eta/(BL\ \Delta w_{min})}$. This allows the average learning rate to be the same as in the baseline model.

Ideally, the RPU device should be analog i.e., the conductance change due to a single coincidence event Δg_{min} should be arbitrarily small, thus continuously covering all the allowed conductance values. To determine the largest acceptable Δg_{min} due to a single coincidence event that does not produce significant error penalty, the parameter Δw_{min} is scanned between 0.32 and 0.00032, while other parameters are fixed as shown in **Figure 3A**. While for large Δw_{min} the convergence is poor since it controls the standard deviation of the stochastic update rule, for smaller Δw_{min} the results are approaching the baseline model. The Δw_{min} smaller than 0.01 produces a classification error of 2.3% at the end of 30$^{\text{th}}$ epoch which is just 0.3% above the 2.0% classification error of the baseline model.

To determine minimum and maximum conductance values that RPU devices should support for the algorithm to converge, a set of training curves is calculated as shown in **Figure 3B**. Each curve is defined by the weight range where the absolute value of weights $|w_{ij}|$ is kept below a certain bound that is varied between 0.1 and 3. The other parameters are identical to **Figure 3A**, while Δw_{min} is taken as 0.001 to assure that the results are mostly defined by the choice of the weight range. The model with weights $|w_{ij}|$ bounded to values larger than 0.3 results in an acceptable error penalty criteria of 0.3% as defined above. Since, the parameter Δg_{min} (and g_{ij}) is proportional to Δw_{min} (and w_{ij}) through the amplification factor defined by peripheral circuitry, the number of coincidence events required to move the RPU device from its minimum to its maximum conductance value can be derived as $\left(\max\left(g_{ij}\right) - \min\left(g_{ij}\right)\right)/\Delta g_{min} = \left(\max\left(w_{ij}\right) - \min\left(w_{ij}\right)\right)/\Delta w_{min}$. This gives a lower estimate for the number of states that are required to be stored on an RPU device as 600.

In order to determine the tolerance of the algorithm to the variation in the incremental conductance change due to a single coincidence event Δg_{min}, the Δw_{min} value used for each coincidence event is assumed to be a random variable with a Gaussian distribution. Corresponding results are shown in **Figure 3C**, where the standard deviation is varied while the average Δw_{min} value is set to 0.001. In our models it is allowed to have coincidence events that result in a change in the opposite direction if the random value is less than -1 (or -100%). As it is seen, the algorithm is robust against the randomness on the weight change for each coincidence event and models with a standard deviation below 150% of the mean value reach acceptable 0.3% error penalty.

For stochastic models illustrated in **Figure 3D**, yet another randomness, a device-to-device variation in the incremental conductance change due to a single coincidence event Δg_{min}, is introduced. In this case the Δw_{min} used for each RPU device is sampled from a Gaussian distribution at the beginning of the training and then this fixed value is used throughout the training for each coincidence event. For all stochastic models in **Figure 3D**, the average Δw_{min} value of 0.001 is used while the standard deviation is varied for each model. In our models it is allowed to have some devices that perform updates in the opposite direction throughout the training if the random value is less than -1 (or -100%). Results show that the algorithm is also robust against the device-to-device variation and an acceptable error penalty can be achieved for models with a standard deviation up to 110% of the mean value.

To determine tolerance of the algorithm to the device-to-device variation in the upper and lower bounds of the conductance value, we assumed upper and lower bounds that are different for each RPU device for the models in **Figure 3E**. The bounds used for each RPU device are sampled from a Gaussian distribution at the beginning of the training and are used throughout the training. For all of the stochastic models in **Figure 3E**, mean value of 1.0 for upper bound (and -1.0 for lower bound) is used to assure that the results are mostly defined by the device-to-device variation in the upper and lower bounds. We note that as the standard deviation becomes large enough some devices may encode only positive or only negative weight values. Moreover, some devices might even have an upper bound that is smaller than the lower bound and those devices are assumed to be stack at the middle point and do not respond to the updates. Including all of these contributions, **Figure 3E** shows that the algorithm is robust against the variation in the bounds and models with a standard deviation up to 80% of the mean can achieve acceptable 0.3% error penalty.

Fabricated RPU devices may also show different amounts of change in the conductance value due to positive (Δg^{+}_{min}) and negative (Δg^{-}_{min}) pulses as illustrated in **Figures 1C,D**. To determine how much asymmetry between up and down changes the algorithm can tolerate, the up (Δw^{+}_{min}) and down (Δw^{-}_{min}) changes in the weight value are varied as shown in **Figures 3F,G**. In both cases this global asymmetry is considered to be uniform throughout the whole RPU device array. For each model in **Figure 3F** Δw^{+}_{min} is fixed to 0.001 while Δw^{-}_{min} is varied from 0.95 to 0.25 weaker than the up value. Similarly, **Figure 3G** shows an analogous results for Δw^{-}_{min} fixed to 0.001 while Δw^{+}_{min} is varied. Results show that up and down changes need to be significantly balanced (10% with respect to each other) in order for the stochastic model to achieve an acceptable 0.3% error penalty. We define the threshold value with respect to the mean and therefore 5% imbalance is used as the acceptable threshold. We note that the large fluctuations seen in **Figure 3G** but not in **Figure 3F** is a little surprising. The origin is unclear but according to our preliminary calculations it is not due to stochastic nature of the updates. We performed training using floating point multiplication with an imbalance term and still observed a similar behavior.

In order to determine tolerance of the algorithm to the device-to-device variation in asymmetry, as opposed to a global asymmetry considered in **Figures 3F,G**, the curves in **Figure 3H** are calculated for various values of the standard deviation of $\Delta w^{+}_{min}/\Delta w^{-}_{min}$. The parameters Δw^{+}_{min} and Δw^{-}_{min} for each RPU device are sampled from a Gaussian distribution at the beginning of the training and then used throughout the training for each coincidence event. All the models assume that the average value of Δw^{+}_{min} and Δw^{-}_{min} is 0.001. The standard deviation of $\Delta w^{+}_{min}/\Delta w^{-}_{min}$ needs to be less than 6% of the mean value to achieve an acceptable 0.3% error penalty.

Analog computation is sensitive to various noise sources such as thermal noise, shot noise, etc that are all additive and can be modeled as a single unbiased Gaussian noise. Influence of noise penalty during the weight update cycle is already considered in **Figures 3C–H**. In order to estimate tolerance of the algorithm to noise during forward and backward cycles, we injected Gaussian noise to the results of vector-matrix multiplications with varying standard deviation. After the result of a vector-matrix multiplication is computed, an independent random noise is added to the each element of the resulting vector. For the data shown in **Figure 3I** the same noise distribution is used both for the forward and the backward cycles and an acceptable 0.3% error penalty is reached for a noise with a standard deviation of 0.1. This value is 10% of the sigmoid neuron temperature parameter which is unity. However, the noise requirements for the forward and the backward cycles may not be identical. Indeed, calculations show that when we introduce noise only to the forward cycle, the algorithm can tolerate up to six times larger noise with a 60% standard deviation. The backward cycle is less tolerant with a 10% threshold and therefore it dictates the threshold value derived from **Figure 3I**.

The radar diagram in **Figure 4A** summarizes specifications of RPU devices that are derived from the "stress tests" performed in **Figure 3**. Axes C-I correspond to experiments in **Figures 3C–I**, respectively. Solid line 1 connects threshold values determined for these parameters for an acceptable 0.3% error penalty. Note that these specifications differ significantly from parameters typical for NVM technologies. The storage in NVM devices is digital and typically does not exceed a few bits. This constraint is imposed by the system requirement to achieve high signal-to-noise ratio for read and write operations. In addition, the write operation does not depend on history as it overwrites all previously stored values. In contrast, weight values in the neural network operation are not needed to be written and resolved with very high signal-to-noise ratio. In fact, the algorithm can withstand up to 150% of noise in the weights updates (parameter C) and can tolerate up to 10% reading noise on columns or rows (parameter I). However, as opposed to a few bit storage capacity on NVM devices, a large number of coincidence events (over 600 from **Figure 3B**) is required for the RPU device to keep track of the history of weight updates. In addition, in contrast to high endurance of full swing writing between bit levels required for NVM devices, RPU devices need to have high endurance only to small incremental changes, $\triangle g_{min}$.

Combined contribution of all parameters considered in **Figure 4A** can be additive and therefore exceed the acceptable 0.3% error penalty. **Figure 4B** shows training results when effects of more than one parameter are combined. When all parameters (C, D, E, F, G, H, and I) are combined at the threshold the test error reaches 5.0% that is 3.0% above the baseline model. Although this penalty can be acceptable for some applications, it is significantly higher than the 0.3% error penalty considered above.

This 3.0% penalty is higher than a simple additive impact of uncorrelated contributions indicating that at least some of these parameters are interacting. It opens the possibility of optimizing the error penalty by trading off tolerances between various parameters. For example, the model that combines only parameters C, D, and E at the threshold, as shown by curve 2 in **Figure 4B**, gives 0.9% error penalty that is about the expected sum of individual contributions. Note that these parameters are defined by imperfections in device operation and by device-to-device mismatch that are all controlled by fabrication tolerances in a given technology. Even for deeply scaled CMOS technologies the fabrication tolerances do not exceed 30% that is much smaller than 150, 110, and 80% used

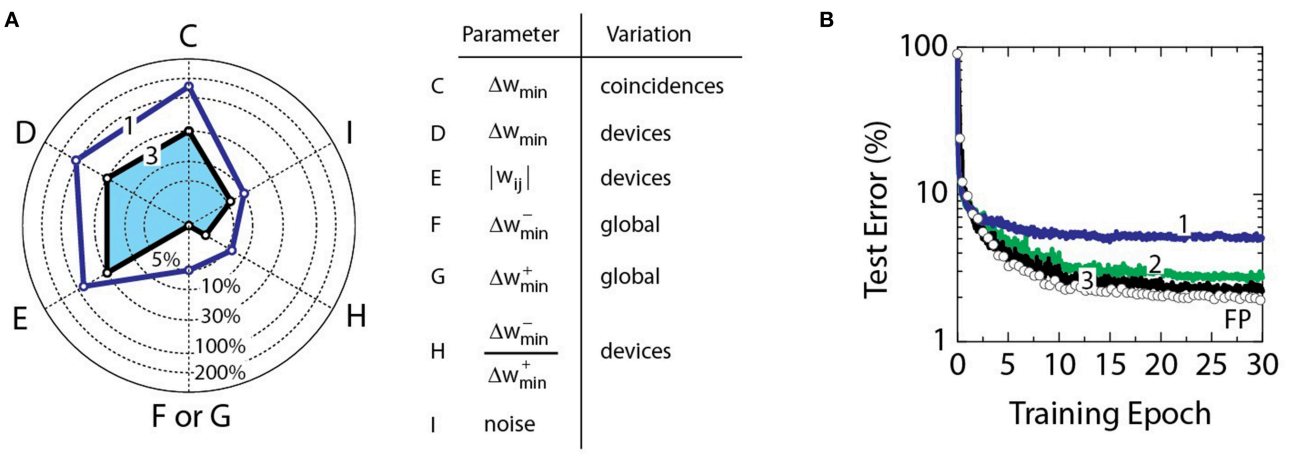

FIGURE 4 | (A) Line 1 shows threshold values for parameters from **Figure 3** assuming a 0.3% error penalty. Parameters C–I correspond to experiments in **Figures 3C–I**, respectively. The blue shaded area bounded by line 3 results in at most 0.3% error penalty when all parameters are combined. **(B)** Curve 1 corresponds to a model with all parameters combined at the threshold value as shown in the radar diagram by line 1. Curve 2 corresponds to a model with only C, D, and E combined at the threshold. Curves 1 and 2 have a mean value of 1.0 for weight bounds $|w_{ij}|$. Curve 3 corresponds to a model with C, D, E at 30%, F/G at 0%, H at 2 and I at 6%, all combined as shown in the radar diagram by line 3. Curves 3 has a mean value of 0.6 for weight bounds $|w_{ij}|$. All curves use a mean value of 0.001 for $\triangle w_{min}$.

for calculation of curve 2. The contributions of C, D, and E to the error penalty can be eliminated by setting the corresponding tolerances to 30% (data not shown).

Among the parameters of **Figure 4A**, the asymmetry between up and down changes in the conductance value of RPU devices (parameter F, G, and H) is the most restrictive. Parameter F (or G) is the global asymmetry that can be compensated by controlling pulse voltages and/or number of pulses in the positive and negative update cycles, and hence even asymmetries higher than the threshold value of 5% can be eliminated with proper design of peripheral circuits. In contrast, the parameter H that is defined by device-to-device variation in the asymmetry, can be compensated by peripheral circuits only if each RPU device is addressed serially. To maintain the $O(1)$ time complexity, the device mismatch parameter H and the noise parameter I can be co-optimized to reduce the error penalty. The resulting model illustrated by the blue shaded area bounded with curve 3 in **Figure 4B** achieves at most 0.3% error penalty. For this model parameters C, D, and E are set to 30% while F (or G) is set to zero, H is set to 2%, and I is set to 6%. Alternatively, the same result (data not shown) can be obtained by restricting the noise parameter I to 2.5% and increasing the device mismatch tolerance H to 4% that can simplify the array fabrication in expense of designing less noisy circuits.

In addition to the parameters considered above, RPU device may also show dependence on the conductance change on the stored conductance value $\triangle g_{min}(g_{ij})$. Such a behavior introduces an update rule that depends on the current weight value which can be written as $\triangle w_{min}(w_{ij})$. We performed simulations including a weight dependent update rule with different functional forms for $\triangle w_{min}(w_{ij})$ that included a linear or a quadratic dependence on weight value. In the first set of simulations we assume that the updates are balanced for any given weight value such that $\triangle w^{+}_{min}\left(w_{ij}\right) = \triangle w^{-}_{min}\left(w_{ij}\right)$ and therefore already satisfy the imbalance criteria H throughout the whole weight range. These simulation results show that the dependence of $\triangle g_{min}$ on g_{ij} is not an important parameter as no additional error penalty above 0.3% is observed even when $\triangle w_{min}$ is varied by a factor of about 10. However, when we introduce weight dependent updates that are not balanced, we observe additional error penalty as this condition violates imbalance criteria H.

Circuit and System Level Design Considerations

The ultimate acceleration of DNN training with the backpropagation algorithm on a RPU array of size $N \times N$ can be approached when $O(1)$ time complexity operation is enforced. In this case overall acceleration is proportional to N^2 that favors very large arrays. In general the design of the array, peripheral circuits, and the whole system should be based on an optimization of the network parameters for a specific workload and classification task. In order to develop a general methodology for such a design, we will use the results of the analysis presented above as an example with understanding, however, that the developed approach is valid for a larger class of more complicated cases than a relatively simple 3 layer network used to classify the MNIST dataset in **Figures 2–4**.

RPU Array Design

For realistic technological implementations of the crossbar array, the array size will ultimately be limited by resistance and parasitic capacitance of the transmission lines resulting in significant RC delay and voltage drop. For further analysis we assume that RPU devices are integrated at the back-end-of-line (BEOL) stack in-between intermediate metal levels. This allows the top thick metal levels to be used for power distribution, and the lower metal levels and the area under the RPU array for peripheral CMOS circuitry. Typical intermediate metal levels in a scaled CMOS technology have a thickness of 360 nm, and a width of 200 nm. Corresponding typical line resistance is about $r_{line} = 0.36\ \Omega/\mu m$ with parasitic capacitance of $c_{line} = 0.2\ fF/\mu m$. Assuming a reasonable 1 GHz clock frequency for the pulses used during the update cycle, and allowing RC delay to be at most 10% of the pulse width (0.1 ns), the longest line length should be $l_{line} = 1.64\ mm$. Assuming a reasonable line spacing of 200 nm this results in an array with 4096 × 4096 RPU devices. Since the conductance values of RPU devices can only be positive, we assume that a pair of identical RPU device arrays is used to encode positive (g^{+}_{ij}) and negative (g^{-}_{ij}) weight values. The weight value (w_{ij}) is proportional to a difference of two conductance values stored in two corresponding devices ($g^{+}_{ij} - g^{-}_{ij}$) located in identical positions of a pair of RPU arrays. To minimize the area, these two arrays can be stacked on top of each other occupying 4 consecutive metal levels resulting in a total area of $A_{array} = 2.68\ mm^2$. For this array size a full update cycle (both positive and negative) performed using 1 ns pulses can be completed in 20 ns for $BL = 10$.

In order to estimate an average RPU device resistance, R_{device}, we assume at most 10% voltage drop on the transmission line that is defined by $N \times R_{line}/R_{device}$, where R_{line} is the total line resistance equal to $r_{line}l_{line}$. The contribution of the output resistance of the line drivers to the total line resistance can be minimized by proper circuit design. For an array size of $N = 4096$ the average RPU device resistance is therefore $R_{device} = 24\ M\Omega$. Using this resistance value, and assuming an operating voltage of 1 V for all three training cycles and on-average about 20% activity for each device that is typical for the models of **Figures 2–4**, the power dissipation on a pair of RPU arrays can be estimated as $P_{array} = 0.28\ W$.

Design of Peripheral Circuits

Operation of a single column (or row) during forward (or backward) cycle is illustrated in **Figure 5A**. In contrast to the update cycle, stochastic translators are not needed. Here we assume that time-encoding scheme is used when input vectors are represented by fixed amplitude $V_{in} = 1\ V$ pulses with a tunable duration. Pulse duration is a multiple of 1 ns and is proportional to the value of the input vector. Currents generated at each RPU device are summed on the columns (or rows) and this total current is integrated over the measurement time, t_{meas} by current readout circuits as illustrated in **Figure 5A**. Positive and negative voltage pulses are supplied separately to each of the identical

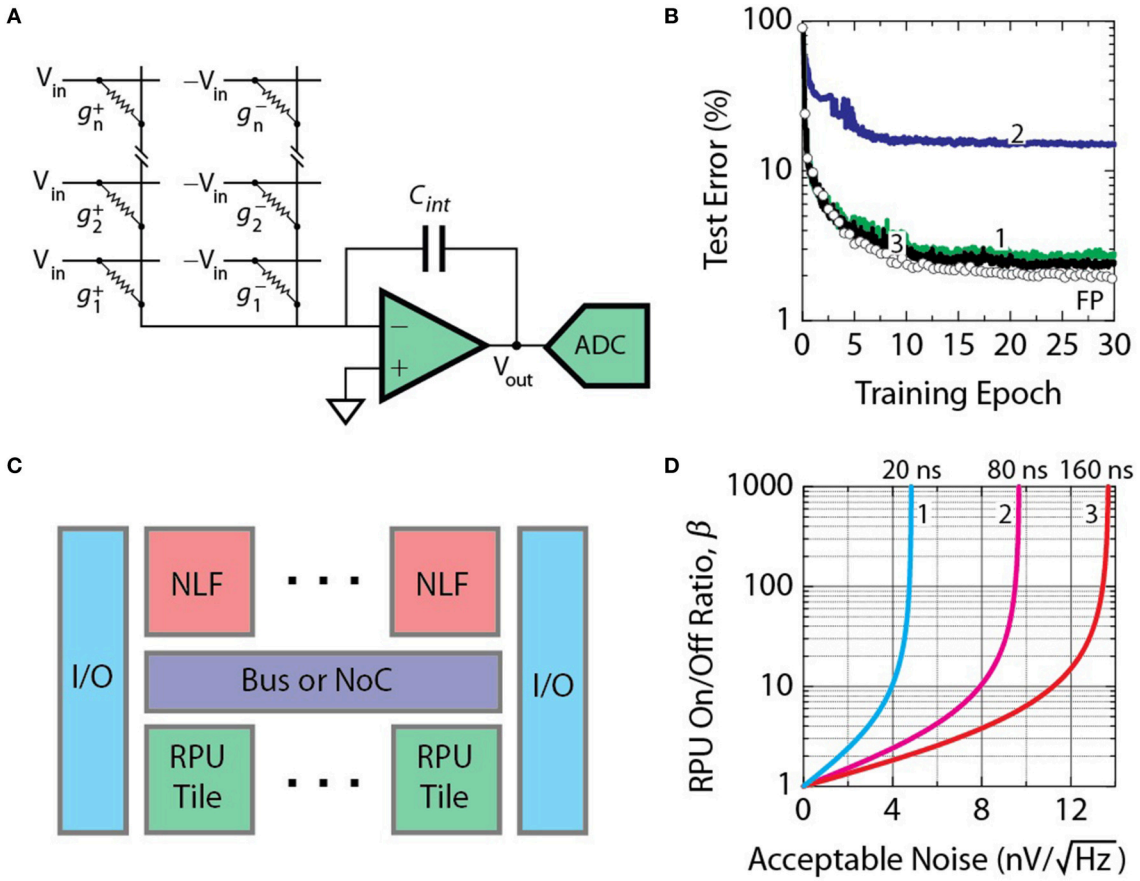

FIGURE 5 | (A) Operation of a single column (or row) during forward (or backward) cycle showing an op-amp that integrates the differential current on the capacitor C_{int}, and an analog-to-digital converter ADC. **(B)** Test error for the network of model 3 in **Figure 4B** with bounds $|\alpha|$ imposed on results of vector-matrix multiplication. Curve 1 corresponds to a model with $|\alpha| = 3$ imposed only on sigmoid activation function in hidden layers. Curves 2 and 3 corresponds to a model with $|\alpha| = 3$ and 12, respectively, imposed on both sigmoid and softmax activation functions. **(C)** Schematics of the architecture for accelerator RPU chip. RPU tiles are located on the bottom, NLF digital compute circuits are on the top, on-chip communication is provided by a bus or NoC, and off-chip communication relies on I/O circuits. **(D)** Acceptable input referred noise levels for various on/off ratio on the RPU devices β and integration times t_{meas}. Curves 1, 2, and 3 corresponds to t_{meas} of 20, 80, and 160 *ns*, respectively.

RPU arrays that are used to encode positive and negative weights. Currents from both arrays are fed into peripheral circuitry that consists of an op-amp that integrates differential current on the capacitor C_{int}, and an analog-to-digital converter ADC. Note, that for time-encoded pulses, the time-quantization error at the input to the RPU array scales inversely with the total number of pulses and, therefore, it is a better approach compared to the stochastic pulsing scheme (O'Connor and Welling, 2016). For the models in **Figure 4B** number of pulses larger than 20 (∼5 *bit* resolution) is enough to eliminate corresponding error penalty.

We define a single RPU tile as a pair of arrays with 4096 × 4096 devices with peripheral circuits that support the parallel operation of the array in all three cycles. Peripheral circuitry includes ADCs, op-amps, STRs consisting of random number generators, and line drivers used to direct signals along the columns and rows. As shown in **Figure 5C** the signals from a tile are directed toward a non-linear function (NLF) circuit that calculates either activation functions (i.e., sigmoid, softmax) and their derivatives as well as arithmetical operations (i.e., multiplication) depending on cycle type and on position of corresponding layer. At the tile boundary input signals to the NLF are bounded to a certain threshold value to avoid signal saturation. **Figure 5B** shows test error for the network of the model 3 in **Figure 4B**, but with bounds $|\alpha|$ imposed on results of vector-matrix multiplication that is equivalent to restricting the NLF input. For neurons in hidden layers the NLF circuit should compute a sigmoid activation function. When the input to this sigmoid NLF is restricted to $|\alpha| = 3$, the resulting error penalty does not exceed an additional 0.4% as shown by curve 1 in **Figure 5B**.

Neurons at the output layer perform a softmax NLF operation, that, when corresponding input is also restricted to $|\alpha| = 3$, results in exceedingly large error as shown by curve 2 in **Figure 5B**. To make design more flexible and programmable it is desired for the NLF in both hidden and output layers to have the same bounds. When bounds on both softmax and sigmoid

NLF are restricted to $|\alpha| = 12$, the total penalty is within acceptable range as shown by curve 3. Assuming 6% acceptable noise level taken from the results of **Figure 4B** and an operation voltage range between -1 V and 1 V at the input to the ADC, the corresponding bit resolution and voltage step required are 9 *bits* and 3.9 mV, respectively. These numbers imply that the acceptable total integrated RMS voltage noise at the input to the ADC (or at the output of the op-amp) should not exceed 5.0 mV.

Noise Analysis

In order to estimate the acceptable level of the input referred noise the integration function of the op-amp should be defined. Voltage at the output of the op-amp can be derived as

$$V_{out} = 2N \frac{V_{in}\, t_{meas}}{R_{device}\, C_{int}} \left(\frac{\beta - 1}{\beta + 1} \right) \quad (4)$$

where β is the conductance on/off ratio for an RPU device. This equation assumes all N devices are contributing simultaneously that makes it hard to design a circuit that would require either a very large capacitor or large voltage swing. However, for a given bounds $|\alpha|$ imposed on the NLF transformation and $|w_{ij}|$ for the weight values, the output voltage should not necessarily exceed the level corresponding to simultaneous contribution of $|\alpha| / |w_{ij}|$ devices. Since, as shown above, an acceptable bound $|\alpha| = 12$ and $|w_{ij}| = 0.6$ is enough, the number N in Equation (4) can be replaced with 20. Assuming that V_{out} signal feeding into the ADC should not exceed 1 V, and the R_{device} is 24 $M\Omega$, the choice of integrating capacitor C_{int} is dictated by the integration time t_{meas} and on/off ratio β. **Figure 5D** presents estimates of acceptable noise levels for various on/off ratios on the devices β and integration times t_{meas}. This noise level corresponds to the input referred noise of the op-amp calculated using standard noise analysis in integrator-based circuits (Jensen et al., 2013). If t_{meas} is taken as 20 *ns* following the quantization error consideration discussed above, the acceptable noise levels are relatively low of the order of just 5 $nV/\sqrt{Hz}$ as seen in **Figure 5D** curve 1. Even an increase of the on/off ratio β to several orders of magnitude does not help to accommodate higher noise. In order to accommodate higher noise t_{meas} needs to be increased with a penalty, however, of increased overall calculation time. As seen from curves in **Figure 5D**, for a given noise level the on/off ratios as small as 2–10 can be acceptable that is, in fact, quite modest in comparison to several orders of magnitude typical for NVM applications. When t_{meas} and β are chosen as 80 *ns* and 8, respectively, corresponding level of acceptable input referred noise shown by curve 2 in **Figure 5D** can be derived as 7.6 $nV/\sqrt{Hz}$. We note that this budget is calculated using the requirements for the backward pass, while for the forward pass the acceptable noise level, as discussed above, is about six times larger with a value of about 45 $nV/\sqrt{Hz}$. Corresponding capacitance C_{int} can also be calculated as 103 fF using Equation (4).

Various noise sources can contribute to total acceptable input referred noise level of an op-amp including thermal noise, shot noise, and supply voltage noise, etc. Thermal noise due to a pair of arrays with 4096 × 4096 RPU devices can be estimated as 7.0 $nV/\sqrt{Hz}$. Depending on exact physical implementation of a RPU device and type of non-linear $I - V$ response, shot noise levels produced by the RPU array can vary. Assuming a diode-like model, total shot noise from a whole array scales as a square root of a number of active RPU devices in a column (or a row), and hence depends on an overall instantaneous activity of the array. The average activity of the network that is typical for the models of **Figures 2–4**, is less than 1% for the backward cycle, while for the forward cycle it is much higher approaching 20%. Correspondingly, these activities result in shot noise values of 3.1 $nV/\sqrt{Hz}$ and 13.7 $nV/\sqrt{Hz}$, for backward and forward cycles respectively. Therefore, the noise in the backward cycle is dominated by the thermal noise with a value of 7.0 $nV/\sqrt{Hz}$ and together with the shot noise contribution fits the total noise budget of 7.6 $nV/\sqrt{Hz}$. In contrast, the noise in the forward cycle is dominated by the shot noise with value of 13.7 $nV/\sqrt{Hz}$ and it also fits the corresponding total noise budget of 45 $nV/\sqrt{Hz}$. We note that longer integration time or smaller array size is needed for higher workloads or additional noise contributions including the noise on the voltage, amplifier noise, etc.

System Level Design Considerations

The tile area occupied by peripheral circuitry and corresponding dissipated power are dominated by the contribution from 4096 ADC. Assuming t_{meas} of 80 *ns* for forward and backward cycles, ADCs operating with 9 *bit* resolution at 12.5 *MSamples/sec* are required. The state-of-the-art SAR-ADC (Jonsson, 2011a,b) that can provide this performance, occupy an area of 0.0256 mm^2 and consume 0.24 mW power that results in a total area of 104 mm^2 and a total power of 1 W for an array of 4096 ADCs. This area is much larger than the RPU array itself, therefore it is reasonable to time-multiplex the ADCs between different columns/rows by increasing the sampling rate while keeping total power unchanged. Assuming each ADC is shared by 64 columns (or rows), the total ADC area can be reduced to 1.64 mm^2 with each ADC running at about 800 *MSamples/sec*. Since we assume that RPU device arrays are built on the intermediate metal levels on top of peripheral CMOS circuitry, the total tile area is defined by the RPU array area of 2.68 mm^2 that leaves about 1.0 mm^2 for other circuitry that also can be area optimized. For example, the number of random number generators used to translate binary data to stochastic bit stream can be significantly reduced to just 2 as no operations are performed on streams generated within columns (or rows) and evidenced by no additional error penalty for corresponding classification test (data not shown). Total area of a single tile therefore is 2.68 mm^2, while the total power dissipated by both RPU arrays and all peripheral circuitry (ADCs, opamps, STR) can be estimated as 2.0 W, assuming 0.7 W reserved for op-amps and STRs.

The number of weight updates per second on a single tile can be estimated as 839 *TeraUpdates/s* given the 20 *ns* duration of the update cycle and 4096 × 4096 array size. This translates into power efficiency of 419 *TeraUpdates/s/W* and area efficiency of 319 *TeraUpdates/s/mm*^{2}. The tile throughput during the forward and backward cycles can be estimated as 419 *TeraOps/s* given 80 *ns* for forward (or backward) cycle with power and area efficiencies of 210 *TeraOps/s/W* and

156 $TeraOps/s/mm^2$, respectively. These efficiency numbers are about 5 orders of magnitude better than state-of-the-art CPU and GPU performance metrics (Gokhale et al., 2014).

The power and area efficiencies achieved for a single tile will inevitably degrade as multiple tiles are integrated together as a system-on-chip. As illustrated in **Figure 5C**, additional power and area should be reserved for programmable NLF circuits, on-chip communication via coherent bus or network-on-chip (NoC), off-chip I/O circuitry, etc. Increasing the number of tiles on a chip will first result in an acceleration of a total chip throughput, but eventually would saturate as it will be limited either by power, area, communication bandwidth or compute resources. State-of-the-art high-performance CPU (IBM Power8 12-core CPU, Stuecheli, 2013) or GPU (NVidia Tesla K40 GPU, NVIDIA, 2012) can be taken as a reference for estimation of the maximum area of 600 mm^2 and power of 250 W on a single chip. While power and area per tile are not prohibitive to scale the number of tiles up to 50 to 100, the communication bandwidth and compute resources needed for a system to be efficient might be challenging.

Communication bandwidth for a single tile can be estimated assuming 5 *bit* input and 9 *bit* output per column (or row) for forward (or backward) cycles that give in total about 90 *GB/s* unidirectional bandwidths that will also satisfy the update cycle communication requirements. This number is about 3 times less than the communication bandwidth in IBM Power8 CPU between a single core and a nearby L2 cache (Stuecheli, 2013). State-of-the-art on-chip coherent bus (over three *TB/s* in IBM Power8 CPU, Stuecheli, 2013) or NoC (2.5 *TB/s* in Chen G. et al., 2014) can provide sufficient communication bandwidth between distant tiles.

Compute resources needed to sustain $O(1)$ time complexity for a single tile can be estimated as 51 *GigaOps/s* assuming 80 *ns* cycle time and 4096 numbers generated at columns or rows. To support parallel operation of n tiles, compute resources need to be scaled by $O(n)$ thus limiting the number of tiles that can be active at a given time to keep the total power envelop on a chip below 250 W. For example, a single core of IBM Power8 CPU (Stuecheli, 2013) can achieve about 50 $GigaFLOP/s$ that might be sufficient to support one tile, however the maximum power is reached just for 12 tiles assuming 20 W per core. Corresponding power efficiency for this design point (Design 1 in **Table 1**) would be 20 $TeraOps/s/W$. Same compute resources can be provided by 32 cores of state-of-the-art GPU (NVIDIA, 2012), but with better power efficiency thus allowing up to 50 tiles to work in parallel. Corresponding power efficiency for this design (Design 2 in **Table 1**) would be 84 $TeraOps/s/W$. Further increase in the number of tiles that can operate concurrently can be envisioned by designing specialized power and area efficient digital circuits that operate fixed point numbers with limited bit resolution. An alternative design (Design 3 in **Table 1**) can be based on just a few compute cores that can process the tile data sequentially in order to fit larger numbers of tiles to deal with larger network sizes. For example, a chip with 100 tiles and a single 50 *GigaOps/s* compute core will be capable of dealing with networks with as many as 1.6 billion weights and dissipate only about 22 W assuming 20 W from compute core and communication bus and just 2 W for RPU tiles since only one is active at any given time. This gives a power efficiency of 20 $TeraOps/s/W$ that is four orders of magnitude better than state-of-the-art CPU and GPU.

DISCUSSION

We proposed a concept of RPU devices that can simultaneously store and process data locally and in parallel, thus potentially providing significant acceleration for DNN training. The tolerance of the training algorithm to various RPU device and system parameters as well as to technological imperfections and different sources of noise has been explored. This analysis allows to define a list of specifications for RPU devices summarized in **Table 2**. Current contenders for RPU devices based on existing NVM technologies might not necessary satisfy all the criteria simultaneously. However, we believe that the results of **Table 2** can be used as a guide for a systematic search for new physical mechanisms, materials and device designs to realize the RPU device concept with realistic CMOS-compatible technology.

We also presented an analysis of various system designs based on the RPU array concept that can potentially provide many orders of magnitude acceleration of deep neural network training while significantly decreasing required power and computer hardware resources. The results are summarized in **Table 1**. This analysis shows that, depending on the network size, different design choices for the RPU accelerator chip can be made that trade power and acceleration factor.

The proposed accelerator chip design of **Figure 5C** is flexible and can accommodate different types of DNN architectures beyond fully connected layers with similar acceleration factors. For example, convolutional layers can be also mapped to an RPU array in an analogous way. In this case, instead of performing a vector-matrix multiplication for forward and backward cycles, an array needs to perform a matrix-matrix multiplication that can be achieved by feeding the columns of the input matrix

TABLE 1 | Summary of comparison of various RPU system designs and state-of-the-art CPU and GPU.

System	Throughput (TeraOps/s)	Power (W)	Power efficiency (GigaOps/s/W)	Network size (number of weights)	Acceleration vs. CPU
CPU Power8 12 Cores	0.676	250	2.7	–	1
GPU NVidia Tesla K40	4.3	242	17.8	–	6.4
Design 1	5000	250	20,100	200 M	7400
Design 2	21,000	250	83,800	840 M	31,000
Design 3	420	22	19,000	1680 M	620

TABLE 2 | Summary of RPU device specifications.

Specs	Parameter	Value	Tolerance
Pulse duration		1 *ns*	
Operating voltage	$\pm V_S$	1 *V*	
Maximum device area		0.04 μm^2	
Average device resistance	R_{device}	24 $M\Omega$	7 $M\Omega$
Maximum device resistance	$\max\left(g_{ij}\right)$	112 $M\Omega$	7 $M\Omega$
Minimum device resistance	$\min\left(g_{ij}\right)$	14 $M\Omega$	7 $M\Omega$
Resistance on/off ratio	$\max\left(g_{ij}\right)/\min\left(g_{ij}\right)$	8	
Resistance change at $\pm V_S$	$\triangle g^{\pm}_{min}$	100 $K\Omega$	30 $K\Omega$
Resistance change at $\pm V_S/2$		10 $K\Omega$	
Storage capacity	$\left(\max\left(g_{ij}\right)-\min\left(g_{ij}\right)\right)/\triangle g_{min}$	1000 levels	
Device up/down asymmetry*	$\triangle g^{+}_{min}/\triangle g^{-}_{min}$	1.05	2%

Note that these numbers are derived from the radar diagram in ***Figure 4A*** *and correspond to the shaded area.* *Global asymmetry in up/down responses can be to a large extend compensated by proper adjustment of pulse widths and/or pulse amplitude.

serially into the columns of the RPU array. In addition, peripheral NLF circuits need to be reprogrammed to perform not only calculation of activation functions, but also max-pooling and sub-sampling. The required connectivity between layers can be achieved by reprogramming tile addresses in a network. The update cycle for a convolutional layer would require computation of the product of two matrixes that are used during the forward and backward cycles. This can be achieved by serially feeding the columns of the input matrix and the columns of the error matrix simultaneous to the RPU array. During the update cycle each RPU device performs a series of local multiplication and summation operations and hence calculates the product of the two matrixes. We note that all three cycles on the RPU array are similar for both convolutional and fully connected layers and do not require reprograming. Indeed, a convolutional layer can be viewed as a fully connected layer with a mini-batch size larger than unity. We emphasize that the throughput of a RPU accelerator chip is independent of the DNN architecture and the mini-batch size and therefore should achieve similar acceleration factors for similar RPU array sizes. However, the RPU device and system specifications should be reconsidered for different DNN architectures and datasets using the approach described in the paper.

Most of the recent DNN architectures are based on a combination of many convolutional and fully connected layers with a number of parameters of the order of a billion. Our analysis demonstrates that a single RPU accelerator chip can be used to train such a large DNNs. Problems of the size of ImageNet classification that currently require days of training on multiple GPUs (Le et al., 2012) can take just less than a minute on a single RPU accelerator chip.

AUTHOR CONTRIBUTIONS

TG conceived the original idea, TG and YV developed methodology, analyzed and interpreted results, drafted and revised manuscript.

ACKNOWLEDGMENTS

We thank Seyoung Kim, Jonathan Proesel, Mattia Rigotti, Wilfried Haensch, Geoffrey Burr, Michael Perrone, Bruce Elmegreen, Suyog Gupta, Ankur Agrawal, Mounir Meghelli, Dennis Newns, and Gerald Tesauro for many useful discussions and suggestions.

REFERENCES

Alaghi, A., and Hayes, J. (2013). Survey of stochastic computing. *ACM Trans. Embed. Comput. Syst.* 12, 1–19. doi: 10.1145/2465787.2465794

Arima, Y., Mashiko, K., Okada, K., Yamada, T., Maeda, A., Notani, H., et al. (1991). A 336-neuron, 28 K-synapse, self-learning neural network chip with branch-neuron-unit architecture. *IEEE J. Solid State Circuits* 26, 1637–1644.

Bi, G. Q., and Poo, M. M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472.

Burr, G. W., Shelby, R. M., di Nolfo, C., Jang, J. W., Shenoy, R. S., Narayanan, P., et al. (2014). "Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses), using phase-change memory as the synaptic weight element,"in *2014 IEEE International, Electron Devices Meeting (IEDM)* (San Francisco, CA).

Burr, G. W., Narayanan, P., Shelby, R. M., Sidler, S., Boybat, I., di Nolfo, C., et al. (2015). "Large-scale neural networks implemented with nonvolatile memory as the synaptic weight element: comparative performance analysis (accuracy, speed, and power)," in *IEDM (International Electron Devices Meeting)* (Washington, DC).

Chen, G., Anders, M. A., Kaul, H., Satpathy, S. K., Mathew, S. K., Hsu, S. K., et al. (2014). "A 340 mV-to-0.9 V 20.2 Tb/s source-synchronous hybrid packet/circuit-switched 16 × 16 network-on-chip in 22 nm tri-gate CMOS," in *2014 IEEE International Soild-State Circuits Conference Digest of techincal Papers (ISSCC)* (San Francisco, CA), 276–277.

Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., et al. (2014). "DaDianNao: a machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture* (Cambridge, UK), 609–622.

Chua, L. O. (1971). Memristor - the missing circuit element. *IEEE Trans. Circuit Theory* 18, 507–519. doi: 10.1109/TCT.1971.1083337

Coates, A., Huval, B., Wang, T., Wu, D. J., Ng, A .Y., and Catanzaro, B. (2013). *Deep Learning with COTS HPC Systems*. Atlanta, GA: ICML.

Gaines, B. R. (1967). "Stochastic computing," in *Proceedings of the AFIPS Spring Joint Computer Conference* (Atlantic City, NJ), 149–156.

Gokhale, V., Jin, J., Dundar, A., Martini, B., and Culurciello, E. (2014). "A 240 g-ops/s mobile coprocessor for deep neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (Columbus, OH), 696–701.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. (2015). Deep learning with limited numerical precision. arXiv:1502.02551 [cs.LG].

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A., Jaitly, N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.* 29, 82–97. doi: 10.1109/MSP.2012.2205597

Indiveri, G., Linares-Barranco, B., Legenstein, R., Deligeorgis, G., and Prodromakis, T. (2013). Integration of nanoscale memristor synapses in neuromorphic computing architectures. *Nanotechnology* 24:384010. doi: 10.1088/0957-4484/24/38/384010

Jackson, B. L., Rajendran, B., Corrado, G. S., Breitwisch, M., Burr, G. W., Cheek, R., et al. (2013). Nanoscale electronic synapses using phase change devices. *ACM J. Emerg. Technol. Comput. Syst.* 9, 1–20. doi: 10.1145/2463585.2463588

Jensen, K., Gaudet, V. C., and Levine, P. M. (2013). "Noise analysis and measurement of integrator-based sensor interface circuits for fluorescence detection in lab-on-a-chip applications," in *International Conference on Noise and Fluctuation* (Montpellier).

Jo, S. H., Chang, T., Ebong, I., Bhadviya, B. B., Mazumder, P., and Lu, W. (2010). Nanoscale memristor device as synapse in neuromorphic systems. *Nano Lett.* 10, 1297–1301. doi: 10.1021/nl904092h

Jonsson, B. E. (2011a). "An empirical approach to finding energy efficient ADC architectures," in *2011 International Workshop on ADC Modelling, Testing and Data Converter Analysis and Design and IEEE 2011 ADC Forum* (Orvieto), 132–137.

Jonsson, B. E. (2011b). "Area Efficiency of ADC Architectures," in *2011 20th European Conference on Circuit Theory and Design (ECCTD)* (Linkoping), 560–563.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks," in *Neural Information Processing Systems* (Lake Tahoe, NV), 1097–1105.

Kuzum, D., Yu, S., and Wong, H. S. (2013). Synaptic electronics: materials, devices and applications. *Nanotechnology* 24:382001. doi: 10.1088/0957-4484/24/38/382001

Le, Q. V., Ranzato M. A., Monga, R., Devin, M., Chen, K., Corrado, G. S., et al. (2012). "Building high-level features using large scale unsupervised learning," in *International Conference on Machine Learning* (Edinburg).

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

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proce. IEEE* 86, 2278–2324. doi: 10.1109/5.726791

Lehmann, C., Viredaz, M., and Blayo, F. A. (1993). generic systolic array building block for neural networks with on-chip learning. *IEEE Trans. Neural Netw.* 4, 400–407.

Li, B., Wang, Y., Wang, Y., Chen, Y., and Yang, H. (2014). "Training itself: mixed-signal training acceleration for memristor-based neural network" in *19th Asia and South Pacific Design Automation Conference (ASP-DAC)* (Suntec City).

Merkel, C., and Kudithipudi, D. (2014). "A stochastic learning algorithm for neuromemristive systems," in *27th IEEE International System-on-Chip Conference (SOCC)*, (Las Vegas, NV), 359–366.

NVIDIA (2012). NVIDIA's next generation CUDA compute architecture: Kepler GK110. *Whitepaper*.

O'Connor, P., and Welling, M. (2016). Deep spiking networks. arXiv:1602.08323 [cs.NE].

Poppelbaum, W. J., Afuso, C., and Esch, J. W. (1967). "Stochastic computing elements and systems," in *Proceedings of the AFIPS Fall Joint Computer Conference* (Anaheim, CA), 635–644.

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

Rumelhart, D. E., Hinton, G. E., and Willims, R. J. (1986). Learning representations by back-propagating errors. *Nature* 323, 533–536.

Saighi, S., Mayr, C. G., Serrano-Gotarredona, T., Schmidt, H., Lecerf, G., Tomas, J., et al. (2015). Plasticity in memristive devices for spiking neural networks. *Front. Neurosci.* 9:51. doi: 10.3389/fnins.2015.00051

Seo, J., Lin, B., Kim, M., Chen, P., Kadetotad, D., Xu, Z., et al. (2015). On-chip sparse learning acceleration with CMOS and resistive synaptic devices. *IEEE Trans. Nanotechnol.* 14, 969–979. doi: 10.1109/TNANO.2015.2478861

Simonyan, K., and Zisserman, A. (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. San Diego, CA: ICLR.

Soudry, D., Di Castro, D., Gal, A., Kolodny, A., and Kvatinsky, S. (2015). Memristor-based multilayer neural networks with online gradient descent training. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 2408–2421. doi: 10.1109/TNNLS.2014.2383395

Steinbuch, K. (1961). Die lernmatrix. *Kybernetik* 1, 36–45.

Strukov, D., Snider, G., Stewart, D., and Williams, R. (2008). The missing memristor found. *Nature* 453, 80–83. doi: 10.1038/nature06932

Stuecheli, J. (2013). "Next Generation POWER microprocessor," in *Hot Chips Conference* (Palo Alto, CA).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., et al. (2015). *Going Deeper with Convolutions*. Boston, MA: CVPR.

Wu, R., Yan, S., Shan, Y., Dang, Q., and Sun, G. (2015). Deep image: scaling up image recognition. arXiv:1501.02876 [cs.CV].

Xu, Z., Mohanty, A., Chen, P., Kadetotad, D., Lin, B., Ye, J., et al. (2014). Parallel programming of resistive cross-point array for synaptic plasticity. *Procedia Comput. Sci.* 41, 126–133. doi: 10.1016/j.procs.2014.11.094

Yu, S., Gao, B., Fang, Z., Yu, H., Kang, J., and Wong, H. S. (2013). A low energy oxide−based electronic synaptic device for neuromorphic visual systems with tolerance to device variation. *Adv. Mater.* 25, 1774–1779. doi: 10.1002/adma.201203680

Yu, S., Chen, P., Cao, Y., Xia, L., Wang, Y., and Wu, H. (2015). "Scaling-up resistive synaptic arrays for neuro-inspired architecture: challenges and prospect," in *International Electron Devices Meeting (IEDM)* (Washington, DC), 451–454. doi: 10.1109/iedm.2015.7409718

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.