Reducing Circuit Design Complexity for Neuromorphic Machine Learning Systems Based on Non-Volatile Memory Arrays

Pritish Narayanan*, Lucas L. Sanches*, Alessandro Fumarola*, Robert M. Shelby*, Stefano Ambrogio*, Junwoo Jang*, Hyunsang Hwang†, Yusuf Leblebici‡, and Geoffrey W. Burr*
*IBM Research – Almaden San Jose, CA 95120 Email: pnaraya@us.ibm.com
†Department of Material Science and Engineering, Pohang University of Science and Technology, Pohang 790-784, Korea
‡EPFL, Lausanne, CH–1015, Switzerland

***Abstract*—Machine Learning (ML) is an attractive application of Non-Volatile Memory (NVM) arrays [1,2]. However, achieving speedup over GPUs will require minimal neuron circuit sharing and thus highly area-efficient peripheral circuitry, so that ML reads and writes are massively parallel and time-multiplexing is minimized [2]. This means that neuron hardware offering full 'software-equivalent' functionality is impractical. We analyze neuron circuit needs for implementing back-propagation in NVM arrays and introduce approximations to reduce design complexity and area. We discuss the interplay between circuits and NVM devices, such as the need for an occasional RESET step, the number of programming pulses to use, and the stochastic nature of NVM conductance change. In all cases we show that by leveraging the resilience of the algorithm to error, we can use practical circuit approaches yet maintain competitive test accuracies on ML benchmarks.**

I. Introduction

Non-Volatile Memory-based crossbar arrays can be used in neuromorphic non-von Neumann computing schemes [1,2], for example in multi-layer perceptrons trained using back-propagation (Fig. 1) [3]. Here, pairs of conductances serve as analog programmable weights and computation (specifically highly efficient multiply-accumulate operations) can be achieved at the location of the data in a massively-parallel analog operation along rows and columns of the array (Fig. 2).

To obtain significant speedup over conventional hardware (e.g. GPUs), it is essential that large portions of the array can operate in parallel with limited time-multiplexing. This necessitates the use of low-area neuron circuits to minimize 'circuit-sharing' (c_s parameter in Figs. 2, 4) [2]. Here, we evaluate a series of circuit tradeoffs where we replace 'exact' but impractical neuron functionalities with approximate but area-efficient alternatives. We show that by leveraging the error tolerance of ML algorithms, such area-efficient neuron designs are able to maintain competitive training performance similar to the same deep neural network implemented entirely in software (also referred to in the text as 'complete software implementation), even in the presence of imperfect devices.

II. Preliminaries

Supervised training of a fully-connected deep neural network (DNN) proceeds in 3 phases. During **Forward Propagation** (Fig. 3, middle), training data (e.g. an input image) are propagated from left-to-right to produce a set of outputs that is the network's 'best guess' at classifying that image. The forward-propagate operation at every layer involves a multiply-accumulate operation along one direction (say, the columns) of the crossbar array, followed by a non-linear 'squashing' or activation function that generates the neuron activations of the next stage.

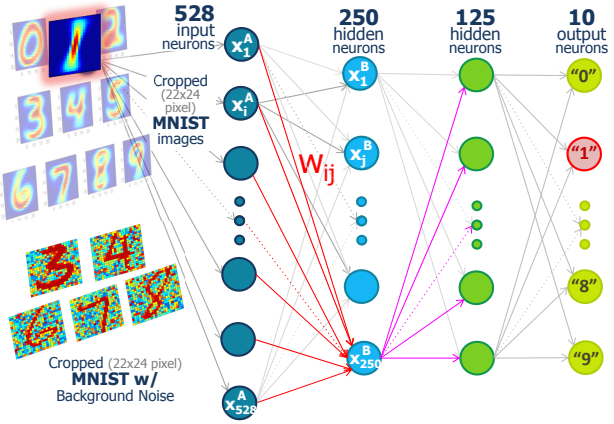

Fig. 1 3-Layer Neural Network used in simulation studies with 528 input neurons, 10 output neurons and two hidden layers with 250 and 125 neurons each [1-3]. Benchmarks include the MNIST dataset of handwritten digits[7], and the MNIST dataset with background noise[8].

Typical squashing functions used for Deep Neural Networks include logistic, hyperbolic tangent ($tanh$) or unbounded Rectified Linear Units (ReLU), all of which require extensive floating point arithmetic and/or look-up tables. Unfortunately, to interface analog crossbar arrays to any such digital compute units, precise analog-to-digital (A-to-D) and digital-to-analog (D-to-A) conversion would be required (Fig. 4, right), making an exact neuron implementation highly infeasible. For example, even highly-efficient A-to-D and D-to-A circuits require an area of at least $\sim 1.5 \times 10^6 F^2$ [4]. For comparison, a piecewise linear (PWL) approximation of the squashing function would be fairly straightforward to implement in the analog domain using a dedicated comparator and a common ramp voltage.

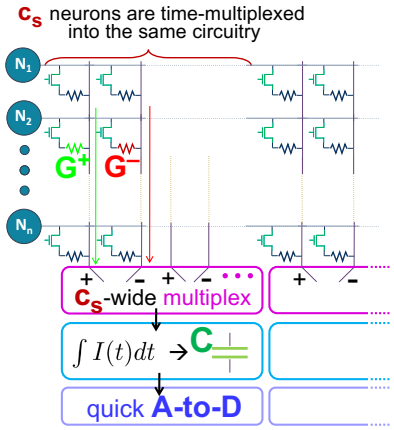

Fig. 2 Neural network weights are encoded into pairs of NVM-conductances (G^+ and G^-). Multiply–accumulate operation is achieved through Ohm's law and current summation. c_s is a measure of how many rows/columns share the same neuron circuitry[2].

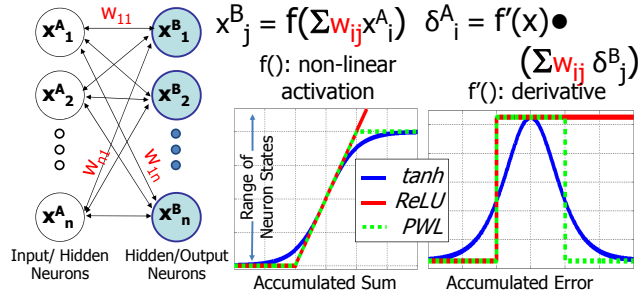

Fig. 3 During forward propagation (middle), every downstream neuron circuit must accumulate weighted contributions from upstream neurons, and handle the non-linear 'squashing', or activation function $f()$. During reverse propagation (right), each neuron must accumulate weighted errors from downstream neurons, scaled by the derivative of the squashing function.

The precision with which the neuron activation is stored and transmitted between stages is also an important design parameter. Lower precision implies fewer digital bits and/or smaller capacitors to hold a smaller number of distinct neuron states, thereby also reducing circuit area.

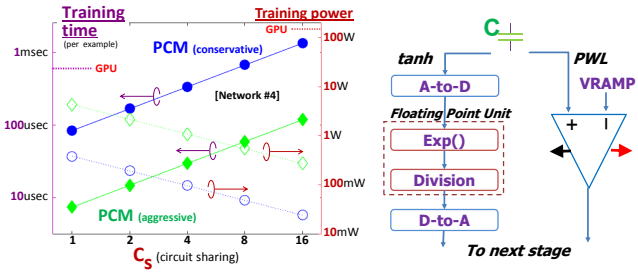

Fig. 4 (Left) Achieving speed-up benefits over GPUs requires a low value of c_s, enabling large parts of the array to operate in parallel [2]. (Right) Implementing exact squashing function (e.g. $tanh$) would be extremely area-inefficient; Piece-wise Linear (PWL) is a much more efficient alternative.

During **Reverse Propagation** (Fig. 3, right), errors calculated from comparison against ground-truth labels are back-propagated from the outputs to the inputs. Back-propagation occurs in a direction orthogonal to forward propagation (e.g. integration along rows). In addition to the same precision consideration, back-propagation calls for the derivative of the forward-propagated neuron activation to be incorporated into the error term. The PWL derivative is a step function which can be implemented with simple circuitry, whereas a more precise $tanh$ derivative would be complex and area-inefficient.

Parallel operation also needs a **crossbar-compatible weight update** rule (Fig. 5). Here, instead of calculating an exact weight gradient by exchanging information across pairs of upstream and downstream neurons (a process that would be extremely slow across possibly millions of synapses), each neuron fires a series of pulses based only on local knowledge of either neuron activation (for the upstream neuron) or back-propagated error (for the downstream neuron) , and overlap of these pulses achieves the weight change [1]. In fully bi-directional NVMs with linear slope, device conductances can be both increased and decreased in a smooth fashion. Each weight change in this case is implemented in two phases with both a SET and a RESET (e.g. a positive weight change is implemented as a G^+ increase and a G^- decrease). This crossbar-compatible weight update rule has been shown to have no effect on neural network performance vs. the exact weight update [1]. For this operation, the number of pulses to be fired and the resulting response of the NVM devices are important considerations for peripheral circuitry.

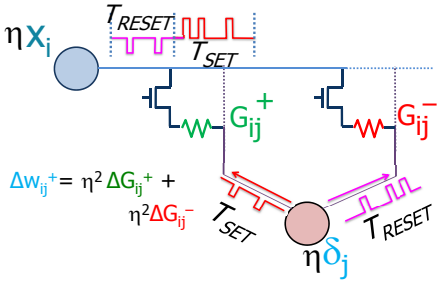

Fig. 5 During parallel weight update, upstream and downstream neurons fire pulses based on local knowledge of x and δ [1,3,6]. Overlap between these pulses updates conductances (in 'Fully Bi-directional' mode, both G^+ and G^- [3]).

III. Results

We consider the MNIST dataset of handwritten digits [5], adapting the same DNN computer simulation matched to PCM experiments from [1]. Unless otherwise specified, we consider a reduced training set (5000 examples), yet the full test set of 10000 examples. For the network in Fig. 1, the baseline implementation achieves 93.7% and 94.3% test accuracy for the $tanh$ and ReLU activation functions respectively, using the crossbar-compatible weight update rule. In comparison, the software implementation of $tanh$ with the exact weight update achieves 94.5% test accuracy. The PWL training (99%) and test accuracies (93.7%) with bi-directional NVM and crossbar-compatible weight update are quite comparable with the above numbers.

Leaky derivatives are often needed in software neural nets, to keep large error terms from being zeroed out during back-propagation (vanishing gradient problem). An approximate version of this is implementable by using a non-zero 'low' derivative value, which allows further tuning of test accuracy (Fig. 6, left). While in this particular case, the zero derivative offers the best result, this is not generally true for all problems and having the flexibility to implement non-zero derivatives is essential. In a digital implementation, one would be restricted to powers of 2 in order to implement the multiplication as a bit-shift operation. In an analog implementation, much more flexibility and tunability is available.

The precision requirements for neurons during forward and reverse propagate were studied. As shown in Fig. 6(right), test accuracy on MNIST shows little to no degradation down to 6 distinct neuron levels for both the $tanh$ and PWL implementations. Such a high tolerance to imprecision in the storage and transfer of neuron values will be extremely helpful in further reducing design complexity.

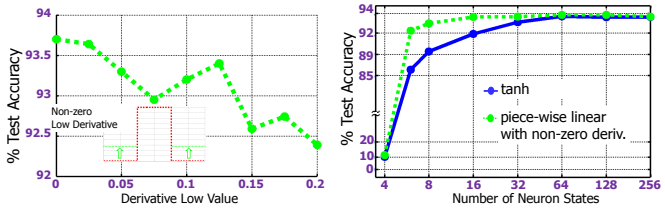

Fig. 6 (Left) – Optimizing the low derivative value may be needed for further improvements in test accuracy, yet requires some circuit complexity to approximate the multiplication operation. (Right) – Neural Networks maintain high test accuracies on MNIST even with a small number of distinct neuron states, enabling area-efficient peripheral circuitry.

The x and δ values, multiplied by the appropriate learning rates, would need to be quantized and stored for the subsequent weight update step. Given that binary registers are large circuits and total area increases at least linearly with the number of pulses, it would be ideal if the weight update could be implemented with a small number of pulses. From Fig. 7, we see that 1-4 pulses per weight update event is more than sufficient to achieve high test accuracy on MNIST. The number of programming pulses is also strongly correlated to the change in conductance per pulse (ΔG). Given that NVM device conductances are bounded, there exists a device-circuit tradeoff. Devices with a large number of steps between min and max conductance (e.g. small ΔG) could utilize more precision in the weight update, leading to better test accuracies. However, with a larger ΔG, it may actually be preferable to use fewer pulses for programming, as too many pulses will tend to over-correct the weights. Fig. 8 shows simulation results of using all of these circuit approximations on the full MNIST dataset with 60000 images. A crossbar-compatible weight update rule where up to 2 pulses can be fired per weight update is used, along with PWL and zero derivative. A test accuracy of 96.32% is achieved, showing excellent correlation with the complete software implementation of the same network [1].

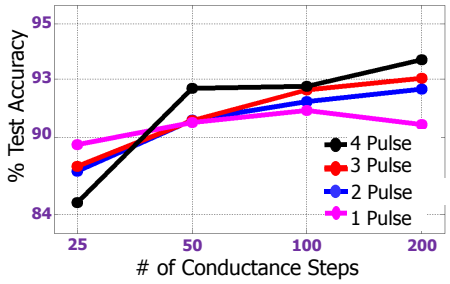

Fig. 7 A small number (1–4) of programming pulses is sufficient to achieve high test accuracy. Investing circuit resources to increase the maximum number of programming pulses only helps for devices with large dynamic range.

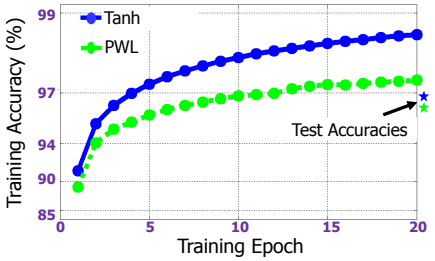

Fig. 8 Even on the full MNIST 60K dataset, area-efficient circuitry can train a network based on bidirectional NVM and approach the performance of the complete software implementation.

Many NVM device candidates such as Phase Change Memory (PCM) and filamentary Resistive RAM (RRAM) show incremental conductance change only in one direction, with an abrupt conductance change in the opposite direction. Neural network circuits built for such devices need to incorporate a fourth mode of operation called **Occasional RESET**[1] [1]. During training, since conductances can only increase, eventually both G^+ and G^- would saturate, causing the net weight to go to zero and the network to freeze (i.e. stop learning). To prevent this, training is occasionally suspended, and individual conductances are sensed. Any pair of conductances found in a pre-defined 'danger zone' (right edges of the 'G-diamond' (Fig. 9, left)) are RESET, followed by SET pulses applied to G^+ or G^- as appropriate to attempt to restore the same weight. This leaves the conductance pair at the left edges of the G-diamond, making it trainable again.

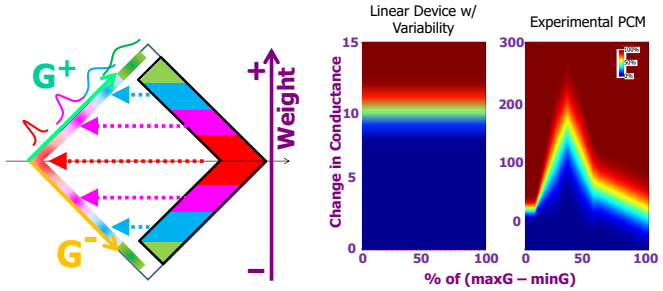

Fig. 9 (Left): A G-diamond shows values of synaptic weight (height above center line) and individual conductances (projections to tilted axes). In NVMs with highly asymmetric conductance response, conductances near the right edges of the G-diamond cannot participate in further training, and need to be occasionally RESET. A number of pulses, targeting the center of a bin, are fired to approximate the weight. Without verification of the final conductance value, there is no guarantee that weights will land in the correct bin; larger weights are subject to higher variation. Middle: a linear device with 10% standard deviation in every conductance change; Right: a model of PCM conductance data matched to experiments over 33K PCM devices and 31M partial-SET pulses [1].

Circuitry for occasional RESET requires sensing individual conductances, applying RESET pulses and potentially verifying that an approximate weight was restored. 'Binning' weights into coarse regions has several advantages over precision sensing, including 1) simplifying sensing requirements 2) reducing wall clock time, as it would be easier to land a weight in a particular bin and 3) preserving device endurance.

To study the impact of binning during occasional RESET, we consider two device models with conductance change in one direction. Both device characteristics are shown in

[1]RESET refers to the abrupt and large decrease in conductance of a PCM element. However, this discussion is equally relevant for some filamentary RRAM, which have Occasional SET with partial RESETs.

Fig. 9 in the form of jump tables [1], which plot distribution of conductance change on the Y-axis as a function of the instantaneous conductance. The first device is a linear device with a nominal conductance change of 10 units (as compared to a conductance range of ∼1000 units), and a standard deviation of 1 unit. The second is the PCM device model from our demonstration of a ∼165000 synapse neural network [1].

In the first RESET study, we assume a uniform distribution of conductances within each bin after the RESET+ Partial SET operation. We consider both the MNIST dataset [5] and a more challenging benchmark, which adds background noise to MNIST [6]. Baseline test accuracy for the latter with the complete software implementation is ∼76%. From Fig. 10, we find that both the linear device and our experimental PCM device can achieve competitive ML performance on both benchmarks, with as few as 8 bins during occasional RESET. Even with 2 or 4 bins, high test accuracies can be obtained. These runs also incorporate all previous approximations discussed, including PWL, step derivative and crossbar-compatible weight update.

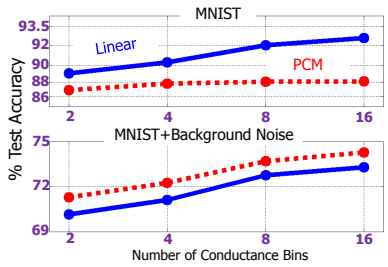

Fig. 10 Test accuracy vs. precision in sensing and readjusting of individual conductances. Fewer conductance bins will reduce circuit requirements.

One further simplification that would considerably reduce circuit complexity would be to eliminate the verification step entirely. In this scenario, based on the statistics of the devices, a number of partial SET pulses are fired post-RESET, to target a particular bin. However, no guarantee is made that the final conductance value obtained actually falls within that bin. Under these conditions, larger conductances (which require more programming steps to reach) are subject to more variation than smaller conductances (Fig. 9). Results (Fig. 11) show that the the PCM device test accuracy is slightly degraded with MNIST (∼88% with 16 bins) but is more significantly degraded (∼34%) with the MNIST background noise benchmark. This implies that SET verification will continue to be necessary with devices that exhibit these levels of stochastic variability. However, with better devices that offer more linearity and control over variability, verification of individual conductances may no longer be needed during the training process.

Figure 12 shows results for MNIST with 60k training images and 10k test images. Three cases are shown in both plots - a baseline with the device characteristics in Fig. 9, a second curve with PWL approximation but perfect RESET, and a third curve with both PWL and uniform RESET-with-verify assumptions. We find that our circuit simplifications do not significantly deteriorate performance compared to the baseline, demonstrating that there is a viable path towards area-efficient circuitry for ML applications, taking into account both device requirements and system performance targets. However, baselines in both cases are below what a full software implementation can achieve (∼97% for this network). Therefore, circuit techniques presented here will need to be combined with device optimizations for better linearity and tighter variability control for practical, larger-scale networks.

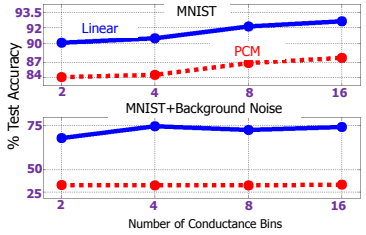

Fig. 11 Test accuracy vs. # of conductance bins: open-loop SET operation (without verify) could work with a linear device and tightly controlled variability. However, current PCM devices will need iterative closed-loop re-programming.

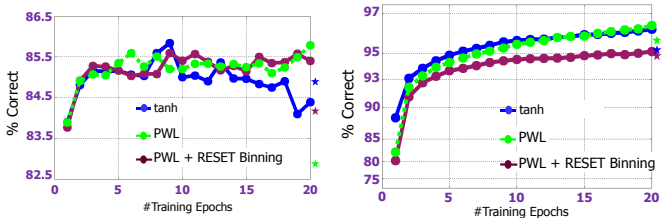

Fig. 12 Training accuracy on with MNIST 60K. Stars are final test accuracy. (Left) – PCM, test accuracies with approx. squashing function+ binning are comparable to exact versions, but lower than with more linear models. (Right) – linear device: test accuracy with PWL + perfect reset (95.75%), and PWL + binning (94.83%) are comparable to exact neuron functionality (95.21%).

IV. Conclusion

In non-von Neumann computing architectures with NVM-based crossbar arrays, neuron circuitry must support a) all standard neural network operations and b) special constraints imposed by the finite conductance ranges of non-volatile memory devices. Several approaches to reduce design complexity and enable speedup over GPUs were discussed. In fully-bidirectional NVMs, the combination of algorithm approximations — such as PWL, step-derivative, limited number of neuron states and careful selection of programming pulses — can achieve competitive ML performance. Devices with abrupt conductance change in one direction require coarse binning of weights during Occasional RESET. Experimental PCM devices may additionally need SET+verify capability, although this could be eliminated with continued device improvements.

References

[1] G. W. Burr et. al., Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses), using phase-change memory as the synaptic weight element *IEDM Tech. Digest*, 29.5 2014.

[2] G. W. Burr et. al., Large–scale neural networks implemented with nonvolatile memory as the synaptic weight element: comparative performance analysis (accuracy, speed, and power) *IEDM Tech. Digest*, 4.4 2015.

[3] D. E. Rumelhart et. al., A General Framework for Parallel Distributed Processing *Parallel Dist Processing, MIT Press*, 1986, pp. 45–76.

[4] L. Kull et. al., A 3.1mW 8b 1.2GS/s single-channel asynchronous SAR ADC with alternate comparators for enhanced speed in 32nm digital SOI CMOS *Proc. of ISSCC*, 26.4 2013.

[5] Y. LeCun et. al., Gradient-Based Learning Applied to Document Recognition *Proc. of IEEE*, 86(11), 1998, pp. 2278–2324.

[6] H. Larochelle et. al., An empirical evaluation of deep architectures on problems with many factors of variation *Proc. of ICML.*, pp. 473–478 (2007).