Improved Deep Neural Network hardware-accelerators based on Non-Volatile-Memory: the Local Gains technique

Irem Boybat[†‡], Carmelo di Nolfo[*], Stefano Ambrogio[*], Martina Bodini[*†], Nathan C. P. Farinha[*§], Robert M. Shelby[*], Pritish Narayanan[*], Severin Sidler[†], Hsinyu Tsai[*], Yusuf Leblebici[†], and Geoffrey W. Burr[*]

[*]IBM Research–Almaden, 650 Harry Road, San Jose, CA 95120, Tel: (408) 927–1512, Email: *gwburr@us.ibm.com*
[†]EPFL, 1015 Lausanne, Switzerland [‡]IBM Research–Zurich, 8803, Rueschlikon, Switzerland
[§]University of Sao Paulo, Av. Trab. Sao-Carlense, 400 - Parque Arnold Schimidt, Sao Carlos - SP, 13566-590, Brazil

***Abstract*—Cognitive computing – which *learns* to do useful computational tasks from data, rather than by being programmed explicitly – represents a fundamentally new form of computing. However, training Deep Neural Networks (DNNs) calls for repeated exposure to huge datasets, requiring extensive computation capabilities (such as many GPUs) and days or weeks of time. One potential approach for accelerating this process are hardware accelerators for backpropagation training based on analog Non-Volatile Memory (NVM).**

This paper describes a novel Local Gains (LG) method which can increase network accuracy, extend the range of acceptable learning rates, and reduce overall weight-update activity (and thus the corresponding energy consumption). We first analyze the impact of different activation functions and the corresponding dynamic range of input and output neurons. We then show that the use of non-negative neuron-activations offers advantages within a crossbar implementation (without degrading accuracy), by causing the sign of the weight-update to depend only on the sign of the backpropagated error.

Then we introduce LG: a *neuron*-centric (NOT *synapse*-centric) modulation of the learning rate based on the sign of successive weight updates. The concept of Safety Margin (SM) – the margin by which the correct output neuron exceeded (or failed to exceed) the strongest incorrect neuron – is introduced, providing a novel way to gauge the robustness of DNN classification performance. We use device-aware DNN simulations to demonstrate higher accuracy, reduced sensitivity to network hyperparameters, and an overall improved training process, as well as lower network activity and reduced energy consumption.

I. Introduction

Deep Neural Networks (DNNs) represent the current state of the art in cognitive computing [1]. However, training these networks requires extensive computational capabilities, leading to large expenditures of time and energy due to separate location of memory and CPU, typically described as the "Von Neumann bottleneck," Fig. 1a. To accelerate this training process, our group has been investigating neuromorphic hardware accelerators for backpropagation training based on analog Non-Volatile Memory (NVM) [2], [3], which exploit the co-location of memory and computation, as shown in Fig. 1b.

Our previous work has focused on Fully Connected, multi-layer perceptron networks trained with the backpropagation algorithm [4], in which weights are encoded in pairs of Phase

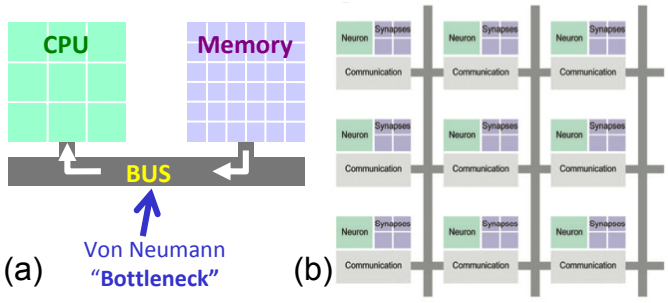

Fig. 1. In the Von Neumann architecture (a), data (both operations and operands) must move to and from the dedicated Central Processing Unit (CPU) along a bus. In contrast, in a Non-Von Neumann architecture (b), distributed computations take place at the location of the data, reducing the time and energy spent moving data around.

Change Memory (PCM) devices [2]. As a result, the multiply–accumulate operation between weights and neuron activations can be performed by parallel read of this analog memory, with Ohm's Law providing the multiply operation and Kirchhoff's Current Law providing the accumulation (summation of products). Fig. 2 shows our prototypical network, with three layers of synapses connected by four neuron layers. We feed this network with images from the MNIST database of handwritten digits [5], cropped to 22 x 24 pixels. The weights are organized in crossbar arrays of PCM and selector (in this case, a MOSFET) pair as shown in Fig. 2, with each weight encoded into the difference between a pair of conductances [6]. We have developed an architectural approach that allows forward-propagation, back-propagation of corrections, and weight update to each be performed by peripheral circuits at the edge of each crossbar array.

Fig. 3 shows our early experimental results of a mixed hardware-software implementation of this three-layer perceptron of 164,885 synapses (Fig. 2), as trained on a 500 x 661 array of mushroom-cell [2], 1T1R PCM devices (180-nm node). As part of this initial work, we also developed a simulator which was able to replicate the experimental results [2]. The resulting training and test accuracy (on the MNIST dataset) was limited to 82% due to the impact of device non-idealities present within the experiment [3]. These

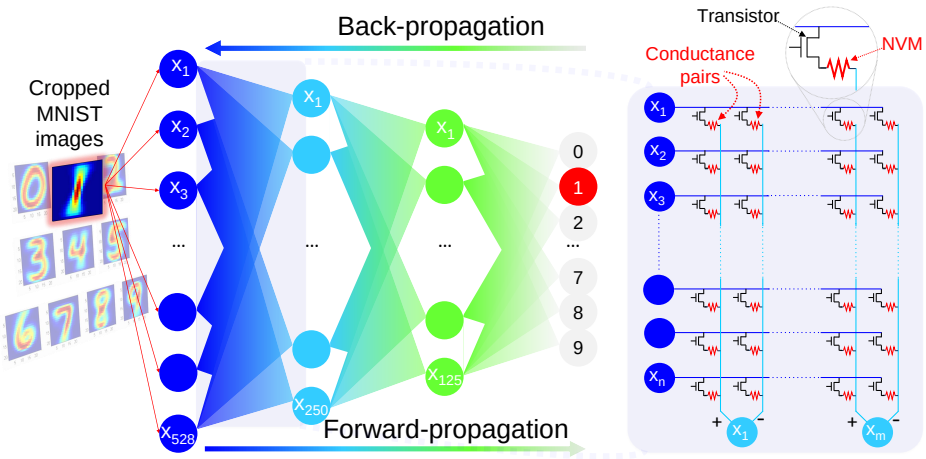

Fig. 2. In forward evaluation of a multi-layer perceptron, each layer of neurons drives the next layer through weights w_{ij} and a nonlinearity $f()$. Input neurons are driven by pixels from successive MNIST images (cropped to 22 x 24 pixels). Every neuron is connected to the other neurons of adjacent layers by networks of programmable synaptic weights, implemented using dense crossbar arrays of nonvolatile memory (NVM) and selector device-pairs.

imperfections were accurately modeled in the simulator by using a "jump-table" [2], [7] – a simulation approach that will be compactly referred to here as the "2–PCM" approach.

Since this original paper, interest in this approach – back-propagation training or subsequent forward-inference-only evaluation of neural networks using large crossbar arrays of analog memory devices – has only increased. Studies of power and speed [8], [9], the impact of device imperfections [7], [10], [11], the impact of circuit approximations [12], [13], and yield [14] have been performed, and connections to ultra-low-power spiking-based networks proposed [15]. Analog non-volatile memory devices ranging from PCM [16] to RRAM [14], [17] to non-filamentary RRAM [18] to 3-terminal devices based on electrochemical intercalation [19] have been proposed and demonstrated.

This paper first analyzes the impact of different activation functions and the corresponding dynamic range of input and output neurons with pure software implementation first (weights encoded as integers) and then with the "2–PCM" approach (weights encoded in a pair of simulated PCMs). We then show that the use of non-negative neuron-activations offers advantages within a crossbar implementation (without degrading accuracy), by causing the sign of the weight-update to depend only on the sign of the backpropagated error. Then, we describe a novel Local Gains (LG) method which improves network accuracy, extends the range of acceptable learning rates, and reduces overall weight-update activity and thus the corresponding energy consumption. We also introduce the Safety Margin (SM) concept to quantify the quality of the network training, and we show some considerations on energy consumption with and without LG.

II. Impact of Activation Function and Input/Output Dynamic Ranges

Fig. 4 shows the computation inside each neuron within the network. The activation x_j^B of the neuron is obtained by summing the overall contributions from the previous layer neurons x_i^A, each multiplied by the corresponding weight w_{ij}. The integrated contribution $\Sigma x_i^A w_{ij}$ is then passed to an hyperbolic tangent function, producing x_j^B. We decided to study the impact of varying the $tanh$ function by using two parameters, α and β, to scale and shift this nonlinear squashing function. Then, we varied the dynamic range of the input, e.g., how we map the values of the pixels of the training and test images to input neuron excitations, with respect to the range of the $tanh$ function. In addition, we focused on the impact of different ranges for the Ground Truth (GT), namely the image label during the training process.

We simulated the same neural network topology as in Fig. 2. Weights were encoded with high precision, using integers

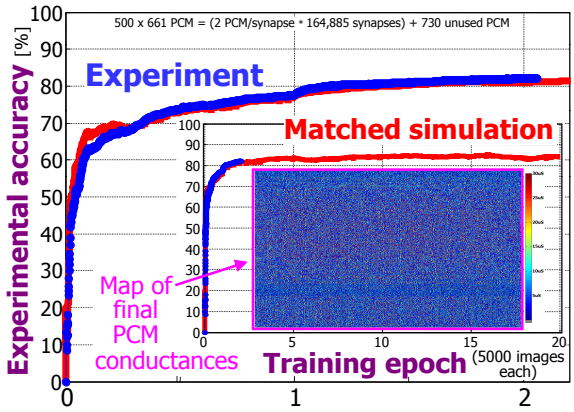

Fig. 3. Training and test accuracy for the same network of 164,885 hardware synapses shown in Fig. 2, with all weight operations taking place on a 500 x 661 array of mushroom-cell PCM devices. We also developed a simulator based on experimental data which matches the experimental results.

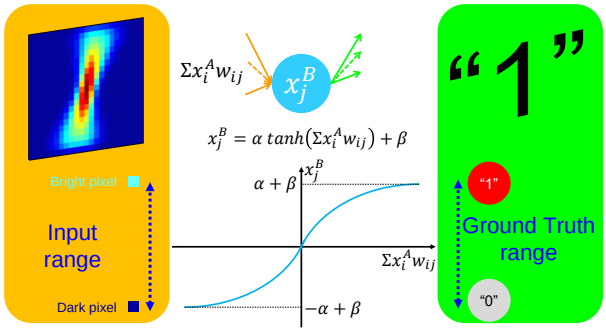

Fig. 4. The neuron activation is obtained by summing all the input contributions $x_i^A w_{ij}$ and by then applying a nonlinear squashing function (here, the hyperbolic tangent function, scaled by α and shifted by β). The input range corresponds to the mapping of input pixels with respect to the dynamic range of the neuron. The Ground Truth range is similarly varied.

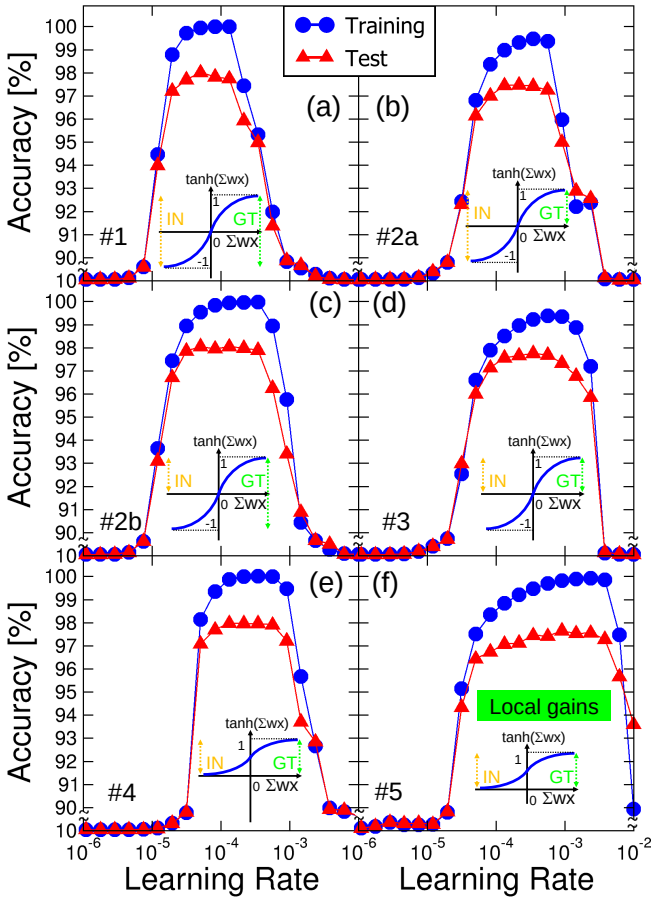

Fig. 5. Calculated training (blue) and test (red) accuracies of a pure-software implementation of our 3-layer neural network (not the "2–PCM" approach) as a function of the learning rate. Different combinations of activation functions and input/GT dynamic ranges have been analyzed: (a) $tanh$ function, input $\in$ [-1 , 1] and GT $\in$ {-1, 1}. (b) $tanh$ function, input $\in$ [-1 , 1] and GT $\in$ {0, 1}. (c) $tanh$ function, input $\in$ [0 , 1] and GT $\in$ {-1, 1}. (d) $tanh$ function, input $\in$ [0 , 1] and GT $\in$ {0, 1}. (e) scaled $tanh$ function, input $\in$ [0 , 1] and GT $\in$ {0, 1}. (f) scaled $tanh$ function, input $\in$ [0 , 1] and GT $\in$ {0, 1} with local gains. Training is done with 20 epochs of 60,000 images; all networks are tested with 10,000 images.

ranging from -1000 to +1000 and integer activations ranging from -255 to +255 (activations plotted in the paper are shown scaled to ± 1). Except where noted, all simulations were run for 20 epochs, which ensured network convergence, using 60,000 images for training and 10,000 images for test. The test accuracy was studied with respect to the variation of the learning rate η, namely the coefficient of every weight update $\Delta W = \eta x \delta$, where x is the neuron activation and δ is the backpropagated error.

A. Symmetric activation function

Fig. 5 shows training and test accuracies as a function of differences in the relative scaling of inputs, GT, and the $tanh$ function, illustrating the individual impact of each scale parameter. Fig. 5a shows the starting case (labelled #1), using the usual hyperbolic tangent $tanh(\Sigma x w)$, so that the activation function, the input range, and GT range are each symmetric and each spans the full interval [-1,1]. Note that the vertical scale accentuates the accuracy range above 90%, with accuracies between 90% and a poorly-trained network (10%) plotted in a highly compressed manner. Accuracy starts rising at a learning rate around 10^{-5}, exhibiting a bell-shaped curve with maximum value at 98% accuracy near 10^{-4}.

In Fig. 5b and (c), the effects of a positive-only GT, case #2a, or a positive-only input, case #2b, are analyzed separately, maintaining the symmetric activation function and same parameters as for case #1. In both situations, test accuracy starts rising around 10^{-5}: #2b shows a larger bell shaped curve, with a larger plateau, roughly showing a similar behavior with respect to #1. In Fig. 5d, case #3, input and GT are fully positive, ranging in the interval [0,1]. The shape of the resulting test accuracy remains comparable to the previous cases, with the peak value around 97.7%. Note that the positive range of GT implies that the output layer of the network is forced by backpropagation to have output between zero and one, thus failing to exploit the available dynamic range from -1 to 1. However, this limitation apparently has negligible effects, at least with the MNIST dataset. All training accuracies show a comparable behavior, generally similar to test accuracy.

B. Positive activation function

After analyzing all the possible combinations of input and GT dynamic ranges, we studied the effect of a fully positive activation function, as shown in Fig. 5e, case #4. We adopted a scaled hyperbolic tangent (α = 0.5, β = 0.5), compressed to the range [0,1], thus matching the corresponding range of input and GT. The resulting accuracies start rising at a slightly higher learning rate, while the accuracy is comparable to previous cases. We observe negligible differences arising from different cases, which should give us a high degree of freedom when designing the real network. Typically, positive activation functions are easier to implement in hardware than bipolar curves. Finally, case #5, Fig. 5f, shows the same situation as in case #4, but with the Local Gain (LG) algorithm – which will be described in the next section – activated. The #5 curve shows a broader plateau, which is highly desirable for reducing sensitivity to learning rate tuning, with no degradation of the maximum test accuracy, around 97.6%.

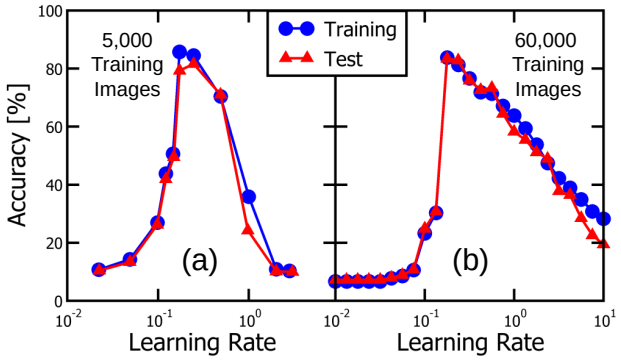

Fig. 6. (a) Training (blue) and test (red) accuracy as a function of the learning rate on 5,000 training images. The network configuration is the same as case #3, but using the device–imperfection–aware simulation first described in [2] ("2–PCM" approach). Compared to (a) the results of [2], using (b) 60,000 images for training only makes a slight difference at higher learning rates.

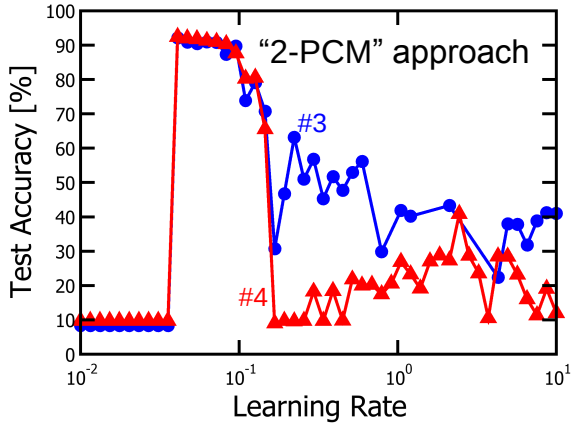

Fig. 7. Test accuracy of the same network in Fig. 2 with the "2–PCM" approach, as a function of learning rate. The network configurations correspond to cases #3 and #4 of Fig. 5d, e. Simulations were run for 20 epochs, trained with 60,000 MNIST images.

The simulations and experimental results in our first paper [2] (Fig. 3) correspond to case #3 in Fig. 5. Fig. 6a replots the results that were obtained. We use that same neural network simulator, capable of capturing the stochastic and highly variable behavior of real experimental PCM devices, and remodel the network of Fig. 2 with weights encoded in pairs of PCM devices ("2–PCM" approach). Simulations were run for 20 epochs, trained and tested with MNIST dataset, respectively with 5,000 images for training and with 10,000 images for the test. Test and training accuracies were evaluated with learning rates between 10^{-2} and 10^{1}. The accuracy curves sharply rise around 10^{-1}, reaching a peak value of 82%. Fig. 6a replicates our results previously published in [2], [3]. We then repeated the same simulation with the same set of parameters, but trained over the entire 60,000 examples of the MNIST dataset, producing similar behavior (Fig. 6b). Peak accuracy is almost the same around 83%, demonstrating that the limited accuracy shown in Fig. 6a cannot be improved by simply training with additional images.

We repeated the same simulation for 60,000 training images, but now optimizing the slope of the hyperbolic tangent activation function and the overall hyperparameters (the simulations in [2] were not intended to be optimized, but instead to show the variation around the particular operating point demonstrated experimentally). Test accuracy results for cases #3 and #4, with the two-PCM scheme representing the same imperfection modeling described in [2], are reported in Fig. 7. While accuracy rises up to 92%, the range of learning rates over which high accuracy is obtained is still quite narrow. This poses challenges in implementing optimal training of the network in hardware.

III. Local Gain Technique

The local gains concept of a synapse-centric local learning rate was first proposed in 1988 [20]. Fig. 8 shows a schematic representation of the scope and impact of local gains: in a network with no local gains, some weights (A), exhibit a steady increase or decrease as training progresses. Others (B), exhibit a dithering behavior, moving up or down as some images are trained and then later in the opposite direction as other images are trained, essentially maintaining a constant mean value. The local gains method rewards weights which head consistently in one direction (either increasing or decreasing, A') and reduces the oscillation of weights found to be dithering (alternating in the sign of the update, B'). With no local gains, the weight update rule is $\Delta W = \eta x \delta$. The original instance of the local gains algorithm modulated the learning rate for each synapse by multiplying ΔW with a local factor g_W [20]. In this original instance, two parameters are required per synapse: the factor g_W and a second variable to record the recent history of that synapse.

Motivated by the synapse-centric local gain algorithm [20], we have developed a *neuron-centric* and crossbar-compatible local gain algorithm, which we call LG. Here the LG parameters (the gain g and the recent-local-history) are stored at the neurons, updated according to the change of sign in x and δ neuron values. Now, weight update is $\Delta W = \eta x \delta g_x g_\delta$, where g_x (g_δ) depends on changes in sign between successive x (δ) values encountered during training. We translate this to g_x increasing when two consecutive x are strictly positive or decreasing otherwise:

$$g_x(i) = \begin{cases} g_x(i-1) + LGC, & sign(x(i)x(i-1)) > 0 \\ g_x(i-1)(1 - LGC), & otherwise \end{cases} \quad (1)$$

$g_x(i)$ corresponds to g_x during the training of image i, updating from the value $g_x(i-1)$ used for the previous training image. LGC is the Local Gain Coefficient, nominally equal to 0.01 in the simulations. Note that we increase g_x linearly and decrease it exponentially to preserve the positive sign of g_x, which varies between a minimum LG (nominally 0.1) and a maximum LG (10).

Similarly, g_δ depends on δ:

$$g_\delta(i) = \begin{cases} g_\delta(i-1) + LGC, & sign(\delta(i)\delta(i-1)) > 0 \\ g_\delta(i-1)(1 - LGC), & otherwise \end{cases} \quad (2)$$

with the same LGC, maximum and minimum LG as for g_x. This means that we calculate the weight update in a crossbar-compatible algorithm, with the upstream neuron firing pulses

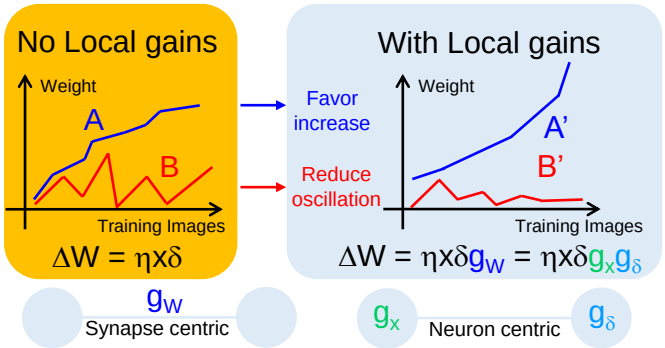

Fig. 8. Local Gain technique rewards the weights which head to a precise direction (either increasing or decreasing) and suppresses weights that are "dithering". The neuron centric approach locates the local gain coefficients into the neurons and not into the synapses.

according to the value of $\sqrt{\eta} g_x x$ and the downstream neuron firing pulses according to $\sqrt{\eta} g_\delta \delta$. The main contribution of the LG algorithm is to locally modulate the global learning rate η at each and every neuron. This neuron-centric approach greatly reduces the number of parameters which must be stored, from 164,885 g_W (one for every synapse) in the synapse-centric representation to less than 1500 (one g_x and one g_δ per neuron).

Both the original and our new neuron-centric LG method require local storage of the recent history. In our neuron-centric LG method, only the history of x is stored in the upstream neuron (during forward propagation), and the history of δ in the downstream neuron (during backpropagation). By using cases #4 and #5, we can avoid the exchange of information between these neurons: since x is always positive, the sign of ΔW can only depend on the sign of δ. This provides significant simplifications when designing the weight update circuits.

A. Initial implementation

We implemented the LG algorithm in the context of PCM devices with the same imperfections exhibited in [2], and show results in Fig. 9 for training with the first 5,000 images from the MNIST training set. As sketched in Fig. 9a, we applied g_x and g_δ in all the layers. Fig. 9b plots the training and test accuracies as a function of the learning rate. Results are not optimal, with a test accuracy just reaching 60%. To understand this behavior, we extracted the probability distribution function (PDF) of g_x, Fig. 9c, corresponding to the best result in Fig. 9b. From the initial distribution of g_x (all g_x = 1), after 20 training epochs we can observe a smooth distribution for g_x of the input layer neurons ($g_{x,IL}$). However, the g_x of all the hidden neuron layers ($g_{x,HL}$) saturates at the maximum value allowed (e.g., at 10.0). The reason is that, while the input pixels are positive or zero, hidden layer neuron activations can only be positive, forcing a continuous increase of g_x.

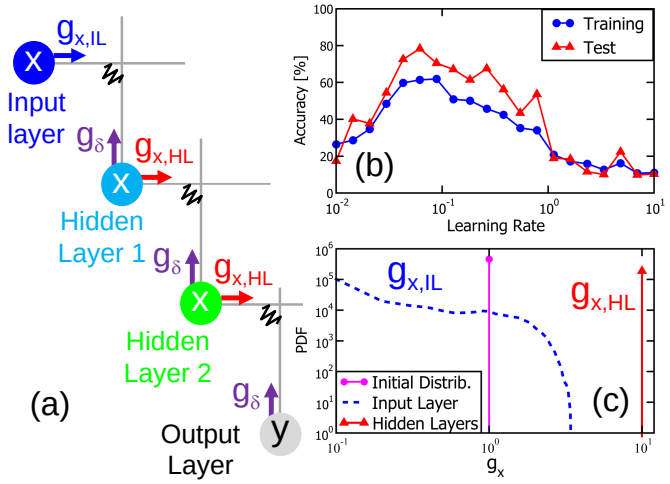

Fig. 9. (a) Schematic of initial LG scheme with g_x applied to input and hidden layers and g_δ applied to hidden and output layers. (b) Corresponding training and test accuracy after 20 epochs for a 3-layer perceptron with different learning rates. (c) Cumulative distribution of g_x before training and after 20 epochs.

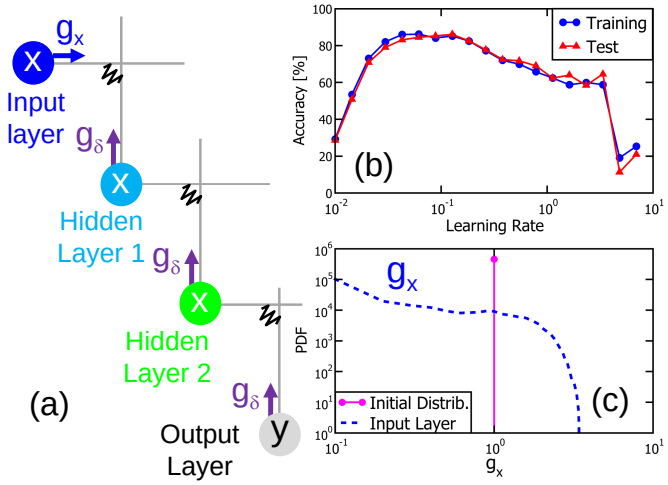

Fig. 10. (a) Schematic of intermediate LG scheme with g_x applied to input layer and g_δ applied to hidden and output layers. (b) Corresponding training and test accuracy after 20 epochs for a 3-layer perceptron with different learning rates. (c) Distribution of g_x before training and after 20 epochs.

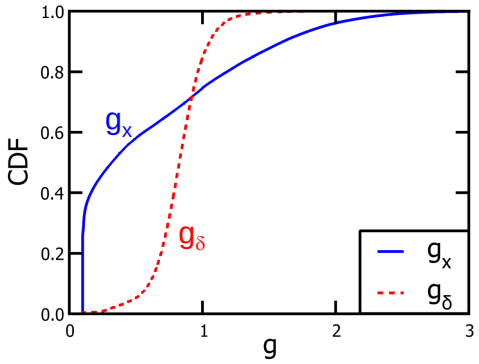

Fig. 11. Cumulative distributions of g_x and g_δ for the configuration of Fig. 10: almost 40% of input layer neurons having g_x stuck at minimum value 0.1. This results in a low neuron activity since the weights are frozen. On the other hand, g_δ varies smoothly with a median slightly below 1.

B. Intermediate implementation

Since g_x saturates for hidden layers, we adapted our method by using a constant g_x = 0.1 in those layers, obtaining the structure of Fig. 10a. We update g_x only in the input layer, resulting in the distribution of g_x in Fig. 10c. The accuracy results in Fig. 10b show increased robustness of training and test accuracies to variations of the learning rate. However, here the LG algorithm still fails to improve the best-case accuracy.

To better understand local gains behavior, we plot the cumulative distribution function (CDF) of g_x and g_δ, Fig. 11. We observe that almost 40% of input layer neurons have g_x stuck at or near the minimum value of 0.1. This results in a low neuron activity since the corresponding weights are frozen and are unable to update their conductances. On the other hand, g_δ varies smoothly in a range slightly below one.

C. Final implementation

The backpropagated δ's carry information about the classification error – in effect, representing the importance of any given neuron (and the synapses upstream of that neuron) in terms of correcting the classification of the particular image

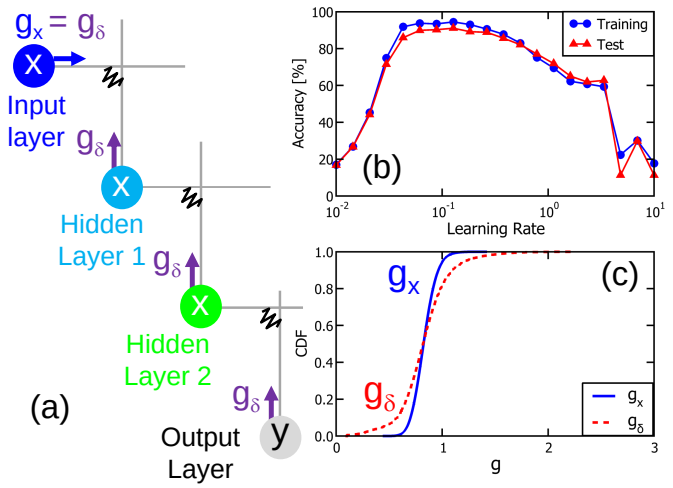

Fig. 12. (a) Schematic of final LG scheme with g_x derived from g_δ of the input layer and g_δ applied to hidden and output layers. (b) Corresponding training and test accuracy after 20 epochs for a 3-layer perceptron for different learning rates. (c) Cumulative distribution of g_x and g_δ before training and after 20 epochs. Now g_x exhibits a behavior similar to that of g_δ.

being trained. Thus if this quantity maintains the same sign at any given input neuron, it implies that this input pixel offers significance for classification. This motivated us to implement a similar distribution for g_x and g_δ, together with the need to avoid frozen weights as in Fig. 11. We pretended there was an additional synaptic layer above the input layer, and propagated the associated error (that would be needed for such a layer of synapses) to the input neuron layer. This information, as shown in Fig. 12a, was then used as the g_x of the input layer. Fig. 12b reports higher accuracies, while Fig. 12c shows the CDF of g_x and g_δ, revealing the absence of stuck weights and similar behaviors, since now both g_x and g_δ derive from the backpropagated error.

A final comparison of g_x for intermediate and final implementation is reported in the color maps in Fig. 13, representing the g_x values. Fig. 13a shows a strong correlation with the dataset, thus revealing that the network suppresses pixels outside the central region of the image, which seems to degrade accuracy. Instead, the final implementation of the LG algorithm exhibits no correlation with the input image, allowing the network to train from all input pixels.

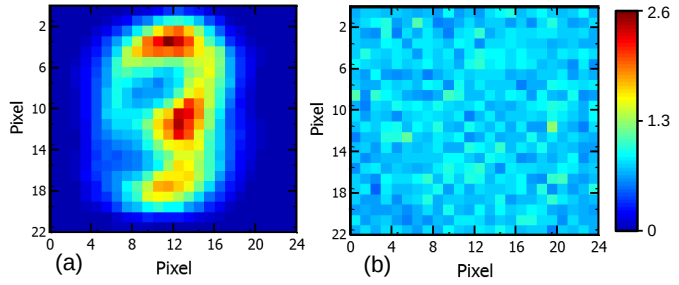

Fig. 13. Map of g_x in the input layer (a) for the initial and intermediate implementations and (b) for the final implementation of the LG algorithm.

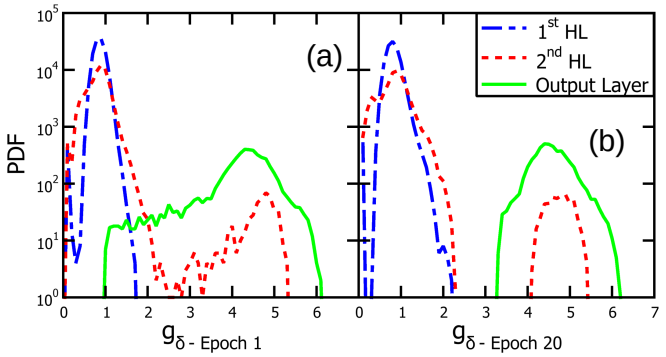

Fig. 14. Distribution of g_δ for the first and second hidden layers (HL) and for the output layer, collected from 500 randomly-selected images within (a) training epoch 1 and (b) epoch 20.

IV. Results

Fig. 14 reports the distributions of g_δ for this final version of the LG algorithm. Data are obtained by collecting g_δ from 500 images distributed throughout (a) the first training epoch and (b) the 20^{th} training epoch. The first hidden layer distribution shows negligible variation on the entire training. Since the initial condition is $g_\delta = 1$, the evolution happens primarily in the first epoch. On the contrary, the output layer shows a drift towards large g_δ between 3 and 6, because these g_δ values directly depend on the classification error (GT – output value). Since any neuron's error is 10% likely to be positive and 90% to be negative, two consequent training steps have a ∼80% probability of having the same δ sign. Thus, on average, g_δ increases. Interestingly, the second hidden layer (HL) shows two statistical families, which probably derive from principal signal paths (family at large value) and secondary paths (family around 1). Further information on what these two distinct distributions represent within the network would be an interesting area of future research work.

We also studied the impact of the choice of LGC, Max LG and Min LG, as reported in Fig. 15. Unless LGC is close to one (at which point, the LG devolves into chaotic behavior as shown in Eqs. 1 and 2 since g shows strong changes), there is no dependence. Similarly, the choice of maximum g_δ or g_x value has little impact. Increasing the minimum value too much degrades accuracy, since this prevents the suppression of dithering weights.

Fig. 5f shows the application of LG to the pure-software implementation of our three-layer neural network, while Fig. 16 adds the effect of the final neuron-centric LG algorithm – for the "2–PCM" approach representing real PCM devices – to the results without the LG algorithm plotted earlier in Fig. 7. Clearly, in both cases the range of learning rates offering high accuracy is larger and increases from 92% to more than 93% in the "2–PCM" approach.

We wanted to better understand the learning process during training by studying the network output. Fig. 17 plots the PDF of the ten output neurons during the forward propagation, taken from Fig. 16 at $\eta = 0.05$, corresponding to the peak. The results are shown for epochs 1 and 20. There are two families for each plot: the left one corresponds to neurons whose GT

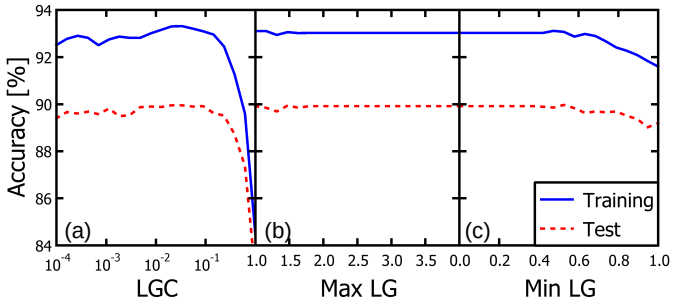

Fig. 15. Training and test accuracy after 5 epochs, 5000 training images, each point averaged over 20 simulations, for a 3-layer perceptron as a function of (a) Local Gain Coefficient (LGC), (b) maximum local gain and (c) minimum local gain.

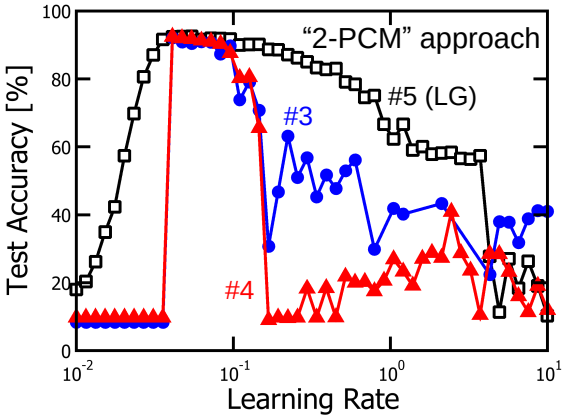

Fig. 16. Test accuracy of the network in Fig. 2 with the 2 PCM scheme, as a function of learning rate, comparing cases #3 and #4 and now #5. The LG algorithm improves accuracy slightly but provides a much broader window of useful learning rates.

output should be 0, the right one to those neurons whose output should be 1. The distance between the peaks of the two families reveals the ability of the system to classify images correctly. The curves in Fig. 17a show a large broadening as a consequence of the activation function ranging between -1 and 1. The output in Fig. 17b, case #4, is limited to the positive side, as is case #5 (Fig. 17c). Fig. 17c shows the effect of LG, which results in a more crisp distinction between the two distributions.

V. Safety Margin

Data reported in Fig. 17 provides information on the ability of the network to separate high and low neuron values, but does not give any relationship between outputs with GT = 1 (the excitation of the correct neuron) and the corresponding outputs with GT = 0 (all the other neurons). For this reason, we introduce the Safety Margin (SM) concept to provide a quantitative metric for the quality of network training. SM is defined as the difference between two particular neuron excitations within the output neuron layer – the excitation of the neuron which *should* be the right answer, and the highest of all the other excitations – as sketched in Fig. 18. A sign is attributed to this quantity: SM is positive when the highest value obtained at the output layer corresponds to the correct

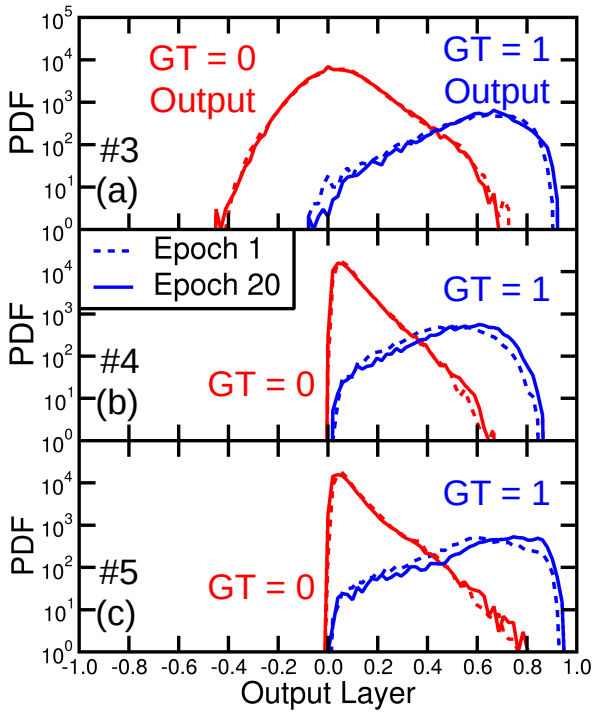

Fig. 17. Distributions of network output layer during forward propagation for first and last epochs, cases #3 (a), #4 (b) and #5 (c). The two families correspond to outputs with corresponding GT = 0 or GT = 1.

classification, and becomes negative when the classification is wrong. Higher learning quality, namely a strong ability to distinguish between the correct classification output and the other (incorrect) neuron outputs, leads to a larger SM. Thus SM is an analog metric that not only encapsulates accuracy, but also the margin by which that correct (or incorrect) classification was made. Note that the conventional definition of DNN loss involves the difference between the entire output vector and GT, while SM only reflects the difference with the two largest neurons. Thus output neuron distributions can have many different loss values yet share the same SM, and other distributions can exhibit many different safety margins yet share the same DNN loss.

Fig. 19 shows the CDFs of SM for cases (a) #3, (b) #4 and (c) #5, for simulations in Figs. 16 and 17. Safety Margins were

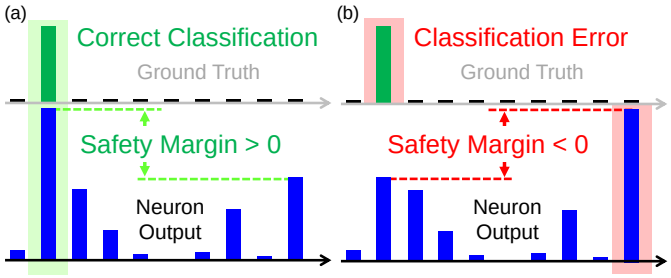

Fig. 18. Scheme of the Safety Margin (SM) concept. (a) When the network classifies the output correctly, the Safety Margin is the positive difference between the correct neuron and the next largest neuron. (b) When the classification is incorrect, the Safety Margin is a negative number indicating the gap by which the output neuron failed to be the highest neuron value.

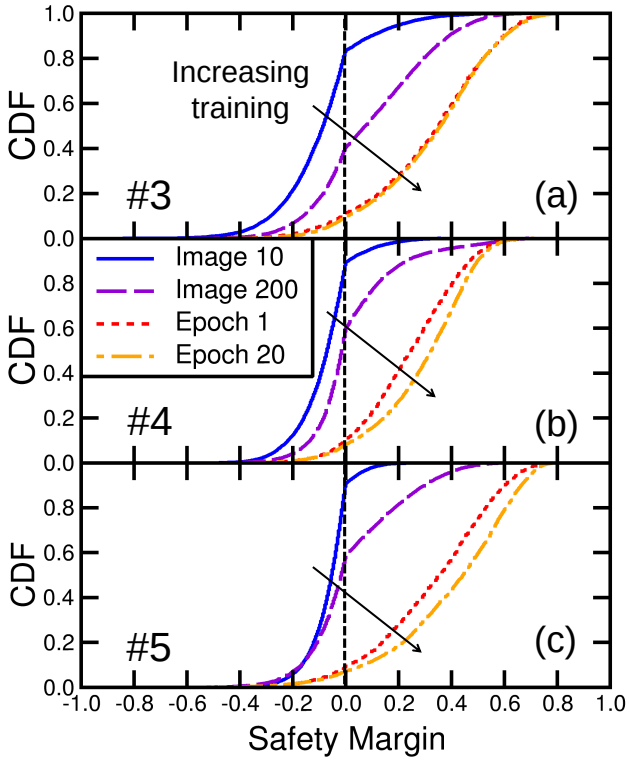

Fig. 19. Cumulative distributions (CDF) of Safety Margin (SM) for cases #3 (a), #4 (b) and #5 (c). The values were extracted by running the entire test set after the first 10 images of the first epoch, after the first 200 images of the first epoch, after the first epoch, and after epoch 20.

evaluated after the first 10 and 200 images of the first epoch, after the first epoch of 60,000 images, and after epoch 20. For increasing number of training images, the CDF shows a shift towards positive values, which means that a higher number of images are correctly classified. Fig. 19c shows a larger shift of the curves towards higher SM with respect to (a) and (b), revealing that LG helps achieve a higher learning quality.

VI. Energy Consumption

We also analyzed the impact of LG on energy consumption. Since the algorithm reduces the weight update coefficients for dithering weights, the number of partial–SET weight-update pulses is reduced. Fig. 20 shows the correlation plot between test accuracy and energy consumption. Global learning rate goes from low to high values. We show both energy consumption due to SET pulses (used for conductance increase and evaluated as 3 pJ for every pulse [2], [21]) and for RESET pulses (corresponding to 30 pJ [2]) for #4 (with no LG) and #5 (with LG). We observe that no advantages arise from the RESET energy since these RESET are periodically performed with the same period (after every 100 examples) in both cases. On the other side, LG leads to accuracy higher than 90% with a SET energy reduction of more than 10x with respect to the case without LG (#4). There is an increase of SET energy for LG at small learning rates, as the network increases local gains haphazardly in a doomed effort to train the network.

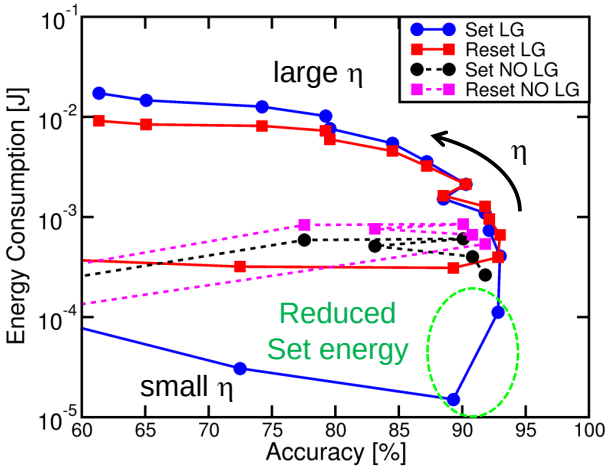

Fig. 20. Correlation between the energy consumption incurred by SET (e.g., weight-update) and occasional-RESET pulses and the associated test accuracy, for cases #4 and #5 (with LG). LG reduces set energy by more than 10x.

VII. Conclusion

In this paper, we introduced a neuron-centric version of the Local Gains (LG) method designed for crossbar-compatible implementation. We showed that this technique can increase network accuracy, extend the range of acceptable learning rates, and reduce energy consumption. We first analyzed the impact of different activation functions and the corresponding dynamic range of input and output neurons, showing that the use of non-negative neuron-activations allows the sign of the weight-update to depend only on the sign of the backpropagated error.

The concept of Safety Margin (SM) – the margin by which the correct output neuron exceeded (or failed to exceed) the strongest incorrect neuron – was introduced, providing a novel way to gauge the robustness of DNN classification performance. We used device-aware DNN simulations to demonstrate higher accuracy, reduced sensitivity to network hyperparameters, and an overall improved training process, as well as lower network activity and reduced energy consumption.

References

[1] Y. LeCun et al., Nature, 521, 436 (2015).
[2] G. W. Burr et al., IEDM Tech. Digest, 29.5 (2014).
[3] G. W. Burr et al., IEEE Trans. Elec. Dev, 62(11), pp. 3498 (2015).
[4] D. E. Rumelhart et al., Parallel Dist. Proc., pp. 45-76 (1986).
[5] Y. LeCun et al., Proc. of IEEE, 86(11) (1998).
[6] M. Suri et al., IEDM Tech. Digest, 79 (2011).
[7] S. Sidler et al., ESSDERC Proc., 440 (2016).
[8] G. W. Burr et al., IEDM Tech. Digest, 4.4 (2015).
[9] S. Agarwal et al.,, Front. Neurosci., 9, 484 (2016).
[10] A. Fumarola et al., ICRC Proc., 1 (2016).
[11] T. Gokmen and Y. Vlasov, Front. Neurosci., 10, 333 (2016).
[12] P. Narayanan et al., IBM J. Res. Dev., to appear (2017).
[13] P. Narayanan et al., Proc. ISCAS, (2017).
[14] S. Yu et al., IEDM Tech. Digest, 16.2 (2016).
[15] J. H. Lee et al.,, Front. Neurosci., 10, 508 (2016).
[16] S. B. Eryilmaz et al., IEDM Tech. Digest, 25.5 (2013).
[17] M. Prezioso et al., Nature, 521, 61 (2015).
[18] J.-W. Jang et al., IEEE Electron Dev. Lett., 36(5) (2015).
[19] Y. van de Burgt et al., Nature Mat., 16(4856), 414 (2017).
[20] R. A. Jacobs, Neural Networks, (4):295307, (1988).
[21] B. Jackson et al., ACM J. Emerg. Tech. Comp. Sys., 9(2), 12 (2013).