Analog Neural Networks with Deep-submicron Nonlinear Synapses

Ahmet Caner Yüzügüler, *Member, IEEE,* Firat Celik, *Member, IEEE,* Mario Drumond, *Member, IEEE,* Babak Falsafi, *Fellow, IEEE* Pascal Frossard, *Fellow, IEEE,*

Abstract—Deep neural network (DNN) inference tasks are computationally expensive. Digital DNN accelerators offer better density and energy efficiency than general-purpose processors but still not sufficient to be deployable on resource-constrained settings. Analog computing is a promising alternative, but previously proposed circuits greatly suffer from fabrication variations. We observe that relaxing the requirement of having linear synapses enables circuits with higher density and more resilience to transistor mismatch. We also note that the training process offers an opportunity to address the non-ideality and non-reliability of analog circuits. In this work, we introduce a novel synapse circuit design that is dense and insensitive to transistor mismatch, and a novel training algorithm that helps train neural networks with non-ideal and non-reliable analog circuits. Compared to state-of-the-art digital and analog accelerators, our circuit achieves 29x and 582x better computational density, respectively.

Index Terms—artificial neural networks, DNN accelerators, analog synapse, training

1 Introduction

Deep neural networks (DNN) have gained tremendous popularity thanks to their outstanding accuracy on difficult tasks in computer science such as image classification, speech recognition and natural language processing. Because of the required high computational density, DNNs, however, have limited applicability in resource-constrained environments. As a consequence, many DNN inference applications such as personal digital assistants (e.g., with speech recognition and natural-language understanding) are offloaded to the cloud which not only requires resource provisioning for communication but also incurs high latency. The latter precludes deployment in systems with hard real-time constraints, or in which dependence on connectivity is a major concern such as self-driving cars or biomedical robotics.

Modern DNN accelerators are primarily based on a linear synapse model and rely on multipliers and adders to calculate the sum of weighted inputs. As such, DNN accelerators' density is fundamentally limited by that of the multiply-and-add units. There are a myriad of techniques to improve DNN accelerator density in the literature such as reduced precision, zero-skipping and data reuse [3], [6]. Unfortunately, with silicon density scaling coming to a halt, these techniques fallen short of dramatic reductions in circuit size.

In contrast, analog neural nets offer a promising approach to increasing computational density by mapping the network directly to circuits. State-of-the-art designs, however, target mimicking the linear synapse characteristics of their digital counterparts and require either bulky multipliers [12] or circuits that are inherently susceptible to the fabrication variations such as current-mirrors [2] or differential amplifiers [7]. Other research advocates embedding the network in memory [10], [11]. While memristor crossbars provide dense computation and minimize weight traffic, they still require activations to be converted from analog to digital (and back) among the network layers, incurring prohibitive overhead. Additionally, memristors suffer from reliability issues, requiring error correction [4].

In this paper, we make the observation that the training stage of a DNN application offers a great opportunity to compensate for both a synapse's non-linearity and deep-submicron transistor variability in analog circuits. Adopting a non-linear synapse relaxes the requirement to include analog multipliers or current-mirrors, and enables synapse circuits that are smaller, faster and lower in power consumption. Based on these insights, we introduce a novel analog synapse circuit. Rather than relying on closed-form models, we propose a *black-box training* model that interpolates data from circuit simulation to calculate gradients. Moreover, because we simulate all deviation due to the fabrication process, our training scheme mitigates the negative impact of transistor variability. Our novel synapse design reinforced by the proposed black-box training scheme offers 29x more computational density and 12x better energy efficiency than state-of-the-art digital accelerators.

2 A novel analog DNN circuit

Artificial neural networks are organized in layers. Each layer consists of a number of neurons, each of which is connected to the previous layers through synapses. A neuron correlates its inputs with its weights to produce an output that is then fed to the next layer through a non-linear activation function. Our proposed circuit, shown in Figure 1a, works similarly. The weight generator circuit produces an analog voltage value, the synapse circuit correlates the output value of the previous layer with the analog weight value, and the soma circuits aggregate the currents from each synapse, process them through its non-linear transfer function, and produces voltage values for the next layer.

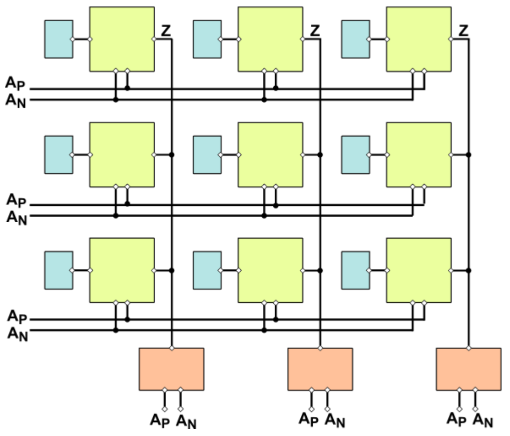
The weight generator circuit is a digital-to-analog converter, which takes a digital weight value, calculated during offline training, as an input. Its diagram is shown in

Figure 1b. The dimensions of each transistor in *WGEN* are adjusted to generate predefined analog values at the output for every digital input combination. This circuit generates $2^{n-1}+1$ analog values between 0 (when W0 is 0) and V_{cc} (when W1-W3 are 0) from an n-bit wide digital input. Because the output of this circuit is connected to the gate of a MOSFET in the synapses, it does not need to supply any current, which simplifies its requirements. We choose this circuit solely due to its simplicity, but it can be replaced with any digital-to-analog converter circuit with the required speed and precision.

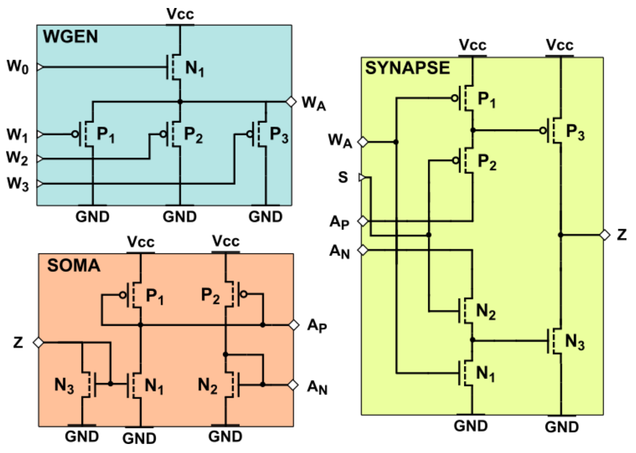
The synapse circuit takes a voltage value from the previous layer as input, processes it with the analog weight voltage value, and produces a current value as output. The input activation value is a complementary pair because the synapse circuit has symmetrical complementary *nmos* and *pmos* parts. These parts are exclusively activated by the digital input bit s, which represents the sign of the weight. When the pmos (nmos) part is activated, the synapse circuit sources (sinks) current into (from) the output node z, which corresponds to a positive (negative) weight in a neural network. The transistor P_1 (N_1) typically operates in the linear mode, and acts as a voltage-controlled resistor, which divides the voltage $V_{cc} - A_p$ (A_n) based on the voltage value W_A. The transistor P_3 (N_3) typically operates in the saturation mode and acts as a voltage-controlled current source. As a consequence, the output current of a synapse is independent of other synapses connected to the same node – except the channel length modulation effect – as long as the synapse remains in the saturation mode. As shown in Figure 1a, the output pins z of each synapse in a column are connected. We call this node the *the aggregating node*, because it adds up the output current of all synapses.Figure 2 shows the output current of a synapse with respect to the aggregated current of the neuron in the previous layer. Unlike prior work [2], [7], our synapse circuit design is neither a current-mirror nor a differential circuit, which are more sensitive to the fabrication variations.

The soma circuit takes the aggregated current from the synapses and converts it to a pair of complementary voltage values for the next layer. We use a current mirror as our soma circuit to decouple the synapses from the consecutive layers. Current mirrors are sensitive to fabrication variation; however, we reduce the effect of fabrication variation by aggressively increasing the soma circuit's dimensions. The silicon area and energy impact of the large soma circuits is small because the number of soma circuits in a typical neural network is much smaller than the number of synapses. The soma circuit also implements a non-linear activation function because the transistor N_3 cannot source current to z, leading to a rectifying effect that is similar to ReLU.

The layout of our circuit is drawn in Cadence Virtuoso using the TSMC 65nm technology. We measured that the design of the synapse, weight generator, and soma circuits occupy an area of $23.1\mu m^2$, $22.4\mu m^2$ and $76.4\mu m^2$, respectively. To train neural networks to be deployed on it, we developed the *black-box training* method, an offline training method for the non-linear analog synapse models designed in deep-submicron technologies.



(a) Connections for a 3x3 layer. Each column forms an artificial neuron. Inputs on the left side are the voltage values generated in the previous layer. The output of the soma blocks are propagated to the next layer.



(b) Top left: Weight generator, Bottom left: Soma, Right: Synapse

Fig. 1: Proposed analog DNN

3 Black-box Training

Since our synapse model is different from the conventional linear models, our circuit cannot run neural networks trained with standard backpropagation equations. Also, because we target 65nm technology, the ideal MOSFET models are no longer accurate, and therefore, cannot be used to model our circuit. As a result, it is not possible to find an accurate closed-form mathematical model of our synapse circuit. However, manufacturers and research groups such as [1] provide complex but accurate SPICE MOSFET models. Therefore, we developed a novel training method, in which we consider our synapse circuit as a black box. We build a single synapse circuit in a SPICE simulation tool and simulate its DC response. We store the simulated response of the synapse and soma circuits (denoted as F and H) in look-up tables and numerically calculate their derivatives. Then, we interpolate this data during training to approximate the outcome of the forward propagation and to calculate the gradients.

The response of our synapse circuit depends on the analog voltage weight value, W_A, the complementary pair of voltage activation values, A_P and A_N, and the voltage value

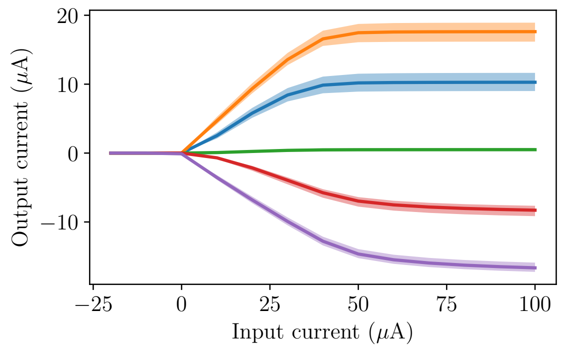


Fig. 2: Synapse characteristics of the proposed circuit. Each line corresponds to a different discrete weight value. Faded colors show the deviation of the synapse response due to the device mismatch, and solid lines show the average response.

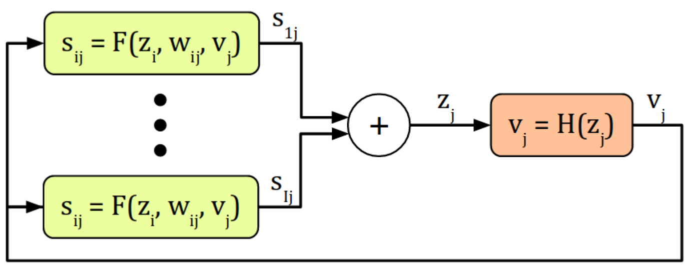


Fig. 3: Circuit decomposition of a neuron. Green and red blocks represent synapse and soma sub-circuits, respectively. The functions F and H are obtained with SPICE simulations. The diagram shows the circular dependency between the synapse and soma models due to the variables v_j^l and s_{ij}^l. The layer index l is dropped in the figure for simplicity.

at the aggregating node, V_z. For simplicity, we consider the aggregated current of the neuron from the previous layer, z^{l-1}, as the input of our synapse function. Equation 1 shows the formulation of our synapse function, F, where w_{ij}^l is the weight value of the synapse between the ith and jth neurons of the layers $l-1$ and l, and v_j^l is the voltage value at the aggregating node. At this stage, we ignore the fabrication mismatch and consider all synapse responses identical.

$$s_{ij}^l = F(z_i^{l-1}, w_{ij}^l, v_j^l) \tag{1}$$

The output currents of all synapses in a neuron are simply aggregated according to Kirchhoff's Current Law at the aggregating node:

$$z_j^l = \sum_i s_{ij}^l \tag{2}$$

The voltage value at the aggregating node, v_j^l is determined by the soma circuit (denoted as H) with respect to the aggregated output current, z_j^l:

$$v_j^l = H(z_j^l) \tag{3}$$

We decompose our full-network circuit into synapse and soma sub-circuits to approximate its DC operation point without simulating the entire circuit in SPICE. Figure 3 shows the circuit decomposition of a neuron and the circular dependency between its synapse and soma sub-circuits. Once we simulate a single synapse and a single soma circuits in SPICE, we use an iterative method, namely Newton's method, to find all required voltage and current values during a training iteration. In this method, we first initialize v_j^l to an arbitrary value (we take $V_{cc}/2$), and then iterate it with the update rule given in Equation (4) until the solution converges. See Appendix A for the derivation. Solving this system of equations in the forward pass produces the DC operation point values required by the backpropagation as well.

$$v_j^l \longleftarrow v_j^l - \frac{v_j^l - H(\sum_i F(z_i^{l-1}, w_{ij}^l, v_j^l))}{1 - \frac{\partial H(z_j^l)}{\partial z_j^l} \sum_i \frac{\partial F(z_i^{l-1}, w_{ij}^l, v_j^l)}{\partial v_j^l}} \tag{4}$$

We modified the backpropagation algorithms by taking the recursion of those two functions into account. Our modified equations result in the following formula to backpropagate the error term. See Appendix B for the derivation.

$$\delta_j^l = \sum_k \delta_k^{l+1} \frac{\frac{\partial F(z_j^l, w_{jk}^{l+1}, v_k^{l+1})}{\partial z_j^l}}{1 - \frac{\partial H(z_k^{l+1})}{\partial z_k^{l+1}} \sum_i \frac{\partial F(z_i^l, w_{ik}^{l+1}, v_k^{l+1})}{\partial v_k^{l+1}}} \tag{5}$$

Once the error terms for every neuron is calculated with Equation (5), we calculate the gradients as follows:

$$\frac{\partial C}{\partial w_{ij}^l} = \delta_j^l \frac{\partial F(z_i^{l-1}, w_{ij}^l, v_j^l)}{\partial w_{ij}^l} \tag{6}$$

Interpolating the simulation data is more costly than performing the multiplications in a conventional training. The synapse function F is evaluated by a trilinear interpolation, which requires 11 memory reads, 10 multiplication/division, and 20 addition/subtraction operations. The soma function H is evaluated by a linear interpolation, which requires 3 memory reads, 2 multiplication/division, and 4 addition/subtraction operations. In a conventional training, such operations can be achieved with only 2 memory reads and 1 multiplication. Moreover, solving the loop in Figure 3 iteratively with Newton's method increases the training cost. To prevent this, we set a maximum number of iterations for Newton's method. We find that five iterations are usually sufficient to obtain an accurate approximation. Nevertheless, decomposing the circuit and using Newton's method to solve it frees us from single-threaded SPICE simulations, and allows parallelization at the level of neurons and batches.

Due to variations in the manufacturing process, each transistor exhibits slightly different characteristics, a phenomenon called *device mismatch*. This mismatch causes a variation in the classification accuracy for each fabricated chip. The authors of [2] propose to measure the response of each neuron on a fabricated chip and train the neural network according to the measured responses. However, retraining neural networks for each fabricated chip would be costly. Therefore, we aim to achieve a fabrication-agnostic training method in this work. One of the factors to enable this in our work is that our synapse circuit is more resilient to the fabrication mismatch compared to the prior work. To further reduce the deviation in the classification accuracy, we also address this issue in our training algorithm.

The process variation is modeled as Gaussian noise added to the threshold voltage of each transistor [9]; with the standard deviation given in Equation (7). We run Monte Carlo simulations in SPICE to calculate the deviation in synapse response caused by the device mismatch. Then, we add this measured Gaussian error to the synapse output currents during the forward pass of the training, as shown in Equation (8). This serves two purposes. First, the trained neural network learns to classify the samples correctly with a noisy synapse response. Second, the added noise acts as a regularizer during training. We ignore the process variation while calculating the derivatives in the backward pass, as a small error does not change the slope of synapse response significantly, and it only affects the training time in a minor way.

$$\sigma_{\Delta VT} = \frac{A_{VT}}{\sqrt{WL}} \tag{7}$$

$$\tilde{s}^{l}_{ij} = F(z^{l-1}_{i}, w^{l}_{ij}, v^{l}_{j}) + \mathcal{N}(0, \sigma^{2}_{\Delta VT}) \tag{8}$$

4 Evaluation

4.1 Experimental Settings

We trained a neural network with the IRIS dataset [5] using black-box training. The IRIS dataset has four features and three classes with 150 samples in total. Our trained network has two hidden layers, each of which has ten neurons. We had to keep our neural network small due to long simulation times. We divided all the current values by a factor of 50×10^{-6} to prevent the numerical errors during the training, and we trained our network for 1000 iterations with a batch size of 20 and a learning rate of 5.0. We used the 65nm MOSFET models from [1] in our SPICE simulations and run our netlists with LTSpice with options $gmin = 1e-10$ and $abstol = 1e-10$. We take the device mismatch constant A_{VT} as $3mV\mu m$ for 65nm process in all our training and testing.

We use the *softmax cross-entropy* as the cost function as it is the most commonly used cost function for classification tasks. We take the aggregated output currents in the last layer as logits. For the weight update, we use *Momentum optimizer* with its default settings. Since our weights are discrete, we update the weights with stochastic rounding, which is given in Equation (9).

$$\begin{aligned} M^{l}_{ij} &\longleftarrow \lambda M^{l}_{ij} + \gamma \frac{\partial C}{\partial w^{l}_{ij}} \\ f^{l}_{ij} &\longleftarrow w^{l}_{ij} - M^{l}_{ij} \\ w^{l}_{ij} &\longleftarrow \begin{cases} \lfloor f^{l}_{ij} \rfloor, & \text{with probability } 1 - (f^{l}_{ij} - \lfloor f^{l}_{ij} \rfloor) \\ \lfloor f^{l}_{ij} \rfloor + 1, & \text{with probability } f^{l}_{ij} - \lfloor f^{l}_{ij} \rfloor \end{cases} \end{aligned} \tag{9}$$

4.2 Baselines

Efficient Inference Engine (EIE) [6] is a digital hardware accelerator to process the inference of fully-connected layers. In their implementation they use 4-bit weights and 16-bit activations. This is similar to our case, as the weights in our circuit are also 4-bit, whereas the activation values are analog signals. A large portion of the silicon area and power consumption in EIE is memory and sparsity decoders. Thus, we only consider its arithmetic and the activation units, which constitute only 3.58% of its overall area, and 26.16% of its overall power consumption.

Low-Power Analog Hardware (LPAH) proposed in [2] is a current-mirror based analog circuit for neural networks. They evaluate their results with two test cases with an average input current of the synapses of 15nA and 45nA. We compare our results with the latter because it has the same energy/op, yet it is faster. The paper does not report the silicon area, but it gives the dimensions of their transistors. Thus, we estimate their silicon area by scaling them with our dimensions and our layout design area in the 65nm process.

We choose two metrics for comparison: The energy per multiply-and-accumulate operation (MAC) and the computational density. Although our analog circuit does not do any multiply-and-accumulate (MAC) operations, we still report our findings in terms of MACs. Each MAC operation mentioned in our results corresponds to a synapse operation.

4.3 Results

We split the IRIS dataset into 90/30/30 samples for training, validation and testing and our trained neural network has a validation accuracy of 30/30, and test accuracy of 27/30. We choose this dataset because the previous work LPAH [2] also used the same dataset to evaluate their fabricated chip. In their experiments, they split the dataset into 120/30 samples for training and testing without validation, and achieve 30/30 on their tests.

In order to show the effect of fabrication variation on the classification accuracy, we scaled the synapse and soma circuit dimensions by a scaling factor and measured the classification accuracy 200 times in the presence of device mismatch noise. Figure 4 shows the test accuracy with the fabrication noise. As we increase the circuit dimensions, expectedly, the device mismatch decreases, leading to less circuit response variance and results in less variance in the classification accuracy. In other words, increasing the circuit dimensions enables achieving the same classification accuracy with less variance across fabricated chips. On the other hand, smaller circuits benefit from more computational density and better energy efficiency. Therefore, there is a trade-off in our analog circuit between the computational density/energy efficiency and the variation in the classification accuracy across the fabricated chips. All of our results in this section are based on simulations.

Since the results in LPAH are reported for a neural network with 25.1k synapses and 160 somas, we also projected our results accordingly and showed them in Table 1. The comparison of the energy/MAC metric shows that the analog circuits are significantly more energy efficient than the digital accelerator. The reason for this large gap

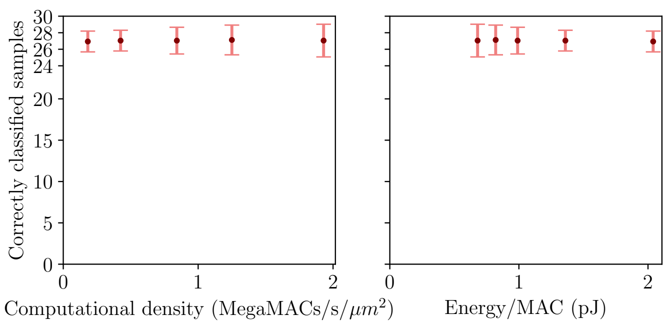


Fig. 4: Number of test samples classified correctly out of 30 samples versus the computational density (left) and energy per MAC operation (right). Each design point represents a circuit, whose area is scaled by 0.8, 0.9, 1.0, 1.2, and 1.5. Dots show the mean, and the bars show 2 times the standard deviation of 200 test runs, which corresponds to the 95% of the distribution.

is the three orders of magnitude higher transistor count in digital multipliers. The energy efficiency of LPAH is slightly better than ours. This is because we intentionally keep the synapse dimensions large to reduce the device mismatch effect, which enables fabrication-agnostic training. In terms of computational density, our proposed circuit is 29x better than EIE, and 582x better than LPAH. One of the reasons why our circuit is much denser than EIE is again the much lower transistor count compared to digital multipliers. Another reason is the absence of an activation unit; as our synapse and soma circuits obviate the need for an extra activation unit. The significant difference in computational density between our work and LPAH is due to its 500x slower response time.

	LPAH	EIE	This work
Delay (ns)	10400	1.25	21
Energy/MAC (pJ)	0.12	3.0	0.26
Computational Density (MACs/s/μm^2)	1.75×10^3	3.5×10^4	1.02×10^6

TABLE 1: Comparison with baseline accelerators. Delay values for LPAH and this work correspond to the input-to-output time, whereas the delay for EIE is one clock cycle.

5 Future Work and Discussion

One of the distinguishing features of our circuit is that the weight generation and the synapse are decoupled, so the speed and area of the synapse circuit are independent of weight bit-width. In prior work [2], in contrast, each additional weight bit requires an additional current-mirror, which quadratically increases the area and power cost. Therefore, our proposed circuit can scale much better with increasing weight precision. To that end, we are planning to extend the precision of our weight generator circuit to support DNN applications that require weight precision higher than 4 bits. In the meantime, we are also planning to explore different types of digital-to-analog converter types for our weight generator circuit, such as capacitor-based solutions as in [8].

Our circuit is applicable to any type of neural network, yet we believe that the ideal type would be recurrent neural networks (RNN). In a typical RNN, the vast majority of the output nodes are *hidden states*, which are fed back to the input. This structure enables us to keep most of the input/output signals in the analog domain and avoid the overhead caused by the data converters. Also, many of the DNN workloads on mobile devices are tasks like speech recognition and NLU, and they utilize variants of RNNs. Therefore, we aim to benchmark our design with an RNN workload next, and we expect to show a remarkable improvement in performance and energy-efficiency. Although we showed that our circuit and training algorithm give accurate classification results in the IRIS dataset, we aim to show its scalability in more challenging datasets with larger networks. To do so, we need an efficient parallel implementation of our training method. Approximating the overall circuit with Newton's method allows us to parallelize the training in the level of neurons and batches. However, we still need a low-level optimization to compensate for the extra cost of the interpolation and the iterative process.

6 Conclusion

In this work, we introduced a novel analog synapse circuit and a black-box training methodology. Our synapse circuit is more resilient to fabrication error than the prior work, and it is simple enough to have a low hardware footprint. Since the response of our synapse circuit is non-linear, it requires an unconventional training method. Thus, we introduce the black-box training method, which can calculate gradients even when the closed-form mathematical model of the synapse is not known. Our simulation results show that the our circuit combined with the black-box training can achieve a classification accuracy with minimal deviation across fabricated chips, without any need for calibration or re-training. Compared to the baseline analog neural network, our circuit achieves 582x better computational density while having a comparable energy efficiency. Compared to the baseline digital accelerator, our circuit offers 12x better energy efficiency and 29x better computational density.

Appendix A
Derivation of the update rule with Newton's method

Substituting Equation (1) and (2) into (3):

$$v_j^l = H(z_j^l) = H(\sum_i s_{ij}^l) = H(\sum_i F(z_i^{l-1}, w_{ij}^l, v_j^l)) \quad (10)$$

$$0 = v_j^l - H(\sum_i F(z_i^{l-1}, w_{ij}^l, v_j^l)) = N(v_j^l) \quad (11)$$

We want to find the root of $N(v_j^l)$. Newton's method suggests that v_j^l should be updated iteratively with the following update rule:

$$v_j^l \longleftarrow v_j^l - \frac{N(v_j^l)}{N'(v_j^l)} \quad (12)$$

Derivative of $N(v_j^l)$ with respect to v_j^l can be obtained by applying the chain rule:

$$\begin{aligned}\frac{\partial N(v_j^l)}{\partial v_j^l} &= 1 - \frac{\partial H(z_j^l)}{\partial z_j^l}\frac{\partial z_j^l}{\partial v_j^l} = 1 - \frac{\partial H(z_j^l)}{\partial z_j^l}\frac{\partial}{\partial v_j^l}\sum_i s_{ij}^l \\ &= 1 - \frac{\partial H(z_j^l)}{\partial z_j^l}\sum_i \frac{\partial s_{ij}^l}{\partial v_j^l} \\ &= 1 - \frac{\partial H(z_j^l)}{\partial z_j^l}\sum_i \frac{\partial F(z_i^{l-1}, w_{ij}^l, v_j^l)}{\partial v_j^l}\end{aligned} \tag{13}$$

Substituting Equation (1) and (13) into (12):

$$v_j^l \longleftarrow v_j^l - \frac{v_j^l - H(\sum_i F(z_i^{l-1}, w_{ij}^l, v_j^l))}{1 - \frac{\partial H(z_j^l)}{\partial z_j^l}\sum_i \frac{\partial F(z_i^{l-1}, w_{ij}^l, v_j^l)}{\partial v_j^l}} \tag{14}$$

Appendix B
Derivation of backpropagation equations

Defining the error term of the jth neuron of lth layer as usual:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{15}$$

To propagate the error into the previous layers, we modify Equation (15) as:

$$\delta_j^l = \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1}\frac{\partial z_k^{l+1}}{\partial z_j^l} \tag{16}$$

Taking the partial derivative of Equation (2) and applying the chain rule:

$$\begin{aligned}\frac{\partial z_k^{l+1}}{\partial z_j^l} &= \frac{\partial}{\partial z_j^l}\sum_i s_{ik}^{l+1} = \sum_i \frac{\partial s_{ik}^{l+1}}{\partial z_j^l} \\ &= \sum_i \frac{\partial F(z_i^l, w_{ik}^{l+1}, v_k^{l+1})}{\partial z_j^l} + \sum_i \frac{\partial F(z_i^l, w_{ik}^{l+1}, v_k^{l+1})}{\partial v_k^{l+1}}\frac{\partial v_k^{l+1}}{\partial z_j^l}\end{aligned} \tag{17}$$

The first summation term in Equation (17) results in a non-zero value only when $i = j$. Substitute 3 into the second summation term and apply the chain rule:

$$\begin{aligned}\frac{\partial z_k^{l+1}}{\partial z_j^l} =& \frac{\partial F(z_j^l, w_{jk}^{l+1}, v_k^{l+1})}{\partial z_j^l} \\ &+ \sum_i \frac{\partial F(z_i^l, w_{ik}^{l+1}, v_k^{l+1})}{\partial v_k^{l+1}}\frac{\partial H(z_k^{l+1})}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l}\end{aligned} \tag{18}$$

The terms $\frac{\partial H(z_k^{l+1})}{\partial z_k^{l+1}}$ and $\frac{\partial z_k^{l+1}}{\partial z_j^l}$ do not depend on i, so they can be taken out of the summation:

$$\begin{aligned}\frac{\partial z_k^{l+1}}{\partial z_j^l} =& \frac{\partial F(z_j^l, w_{jk}^{l+1}, v_k^{l+1})}{\partial z_j^l} \\ &+ \frac{\partial z_k^{l+1}}{\partial z_j^l}\frac{\partial H(z_k^{l+1})}{\partial z_k^{l+1}}\sum_i \frac{\partial F(z_i^l, w_{ik}^{l+1}, v_k^{l+1})}{\partial v_k^{l+1}}\end{aligned} \tag{19}$$

Rewriting the Equation (19) gives us $\frac{\partial z_k^{l+1}}{\partial z_j^l}$:

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = \frac{\frac{\partial F(z_j^l, w_{jk}^{l+1}, v_k^{l+1})}{\partial z_j^l}}{1 - \frac{\partial H(z_k^{l+1})}{\partial z_k^{l+1}}\sum_i \frac{\partial F(z_i^l, w_{ik}^{l+1}, v_k^{l+1})}{\partial v_k^{l+1}}} \tag{20}$$

Substituting Equation (20) into (16) gives the error back-propagation formula:

$$\begin{aligned}\delta_j^l &= \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \delta_k^{l+1}\frac{\frac{\partial F(z_j^l, w_{jk}^{l+1}, v_k^{l+1})}{\partial z_j^l}}{1 - \frac{\partial H(z_k^{l+1})}{\partial z_k^{l+1}}\sum_i \frac{\partial F(z_i^l, w_{ik}^{l+1}, v_k^{l+1})}{\partial v_k^{l+1}}}\end{aligned} \tag{21}$$

The gradient with respect to each weight can be calculated as:

$$\frac{\partial C}{\partial w_{ij}^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{ij}^l} = \delta_j^l\frac{\partial F(z_i^{l-1}, w_{ij}^l, v_j^l)}{\partial w_{ij}^l} \tag{22}$$

Note that the gradient for the bias terms is simply considered as a special case in which the input current z_i^{l-1} is a constant value.

References

[1] Predictive technology group. http://ptm.asu.edu/modelcard/2006/65nm_bulk.pm. Accessed: 2019-04-09.
[2] J. Binas, D. Neil, G. Indiveri, S.C. Liu, and M. Pfeiffer. Precise deep neural network computation on imprecise low-power analog hardware. *CoRR*, 2016.
[3] Y. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 2017.
[4] B. Feinberg, S. Wang, and E. Ipek. Making memristive neural network accelerators reliable. In *IEEE International Symposium on High Performance Computer Architecture, HPCA*, 2018.
[5] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 1936.
[6] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: Efficient inference engine on compressed deep neural network. *SIGARCH Comput. Archit. News*, 2016.
[7] J. B. Lont and W. Guggenbühl. Analog CMOS implementation of a multilayer perceptron with nonlinear synapses. *IEEE Trans. Neural Networks*, 1992.
[8] M. Milev and M. Hristov. Analog implementation of ANN with inherent quadratic nonlinearity of the synapses. *IEEE Transactions on Neural Networks*, 2003.
[9] M. J. M. Pelgrom, H. P. Tuinhout, and M. Vertregt. Transistor matching in analog CMOS applications. In *International Electron Devices Meeting 1998. Technical Digest*, 1998.
[10] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA*, 2016.
[11] L. Song, X. Qian, H. Li, and Y. Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In *IEEE International Symposium on High Performance Computer Architecture, HPCA*, 2017.
[12] J. M. Zurada. Analog implementation of neural networks. *IEEE Circuits and Devices Magazine*, 1992.