Joint Design of Training and Hardware Towards Efficient and Accuracy-Scalable Neural Network Inference

Xin He, *Member, IEEE,* Wenyan Lu, *Student member, IEEE,* Guihai Yan, *Member, IEEE,* and Xuan Zhang, *Member, IEEE*

***Abstract*—The intrinsic error tolerance of neural network (NN) presents opportunities for approximate computing techniques to improve the energy efficiency of NN inference. Conventional approximate computing focuses on exploiting the efficiency-accuracy trade-off in existing pre-trained networks, which can lead to suboptimal solutions. In this paper, we first present AxTrain, a hardware-oriented training framework to facilitate approximate computing for NN inference. Specifically, AxTrain leverages the synergy between two orthogonal methods—one actively searches for a network parameters distribution with high error tolerance, and the other passively learns resilient weights by numerically incorporating the noise distributions of the approximate hardware in the forward pass during the training phase. Then we incorporate AxTrain framework in an accuracy-scalable NN accelerator designed for high energy efficiency. Experimental results from various datasets with different approximation strategies demonstrate AxTrain's ability to obtain resilient neural network parameters for approximate computing and to improve system energy efficiency. And with AxTrain-guided NN models our proposed accuracy-scalable NN accelerator could achieve significantly higher energy efficiency with limited accuracy degradation under joint approximation techniques.**

***Index Terms*—Approximate computing, Neural network accelerator, Hardware-oriented training, Sensitivity analysis, Energy efficient architecture, Near threshold voltage, approximate multiplier.**

I. Introduction

An Artificial Neural Network (ANN) is a biologically inspired machine learning model that has been practically demonstrated to deliver superior performance in many recognition, mining, and synthesis (RMS) applications [1]. The success of ANN can be attributed to innovations across the computing system stack: To achieve higher accuracy, deeper and more complex networks are created along with more advanced training algorithm. To speed up network training and deployment, powerful specialized parallel computing engines (e.g., GPUs) are designed to accelerate computationally intensive mathematical operations. Despite the improved performance, energy efficiency remains a limiting factor when deploying advanced ANNs in IoT devices with stringent power budgets. For example, a typical wearable health monitoring device has a maximum power envelop around 180mW [2].

A growing body of research has been proposed to tackle energy efficiency from diverse perspectives. Algorithmically, the focus is to simplify neural network (NN) by either proposing concise network models (e.g. ResNet for ImageNet [3], and binary neural networks [4]); or pruning and compressing existing models [5]. From the hardware perspective, efficiency-driven optimizations have been conducted at the architecture, circuit, and device levels. Customized NN accelerators aim at higher energy efficiency[6]; approximate circuits trade accuracy for energy efficiency; and emerging technologies (e.g. RRAM crossbar) provide low-power NN computing substrates [7] [8]. In this paper, we investigate an auxiliary approach with a focus on network training that can be generally applied to facilitate diverse approximate computing techniques, and studies the efficacy of applying the training approach to achieve accuracy-scalable computing. The approach is orthogonally compatible with techniques to improve energy efficiency from other domains .

Existing approximate computing techniques are confined to exploiting pre-trained NNs, which can result in suboptimal solutions. Without knowledge of the underlying hardware, NN algorithms optimize only for accuracy under the assumption of ideal hardware implementation, yet they do not consider error tolerance due to hardware nonidealities. Therefore, small noises from approximate hardware may lead to severe network accuracy degradation. Compromises often have to be made to maintain the accuracy target, leading to conservative approximation and failure to exploit all the opportunities for efficiency improvement.

The key question is how to train a robust neural network that not only achieves high accuracy given ideal hardware assumptions, but is also resilient to noise and errors, so that more aggressive approximation could be applied without severely compromising accuracy. As Fig.1 illustrates, a conventional training algorithm is dedicated to searching for a "global" minimum which has the smallest loss across the weight space, ignoring higher loss in the vicinity of the minimum. Thus perturbations by approximate computing could easily result in significant loss, as indicated by "Local minimum 1". Instead of minimizing loss at a single minimum point, our proposed approximate computing oriented training seeks a "near optimal" minimum where a "flat" and "good enough"

X. He and X. Zhang are with the Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO, 63130 USA e-mail: ({xin.he, xuan.zhang}@wustl.edu).

W. Lu and G. Yan are with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences. W. Lu is also with the University of Chinese Academy of Sciences, Beijing, China e-mail: ({luwenyan, yan})@ict.ac.cn

Manuscript received Dec 15, 2017.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2018.2845396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

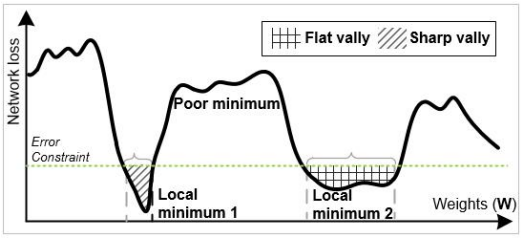

Fig. 1. Different types of minimums in NN weight space.

loss surface is preferred and the globally smallest error is not mandatory, as "Local minimum 2" depicts. Thanks to the flat error surface, the NN now exhibits a higher degree of tolerance for approximate computing induced noise.

In this paper, we first propose AxTrain, a hardware-oriented NN training framework for approximate computing. AxTrain explores two different paths towards high resilience: 1) an active method (AxTrain-act) that explicitly biases the training process to a noise insensitive minimum, and 2) a passive method (AxTrain-pas) that exposes the model of low-level hardware imperfection to the high-level training algorithm for noise tolerance. AxTrain then leverages the synergy between active and passive methods to facilitate approximate computing.

In the AxTrain-act method, the innovation is to *guide the training algorithm to improve both network loss and noise resilience directly*. During training, noise sensitivity is also back propagated along with network loss to the network parameters, and those parameters get updated in order to minimize loss and noise sensitivity. This solution can be seen as an artificial regularization term to bias the training algorithm towards a high resilience (flat) and accurate (near optimal) minimum, similar to the L2 norm regularization for the overfitting problem.

For the AxTrain-pas method, the error tolerance property of the NN is leveraged to reduce side effect from approximate computing. Rather than training with ideal hardware models, numerical functional models of the approximate hardware are incorporated along the forward pass in the training step, so that the training algorithm can learn the noise distribution of the approximate hardware on its own and *descend to a minimum which is robust to approximate computing*. Because with knowledge of the approximate hardware being employed in the computing system, the training process experiences different train sets with slightly modified statistical distributions in each epoch, and arrives at a robust model that yields high accuracy with approximate computing.

Since the proposed AxTrain framework can be used to find a robust NN which facilitates approximate computing, this advantage motivates the design of an accuracy-scalable NN accelerator which can support various accuracy modes. Under a relaxed accuracy constraint, the NN accelerator with AxTrain-guided model can be expected to achieve higher energy efficiency from approximate computing. We employ variable-latency approximate multipliers and lower-than-nominal voltage (including near-threshold voltage cases) supplied SRAM storage into an existing accelerator architecture to implement an accuracy-scalable NN accelerator. For low voltage SRAM storage, we trade the increased noise probability for reduced power consumption by leveraging lower than nominal supply voltage, while for the approximate multipliers the multiplication accuracy can be reduced for higher energy efficiency by raising their operating frequency. In this way, the proposed accuracy-scalable accelerator can be considered as performing "dynamic (SRAM) voltage and (multiplier) frequency" (DVFS) scaling to adapt to an appropriate accuracy mode and maximize energy efficiency according to the given accuracy requirement.

II. Related work and Background

A. Related Work

Approximate computing is a promising technique for efficiency optimization [9][10][11][12]. Due to the intrinsic noise tolerant capability of the NN, diverse techniques have been explored in prior work that apply approximate computing approaches to improve NN energy efficiency:

As an example that uses circuit-level techniques to handle memory bit upsets from error-prone but low power SRAM storage in NN accelerators, Minerva employs Razor sampling circuits for fault detection and equips the weight fetch stage with bit masking and word masking for flipped weights to mitigate bit-flip errors caused by Near threshold voltage (NTV) based weight storage [13].

Several prior work follows another direction by fully leveraging NN's self-healing property and demonstrates the benefit of combining NNs with approximate computing: Olivier shows NN accelerators can tolerate transistor-level faults by iteratively retraining a NN and choose the NN which meets certain accuracy requirement under those faulty transistors [14]; Zidong *et al.* perform an exhaustive design space exploration of an approximate accelerator in which all multipliers are accuracy-configurable and can be configured individually. Although it is not cost-efficient to design and deploy an approximate accelerator which support arbitrary configuration, Zidong's work nonetheless sheds light on exploiting NN's error tolerant characteristics for approximate computing [15]. Recent research proposes more explicit training techniques to exploit NN's intrinsic error tolerance and flexibility to improve efficiency. For example, both AxNN and ApproxAnn take neuron criticality into consideration and perform periodical retraining to heal the noise from approximate computing[16][17]. AxNN proposes the characterization of neuron criticality first, then replace those non-critical neurons with their approximate counterparts. To ensure targeted accuracy, iterative retraining with the functionality of approximate multipliers is carried out for error recovery. Inspired by AxNN, ApproxAnn proposes a more reliable way to quantify neuron criticality and adopts iterative heuristics to determine optimal combinations of approximate configurations (precision of the multipliers, choice of skipping memory load, bitwidth of datapath) for approximate computing so that energy efficiency can be maximized.

Although both AxNN and ApproxAnn strive to take the advantage of NN's pliable training process to improve energy

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2018.2845396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

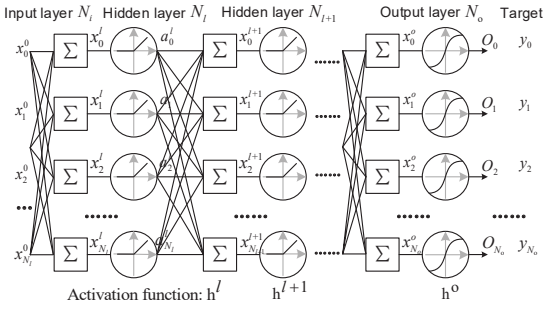

Fig. 2. A typical neural network topology.

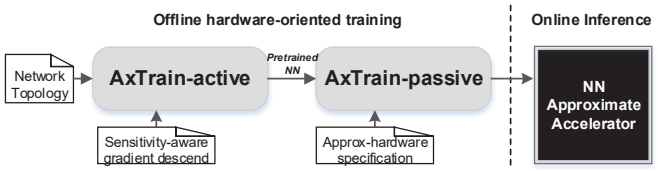

Fig. 3. Proposed AxTrain framework: 1)First AxTrain-active approach generates a pretrained NN model, 2) then AxTrain-passive approach works on the pretrained NN to directly learn hardware characteristics. 3)Finally the obtained NN model can be used for offline approximate inference in NN accelerator.

efficiency, certain limitations in their techniques persist: 1) They require highly configurable hardware where modes of multipliers can be individually configured to different modes; 2) For large-scale networks with time-multiplexed multipliers due to area and power constraints, periodic runtime multiplier reconfiguration will inevitably degrade accelerator performance. 3) Approximation is performed on a pre-trained network with hardware-agnostic training, which does not optimize for error tolerance, so the target accuracies in their designs are met with relatively conservative approximations. All these limitations motivate our AxTrain framework.

B. Neural Network Preliminary

Neural network is a network made of artificial neurons. Inspired by real brain, neural network is expected to roughly approximate certain functionality with simplified layer structures. With enough neurons, neural network could act as a universal approximator to represent any functions.

At the architecture level, ANN can be seen as a parallel computing engine which consists of a large number of basic hardware elements, such as multipliers, accumulators, and nonlinear transformation units. A typical neural network as shown in Fig.2 consists of an input layer, multiple hidden layers, and an output layer. During the **forward pass**, the input layer retrieves inputs, a_{in}, of a task sample and directly passes them to the next layer. To generate activations, a_i, for each neuron i in a hidden layer, the hidden layer first performs multiplication and accumulation, $\sum_{j=1}^{n} a_j \times w_{ij}$, using activations a_j from the previous layer and network parameters W (including weights and biases), then feeds the intermediate results to a nonlinear transformation h, such as Sigmoid ($\frac{1}{1+exp^{-x}}$) for the output layer, ReLu ($max(x, 0)$) for hidden layers. This process is repeated layer by layer until the output layer is reached, and the final activations (outputs) from the output layer are generated for regression and classification.

NN training aims at exploring network parameters which minimize the error between network outputs and targets. To reduce the error, **backpropagation (BP)** is used to propagate output error backward from the output layer to the previous layers consecutively and to quantify error contributions from network parameters by taking derivatives of the output error with respect to these parameters. Then the parameters are updated in a backward pass using stochastic gradient descent to the derivatives to reduce output error. The mathematical equations for training can be briefly summarized as:

The derivative of the output error with respect to ith neuron in layer l is

$$\frac{\partial E}{\partial x_i^l} = (\sum_{j=1}^{N_{l+1}} \frac{\partial E}{\partial x_j^{l+1}} \times w_{ji}^{l+1}) \times h'(x_i^l) \quad (1)$$

The weights' gradient and updating method are derived as

$$\frac{\partial E}{\partial w_{ji}^l} = \frac{\partial E}{\partial x_j^l} \times a_i^{l-1} \quad (2)$$

$$w_{ji}^l = w_{ji}^l - \eta \Delta w_{ji}^l \quad (3)$$

where η is the learning rate.

III. AxTrain Framework for Accuracy-Scalable Computing

In this section, we present the proposed hardware-oriented AxTrain framework that searches for a "near optimal" and resilient minimum to facilitate approximate computing and achieve a better trade-off between inference accuracy and energy efficiency. Specifically, AxTrain exploits two different methods: AxTrain-act explicitly regularizes the NN to descend to parameter distributions that are insensitive to noise; and AxTrain-pas intentionally models approximate computing-induced noise in the forward-pass of the training and internalizes the noise distribution in its learned weights. Finally AxTrain leverages the synergy between the active and passive methods by first training with AxTrain-act to reduce overall sensitivity and then with AxTrain-pas to learn hardware-specific noise. The workflow of AxTrain is illustrated in Fig.3.

A. AxTrain-active Method

1) Define network sensitivity-oriented regularization: AxTrain-act introduces robustness as an additional regularization term to an NN's cost function to drive NN training. In machine learning, regularization is a process that can introduces prior knowledge to the training process to express preference in the solution. For example, an L2 regularization term reduces the magnitudes of NN weights and limits NN capacity to prevent over-fitting. Similarly, AxTrain-act defines robustness and incorporates it into the cost function for training, as illustrated below.

$$E_{tot} = E + \lambda \cdot S(w) \quad (4)$$

where E is the original NN output error. $S(w)$ represents the network sensitivity, and a lower sensitivity suggests higher

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2018.2845396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

resilience and more robustness to noise. We use λ as a preference factor for sensitivity. Based on Eq.4, AxTrain-act minimizes not only network error but also noise sensitivity. To reduce the output error, E, training algorithm employs backpropagation to evaluate the gradient $\frac{\partial E}{\partial w_{ij}^l}$ and update network weights as described in Section 2.

Since the magnitude of $S(w)$ should reflect how output deviations are affected by noisy weights, we define a NN's sensitivity as

$$S(w) = \sum_k (\sum_{\forall l,ij} |w_{ij}^l| |\frac{\partial O_k}{\partial w_{ij}^l}|) \tag{5}$$

This definition satisfies four important aspects: 1) We employ absolute values to guarantee that the training process works on worst-case sensitivity reduction, and noises from those sensitive weights cannot cancel out each other to arrive at a smaller $S(w)$. 2) $\frac{\partial O_k}{\partial w_{ij}^l}$ is the derivative of an output k to a weight ij in layer l, which is used to measure the outputs' response with respect to weights perturbation. 3) $|w|$ is also incorporated. The induced noise from approximate hardware (low voltage storage in this case) is related to the maximum values of stored data, i.e., larger weights requires more integer bits to represent so that bit-flips in the high order bits can result in larger weight deviations. 4) To minimize heuristic intervention in the optimization process, we capture the total sensitivity by summing across all weights, instead of ranking or partitioning individual weight contribution [16]. Based on this definition, we can infer that a network with small $S(w)$ would behave similarly with and without noise, and hence exhibit better resilience against approximation. The challenge now is *how to reduce network sensitivity $S(w)$ in training*.

2) Estimate gradients:
Inspired by BP and SGD, we propose to calculate the gradients that measure how the sensitivity changes with respect to the weights and then update the weights accordingly to reduce sensitivity, similar to the conventional BP weight updates for minimizing loss.

Taking a specific weight ij as an example, to minimize the sensitivity we should make the update along its negative gradient $\frac{\partial S(w)}{\partial w_{ij}}$, which can be derived as

$$\begin{aligned}\frac{\partial S(w)}{\partial w_{ij}} &= \frac{\partial \sum_k (\sum_{ab} |w_{ab}| |\frac{\partial O_k}{\partial w_{ab}}|)}{\partial w_{ij}} \\ &= \sum_k (sign(w_{ij}) |\frac{\partial O_k}{\partial w_{ij}}| \\ &+ \sum_{ab} (|w_{ab}| sign(\frac{\partial O_k}{\partial w_{ab}}) \cdot (\frac{\partial^2 O_k}{\partial w_{ab} \partial w_{ij}}))\end{aligned} \tag{6}$$

The first term, $sign(w_{ij})|\frac{\partial O_k}{\partial w_{ij}}|$, is evaluated in conventional BP.

Evaluation of the second term for all w is complicated because of the second order derivative (Hessian matrix). Directly calculating the Hessian is a time-consuming process, hence we adopt Pearlmutter's algorithm [18] to speed up the computation, since Pearlmutter's algorithm can compute NN's "Hessian (H) vector (V) product" in $O(n)$ time simply by **another round of forward-backward propagation.** In our case, $|w| \cdot sign(\frac{\partial O}{\partial w})$ could be denoted as vector V, while $\frac{\partial^2 O}{\partial w_{ab} \partial w_{ij}}$ as the Hessian matrix H for all parameters. Pearlmutter's algorithm proposes the R operator which facilitates calculation as

$$R_V\{f(w)\} = \left.\frac{\partial f(w + rV)}{\partial r}\right|_{r=0} \tag{7}$$

And based on the definition of R, the basic rules for R can be derived:

$$\begin{aligned}&R_V\{f(w) + h(w)\} = R_V\{f(w)\} + R_V\{h(w)\} \\ &R_V\{f(w) \cdot h(w)\} = f(w) R_V\{h(w)\} + h(w) R_V\{f(w)\} \\ &R_V\{f(h(w))\} = f'(h(w)) + R_V\{h(w)\}\end{aligned} \tag{8}$$

Hence the second term $V \times H$ of Eq.6 is transformed into $R_V\{\frac{\partial O_k}{\partial w_{ij}^l}\}$. After applying R operator to Eq.2, we have:

$$\begin{aligned}R_V\{\frac{\partial O_k}{\partial w_{ij}^{l+1}}\} &= R_V\{\frac{\partial O_k}{\partial x_i^{l+1}} \cdot a_j^l\} \\ &= R_V\{\frac{\partial O_k}{\partial x_i^{l+1}}\} a_j^l + R_V\{a_j^l\} \frac{\partial O_k}{\partial x_i^{l+1}}\end{aligned} \tag{9}$$

To compute this equation, we can obtain $\boldsymbol{R_V\{\frac{\partial O_k}{\partial x_i^{l+1}}\}}$ and $\boldsymbol{R_V\{a_j^l\}}$ with a second round propagation as follows:

a) For the forward pass, the R operator is applied to get $\boldsymbol{R_V\{a_j\}}$:

$$\begin{aligned}R_V\{x_j^{l+1}\} &= R_V\{\sum_{i=0}^{n} a_i^l \cdot w_{ji}^{l+1}\} \\ &= \sum_i V_{ji}^{l+1} \cdot a_i^l + \sum_i w_{ji}^{l+1} \cdot R_V\{a_i^l\}\end{aligned} \tag{10}$$

$$\begin{aligned}R_V\{a_j^{l+1}\} &= R_V\{h^{l+1}(x_j^{l+1})\} \\ &= h^{(l+1)\prime}(x_j^{l+1}) \cdot R_V\{x_j^{l+1}\}\end{aligned} \tag{11}$$

For the input layer, $R_V\{a_j^{(0)}\} = 0$. After forward propagation, we can get $R_V\{a_j^l\}$;

b)For the backward pass in the hidden layers to get $\boldsymbol{R_V\{\frac{\partial O_k}{\partial x_i}\}}$:

$$\begin{aligned}R_V\{\frac{\partial O}{\partial x_i^l}\} &= R_V\{(\sum_{j=1}^{N_{l+1}} \frac{\partial O}{\partial x_j^{l+1}} \cdot w_{ji}^{l+1}) \cdot h'(x_i^l)\} \\ &= h''(x_i^l) R_V\{x_i^l\} (\sum_{j=1}^{N_{l+1}} \frac{\partial O}{\partial x_j^{l+1}} \cdot w_{ji}^{l+1}) \\ &+ h'(x_i^l) (\sum_{j=1}^{N_{l+1}} \frac{\partial O}{\partial x_j^{l+1}} \cdot V_{ji}^{l+1}) \\ &+ h'(x_i^l) (\sum_{j=1}^{N_{l+1}} R_V\{\frac{\partial O}{\partial x_j^{l+1}}\} \cdot w_{ji}^{l+1})\end{aligned} \tag{12}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2018.2845396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

c)For the backward pass in the output layer:

$$\begin{aligned} R_V\{\frac{\partial O}{\partial x_i^o}\} &= R_V\{\frac{\partial O}{\partial O_i} \cdot h^{(o)\prime}(x_i^o)\} \\ &= \frac{\partial O}{\partial O_i} h^{(o)\prime\prime}(x_i^o) R_V\{x_i^o\} + \frac{\partial^2 O}{\partial O_i^2} R_V\{O_i\} h^{(o)\prime}(x_i^o) \\ &= h^{(o)\prime\prime}(x_i^o) R_V\{x_i^o\} \end{aligned} \tag{13}$$

From the equations above, we can derive $\boldsymbol{R_V\{a_j\}}$ and $\boldsymbol{R_V\{\frac{\partial O_k}{\partial x_i}\}}$, then these two terms can be substituted into Eq.8, and then into Eq.6 so that the influence of network weights on sensitivity, $\frac{\partial S(w)}{\partial w_{ij}}$, can be computed.

3) Update the preference factor adaptively:

As defined in Eq.4, AxTrain-act aims at reducing both the network error and sensitivity. And to control the balance between network error and sensitivity, AxTrain-act uses a preference factor λ to manipulate the magnitude of the sensitivity-related parameter updating. Choosing an appropriate λ is non-trivial, because a large λ will hinder final network accuracy while a small one prevents the full reduction of sensitivity. To choose λ wisely, we leverage an adaptive update policy based on [19] to fully reduce the sensitivity and maintain required accuracy. Instead of using a fixed λ, this policy updates λ based on training history and on a per epoch basis, as detailed in Algorithm 1.Before training, a few hyperparameters are defined as: a initial value of λ is set empirically as one tenth of the learning rate, the adjusting number $\Delta\lambda$ as one tenth or one fifth of λ, a targeted error threshold E_{thres}, a decay factor σ (e.g., 0.5) for penalizing λ and a weight factor γ (e.g., 0.9) for summation of errors E_a in the training history. At the end of each training epoch, a $\Delta\lambda$ will be added to λ to put more focus on reducing sensitivity (Line 5), if the error E_i in the current epoch is smaller than E_{i-1} in the previous epoch or E_i is smaller than the target accuracy threshold E_{thres} (Line 4). Otherwise, λ gets reduced for accuracy (Line 6). Specifically, if E_i is still smaller than the weighted summation E_a in the previous epochs (Line 7), λ is reduced by merely a $\Delta\lambda$ (Line 8), or λ is decayed by the decay factor σ (Line 12). When λ is reduced, λ should be kept to be a non-negative value (Line 10). Finally, E_a is updated to keep track of the output error history (Line 13). The balance between network sensitivity-related optimization and network output error is maintained dynamically by the proposed preference factor updating algorithm. When current error E_i is smaller than previous error E_{i-1}, the algorithm tends to increase the proportion of sensitivity-related loss in the cost function because a conservative value cannot fully exploit the error tolerance. It is worth noting that the magnitude of the updating step $\Delta\lambda$ (in line 5) is set to a small value (one-tenth or one-twentieth of the initial) so that the output error will not fluctuate much. In case of sudden error drop, the λ is decayed by σ (0.5 in our experiments) for error recovery. With the dynamical adaptation capability, AxTrain-act ensures both the overall output quality and the noise tolerance.

Algorithm 1 Preference factor λ updating policy

Parameters: Initial λ, $\Delta\lambda$, Output error threshold E_{thres}, Decay factor σ for λ, Weight factor γ for error accumulation

1: Accumulated error $E_a = 0$
2: **while** E_{tot} does not convergence **do**
3: Output Error E_i = AxTrain-act() in ith epoch
4: **if** $E_i < E_{thres}$ or $E_i < E_{i-1}$ **then**
5: $\lambda = \lambda + \Delta\lambda$ //Biasing training to put more focus on sensitivity reduction
6: **else**
7: **if** $E_i <= E_a$ **then**
8: $\lambda = \lambda - \Delta\lambda$ //Biasing training to put more focus on output error reduction
9: **if** $\lambda < 0$ **then**
10: $\lambda = 0$ // Ensuring a non-negative λ
11: **else**
12: $\lambda = \sigma \cdot \lambda$ //Decaying λ to regain network quality
13: $E_a = E_i + (1 - \gamma) \cdot E_a$

B. AxTrain-passive Method

Different from AxTrain-act, which explicitly optimizes for robustness, AxTrain-pas exposes the nonideality of approximate hardware to the training algorithm by numerically mimicking its inexact operations in the forward propagation. Because of the incorporated hardware knowledge, AxTrain-pas can learn the noise distribution from approximate hardware and implicitly exploit the noise insensitive minimum. AxTrain-pas is a hardware-oriented approach which can be generally applied to most approximate techniques in NN accelerators: approximate arithmetic operations [15] for neuron calculation and fuzzy memorization [11] for parameter storage.

In neuron calculation, the most computational intensive operations consist of *weight activation multiplications* and later additions. Multipliers usually consume higher power and contribute more delays to the critical path than the adders used for accumulations, while the precision of multiplications is relatively less critical than that of additions for NN output accuracy [20]. All these considerations make multipliers better candidates for approximation, as shown in Fig.4, where an approximate multiplier is used in a neuron processing element. Every time a neuron forward calculation, $\sum_{j=1}^{n} a_j \cdot w_{ij}$, is performed, the original accurate multiplications are replaced by their approximate counterparts.

Power consumption for parameter storage also plays a significant role in NN accelerators, since NNs often consist of thousands of weight parameters. Fuzzy storage is thus leveraged to trade decreased weight precision for power reduction. Fig.4 also shows fuzzy storage for local and global weights. To model the effect of approximate computing in the training algorithm, AxTrain-pas models the noise induced to network weights by fuzzy memorization whenever the weights are retrieved in the forward propagation. Taking less-than-nominal voltage supplied fuzzy storage (detailed later) as an example, low supply voltage causes random bit flips, since it renders SRAM cells less reliable. During training, AxTrain-pas models

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2018.2845396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

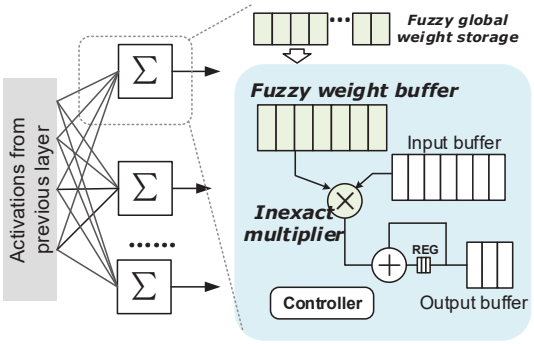

Fig. 4. Approximate computing in neural network accelerators.

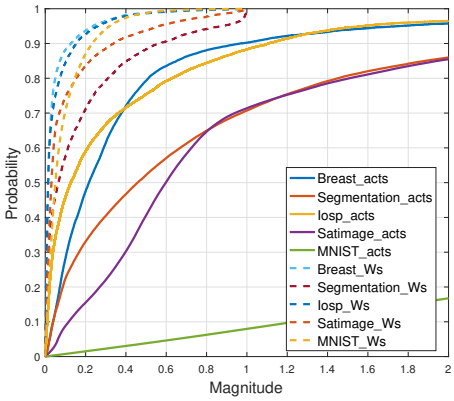

Fig. 5. Cumulative distribution function (CDF) of the magnitudes of NN's weights and activations

the induced flips as stochastic noise [21] and injects the noise by randomly flipping the bits in network weights at a certain probability (based on voltage level and technology). Note that AxTrain-pas applies approximation statically throughout the network. This policy reduces the complexity of hardware implementation, such as the support for runtime multiplier reconfiguration and memory mode switching.

When applying approximate computing in NN, we should first minimize noise from the approximate hardware itself. Taking low voltage supplied storage as an example, the upper bound of noise in network weights is determined by the binary format used to represent network weights, e.g., the noise magnitude for a sign-bit flip in a fixed-point number corresponds to the maximum value that the fixed point format can represent. Hence unnecessary high order bits that do not affect accuracy should be eliminated to confine the effect of the noise. Most network weights can be optimized to reside in a range of $10^{-3} \sim 2^{-1}$, so the integer bits are not necessary to represent the weights. Fig.5 illustrates the cumulative distribution function (CDF) of the magnitudes of NN's weights and activations in five applications. The dotted lines shows the statistical results for NN's weights. On average, above 90% percent of weights are smaller than 0.5.

We also notice the network's activations are almost two orders of magnitude larger than the weights statistically, which suggests activations and weights should be represented in different fixed-point formats. Hence, dynamic fixed point

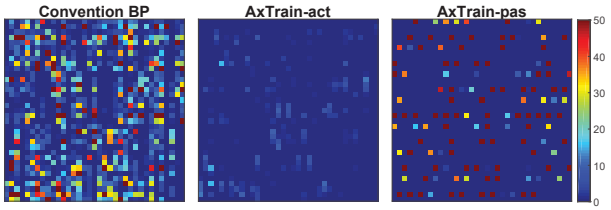

Fig. 6. Comparison of sensitivity maps between conventional BP, AxTrain-act, and AxTrain-pas.

representation is used in NN accelerators to maintain network functionality and confine noise [22][23].

Calculate gradients by straight-through estimator. After augmenting the forward propagation pass with numerical models of approximate computing, a natural question arises: How are the gradients backpropagated through approximate hardware? Given the nonlinear or stochastic nature of approximate hardware, it is hard to analytically compute the precise derivatives across the entire input range for approximate operations. Inspired by Hinton's lecture (12b) [24] and Bengio's work [25], we adopt the "straight-through estimator" technique in AxTrain-act as below:

$$grad\_in = grad\_out \cdot 1 \mid_{|grad\_out|<1} \quad (14)$$

This BP method directly passes gradients from the outputs of an approximate operator to its inputs, while preventing noise-induced large gradients from disturbing the training algorithm's convergence. Base on our experimental evaluation, this BP method is effective for AxTrain-pas training.

We examine the efficacy of AxTrain-act and AxTrain-pas by comparing their weight sensitivity (flatness) with conventional BP. Fig.6 shows the relative sensitivities of weights from NN's last (most critical) layer for the *MNIST digit recognition datasets*, where the deeper blue range indicate less sensitive weights. AxTrain-pas is trained with approximate multipliers in the most aggressive mode. Fig. 6 demonstrates the AxTrain-act significantly reduces the sensitivities across all the network weights, while AxTrain-pas implicitly learns noise distribution and selectively reduces the sensitivity for those weights which suffer larger noise from the approximate multiplier.

IV. Accuracy-Scalable NN Architecture

So far we have discussed AxTrain as a training framework towards generating robust NN models to mitigate accuracy degradation due to the noises from approximate computing. The superior robustness from AxTrain motivates us to explore the accuracy-scalable capability for NN hardware acceleration, i.e., the NN accelerator has the capability to adapt to different accuracy modes of approximate computing. With traditional back-propagation (BP) trained network model, the accelerator rarely operates at approximate modes since noises with small magnitude can even significantly degrade the inference accuracy. By leveraging the robust NN models from AxTrain, the accuracy-scalable accelerator could achieve higher efficiency improvement by operating at an less accurate but efficient mode more often under a slightly relaxed accuracy constraint (e.g., 2% allowed degradation).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2018.2845396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

Hence, to further exploit the advantage of AxTrain, we propose an accuracy-scalable NN accelerator architecture in which both the SRAM weight storage and the multipliers can flexibly switch between the accurate and approximate modes. Specifically, we build the accuracy-scalable NN accelerator based on an existing flexible data-driven NN accelerator named "FlexFlow" [26], as shown in Fig.7. Originally, FlexFlow employs a weight buffer and a neuron buffer for storage, a group of processing engines (PE) for computation, and an instruction decoder for controlling. To perform neuron calculation, each PE consists of a multiplier, an adder, a neuron local memory, a weight local memory, and a controller.

To augment the accelerator with accuracy-scalable capability, SRAM voltage scaling technique and approximate multiplication technique are adopted [11][21]. As shown in Fig.7, global and local SRAM weight storages reside in an individual voltage island and their supply voltage can be dynamically scaled to different levels (accuracy/power modes) [27]. And a variable-latency approximate multiplier is employed in each PE, and the accuracy of those multipliers is controlled by scaling its operating frequency. The proposed fixed point variable-latency multiplier for approximate computing is shown in Fig.8. The entire multiplication has three accuracy modes. 1) In order to perform an inaccurate but fast multiplication, we truncate the four high-order bits (significant bits) in the operands, A, B and sent them to a four-bit multiplier, then the intermediate multiplication result is left-shifted to generate $Out2$ for use. 2) To compensate for accuracy, four high-order bits of an operand A are multiplied with six low order bit of another operand B, then this multiplications results are shifted and added to the former results $Out2$ from four bits multiplier to produce $Out1$. 3) To get the accurate output, the residual multiplications are performed and added up with those former results, then accurate results $Out0$ is attained. Clearly, with higher accuracy demand, the latency of the multiplier will be larger, so its frequency will be smaller. From the simulation results, the critical path latencies of the variable-latency multiplier in three working modes (M2, M1 and M0) are 0.34ns, 0.41ns and 0.60ns respectfully with increasing accuracy. The accurate results of multiplications can be guaranteed by running at low frequency while the approximate results can be obtained by running at higher frequencies. To sum it up, we refer these supported approximations as "Dynamic (SRAM) Voltage (Multiplier) Frequency Scaling (DVFS)" for the NN accelerator. Note that the bitwidth of accelerator's datapath is set to ten while dynamic fixed point format is used to correlate data representation to the magnitude of network parameters. Specifically, for NN's inputs and activations, three bits are used for the integer part while seven bits for the fractional part. For NN's weights, all ten bits are used for the fractional bits since most weights have very small magnitudes. And when the multiplier is set to the conservative approximation mode, M1, during inference, operand A and B should be clarified. In this accelerator, the activation works as operand A while the weight as operand B for high inference accuracy since the magnitudes of activations are larger and the activations are reused repeatedly for different weights in a layer.

The overall workflow of the accuracy-scalable NN accelerator consists of two main phases. Before deployment, AxTrain-guided training finds robust models for approximate computing at different approximation strategies, i.e., AxTrain-act approach first strives to reduce network sensitivity to noise, then AxTrain-pas approach builds NN models for different approximate computing configurations. Meanwhile the corresponding statistical results on accuracy could also be reported. During deployment, based on collected accuracy information, the configurations for approximate hardwares in NN accelerator is determined according to the given target accuracy. For the multipliers, the operating frequency and datapath selection signal are set. For SRAM weight storage, its supply voltage is specified and the corresponding voltage regulator can be configured.

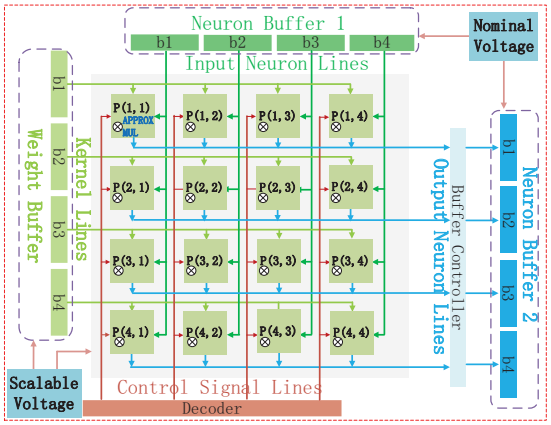

Fig. 7. The proposed accuracy-scalable NN accelerator architecture.

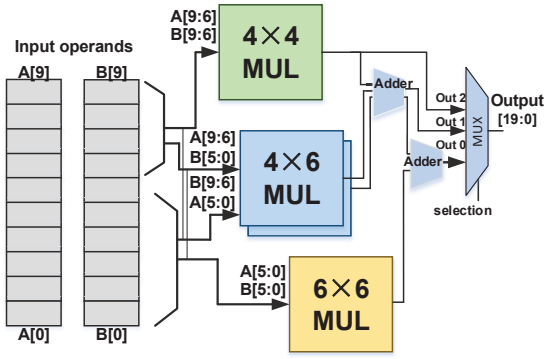

Fig. 8. The 10-bit variable-latency approximate multiplier.

V. Experimental Methodology

Case studies of approximate hardwares for approximate inference. 1) **Two kinds of approximate multipliers.** We use two different kinds of approximate multipliers in the experiments to demonstrate the generality of AxTrain framework to different kinds of designs. First, without loss of generality, to assess the implications of approximate multiplications in NN accelerator, we adopt an existing approximate multiplier for weight-activation multiplication [28]. This design explores the tradeoff between precision and computing efficiency which is similar to changing the effective width k for computation, i.e., with a smaller k configuration, the approximate multi-

plier gains higher energy efficiency at a cost of increased noise. Second, to build and evaluate the accuracy-scalable NN accelerator, we leverage a variable-latency approximate multiplier design as the building blocks for the accelerator. The structure of the variable-latency multiplier is detailed in Section IV. The multiplier has three accuracy modes, i.e., accurate mode M0, conservative mode M1, aggressive mode M2. 2) **Lower-than-nominal voltage storage.** For fuzzy storage, we leverage lower-than-nominal supply voltage for SRAM weight storage. Conventionally, SRAM works as a reliable storage at nominal voltage (e.g., 1.1V). To improve energy efficiency, the SRAM supply voltage can be reduced to the low voltage regime at the risk of bit flipping [21]. In this case, the supply voltage can be treated as a knob to tune approximate computing and determine the noise probability. In the experiment, we use three voltage levels: one slightly lowered voltage level and two NTV voltage levels. We select two representative knobs (flip rate@voltage: 10%@400mV, 1%@660mV, 0.1%@850mV [21]) for each applications, as Table I depicts.

Power and performance evaluation flow. To evaluate the power and performance improvement from approximate multiplier, we first implement approximate accelerator with imprecise multipliers using Verilog and then synthesize the design using the Synopsys Design Compiler with TSMC 65nm library. The power results are gathered using Synopsys PrimeTime; For near threshold voltage based storage, We evaluate the low voltage (including NTV) supplied storage by CACTI-P[29].

Training tool and Dataset. To evaluate the accuracy of NN, we implement the training algorithm and inference simulator using the PyTorch deep learning framework. The datasets we used are detailed in Table I. *Breast cancer, Image segmentation, Ionosphere,* and *Satimage* are obtained from the UCI Machine Learning Repository [30], and *MNIST* is a well known dataset for digit classification [31]. For each dataset, 80% of samples are used for training, while the remaining 20% are used for accuracy testing. In the off-line training, the networks are first trained with AxTrain-act until both the network error and sensitivity cost converge, then a few more epochs (e.g., ten) are trained with AxTrain-pas.

VI. Experimental results

In the experiment, we first examine the effectiveness of proposed AxTrain algorithm with approximate multiplier and low voltage storage separately. We conduct experiments for five representative applications with four approximate multiplier configurations (K1, K2, K3, K4) and two low voltage levels, which include an aggressive (Agg) lower voltage and a conservative (Con) higher voltage, as as Table I illustrates. Then we evaluate the accuracy-scalable NN architecture with AxTrain-guided models first by using the variable latency multipliers which have three accuracy modes (M0,M1,M2), then by incorporating both the variable latency multipliers and low voltage powered SRAM weight storage.

To examine the AxTrain, first we compare the output error of NN under different approximate multiplier configurations with networks trained by conventional BP, the proposed two methods, AxTrain-act and AxTrain-pas separately, and the combining scheme AxTrain. The results are shown in Fig.9. In the figure, the errors under different approximate configurations are normalized to original network results with accurate multipliers for each application.Fig.9 shows that the network outputs suffer larger error with more aggressive approximation configurations. Compared with conventional BP scheme, AxTrain-act exhibits higher noise tolerance by reducing the error by 40.55%, 23.60% and 24.41% on average for K1, K2 and K3, respectively, while AxTrain-pas reduces the error to 59.93.%, 45.38%, 26.03%. When combining the active and passive method together, AxTrain further reduces the errors to 71.95%, 51.90%, 35.26%, respectively. We notice that in a few rare cases (like K4 in *MNIST*), when using multipliers with conservative approximation, AxTrain-act performs slightly better than AxTrain. Due to the intrinsic error tolerance of the NN, the accuracy degradation caused by conservative approximate multipliers is quite small, thus the improvement headroom is limited.

For low voltage supplied SRAM weight storage, we show the results from fifty runs, since the induced bit-flipping is a probabilistic event. Fig.10 demonstrates the average accuracies and the deviations. The output accuracies for both aggressive and conservative low voltage levels increase by 30.03% and 8.57% on average, compared with conventional BP. This figure also indicates AxTrain reduces the side effects of bit flip in aggressive mode and restores the output quality to a higher level, equals to conventional BP attains in conservative mode. Note that accuracy is used as the comparison metric instead of network error, because error magnitude for conventional BP in the *MNIST-Agg* case is too large to be properly shown.

For a thorough evaluation of AxTrain, we also compare the results with two recent approaches, ApproxANN [17] and AxNN [16]. For ApproxAnn, its optimization can be used compatibly with AxTrain-act, thanks to the orthogonality of our training-based approach in supplementing efficient techniques from other domains. Originally, ApproxAnn accepts a *pre-trained BP network* as inputs, and ranks the network neurons based on their criticality. To incorporate approximate computing, ApproxAnn then conducts the multiplication operations for the less critical neurons on the approximate multipliers. To mitigate the newly-introduced output error, ApproxAnn retrains the network. Then neurons ranking is performed again and more less-critical ones are identified if there are still accuracy headroom for approximate computing. As the retraining step progresses, more and more multiplications can be carried out by approximate multipliers while maintaining the targeted accuracy (2% maximum allowed degradation in this case). Unlike AxTrain, which uses one approximate configuration throughout the inference and proposes to reduce the corresponding accuracy loss, ApproxAnn tries to employ as many approximate multiplications as possible to replace the expensive accurate multipliers without exceeding the accuracy requirement. So to make a fair comparison, we compare the number of approximate multipliers ApproxAnn could use with pre-trained networks using conventional BP and AxTrain-act, and we show the results for aggressive approximation (K1, K2)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2018.2845396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

TABLE I
DATASETS AND PARAMETERS.

Dataset	Description	#In,#Hiddens,#Out	Agg/Con volt	#Samples	Accuracy
Breast cancer	Diagnose cancer	30,64,64,2	400mV,660mV	569	99.12%
Image segmentation	Classify outdoor image	19,64,64,7	660mV,850mV	2100	95.71%
Ionosphere	Identify radar target	34,50,50,2	660mV,850mV	350	87.14%
Satimage	Classify satellite image	36,64,64,7	660mV,850mV	6434	86.71%
MNIST	Recognize hand written digit	784,128,128,10	660mV,850mV	60000	97.95%

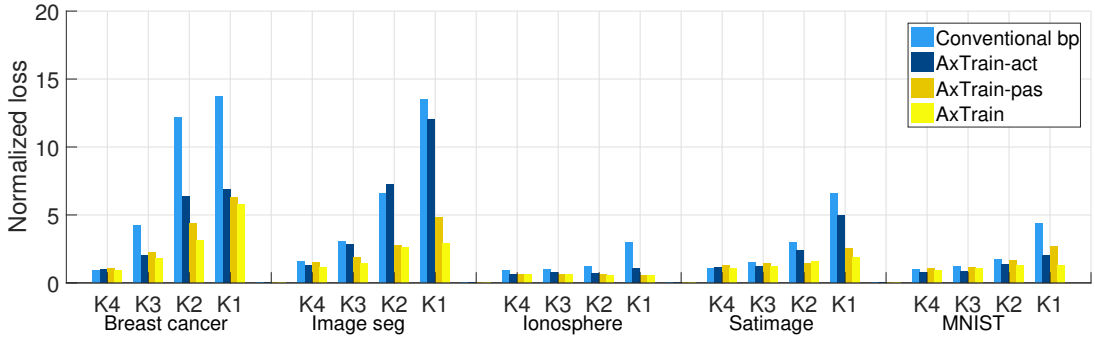

Fig. 9. Output loss under different approximate multipliers.

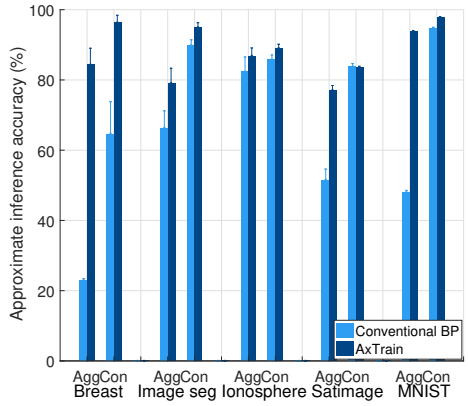

Fig. 10. Accuracy comparison between BP and AxTrain under different low voltage levels.

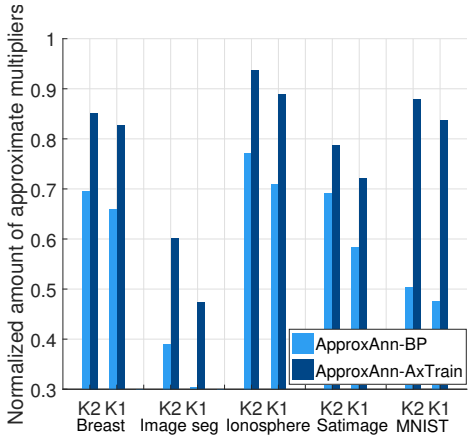

Fig. 11. Number of approximate multipliers used for ApproxAnn-BP and ApproxAnn-AxTrain.

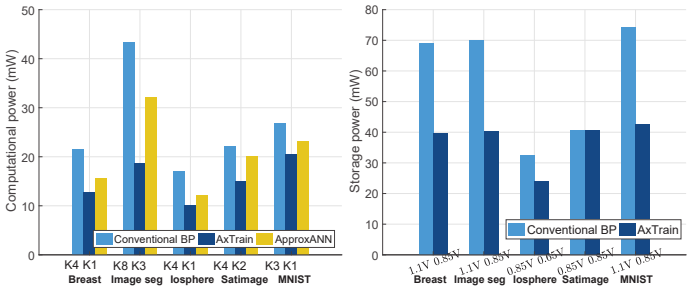

Fig. 12. Power consumption under approximate multiplication and low voltage supplied storage.

in Fig.11. As expected, NN under less aggressive K2 could always employ a larger number of multipliers than under more aggressive K1. ApproxAnn with an AxTrain-trained network could use 20.03% more approximate multipliers on average in the K2 mode and 20.34% more in the K1 mode than ApproxAnn-BP, which means AxTrain helps ApproxAnn to better exploit the power saving opportunity. For AxNN, it measures error contributions of different neurons and statically select neurons for approximate multiplications, then retrains the network for error recovery. AxNN reports the energy improvement over its accurate counterpart under different accuracy constraints (e.g., maximum 2.5% accuracy loss and 7.5% accuracy loss). Specifically, AxNN achieves 1.58X and 1.75X improvement in application energy for the 2.5% and 7.5% quality constraints on average, correspondingly. In this experiment, we evaluate AxTrain in the same way, and the results are listed in Table II. Both the multiplier configurations and the corresponding energy reductions are given under two accuracy constraints. Under the 2.5% and 7.5% constraint,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JETCAS.2018.2845396, IEEE Journal on Emerging and Selected Topics in Circuits and Systems

TABLE II
ENERGY IMPROVEMENT OF AXTRAIN AND AXNN UNDER DIFFERENT ACCURACY CONSTRAINTS

Applications		$< 2.5\%$	$< 7.5\%$
Breast	Reduction	1.91X	1.91X
	MUL Mode	K1	K1
Image	Reduction	1.74X	1.82X
	MUL Mode	K3	K2
Iosp	Reduction	1.87X	1.87X
	MUL Mode	K1	K1
Satimage	Reduction	1.85X	1.89X
	MUL Mode	K2	K1
MNIST	Reduction	2.06X	2.06X
	MUL Mode	K1	K1
AxTrain AVG	Reduction	1.89X	1.91X
AxNN AVG	Reduction	1.58X	1.75X

AxTrain achieves 1.89X and 1.91X energy improvement and outperforms AxNN by 0.31X and 0.16X more reduction, respectively. Note that in AxTrain the energy for the 7.5% accuracy constraint over the 2.5% case improves slightly, this is because in *Breast cancer, Iosp, MNIST* datasets under the 2.5% case, AxTrain already enables the most aggressive approximation mode (K1).

Then, to demonstrate the benefit of AxTrain at the system level, we compare the FlexFlow accelerator's lowest power consumption that approximate computing could attain using AxTrain and conventional BP, while keeping a target accuracy (maximum 2% degradation to accurate implementation), as depicted in Fig.12. On the X axis in this figure, we also show the approximation mode that AxTrain and BP apply. This figure shows that AxTrain's higher noise resilience could result in more aggressive approximation, which leads to lower power consumption than conventional BP. To be specific, computational power and storage power are reduced by 38.78% and 30.81% on average, correspondingly. Notably in the *Satimage* dataset the in low voltage storage case, a conservative voltage of 0.85V is enforced to maintain tight accuracy constraint. By relaxing the allowed degradation to 6%, a 27.01% power reduction was be achieved under 0.66V voltage. We also compare the computational power consumption under approximate multiplier between AxTrain and ApproxANN, and AxTrain requires 22.91% less power consumption than ApproxANN on average.

Finally, in order to evaluate the efficacy of the proposed hardware/software co-design for the accuracy-scalable NN accelerator, we measure the accuracy, power and performance of the accelerator. Specifically, we simulate the accelerator with the variable-latency approximate multipliers working in different accuracy modes with and without AxTrain-guided NN models at the nominal voltage level, as shown in Table III. From the performance and accuracy results, we observe that along with increased approximation level, the performance of NN accelerator improves significantly by 46.34% and 76.47% on average since the multiplication operations

TABLE III
POWER, PERFORMANCE AND ACCURACY OF ACCELERATOR INFERENCE WITH THE VARIABLE-LATENCY MULTIPLIERS W/ AND W/O AXTRAIN

Applications		Mode 0	Mode 1	Mode 2
Breast	Acc w/ (%)	99.12	98.24	95.61
	Acc w/o (%)	97.36	77.19	78.07
	Power (mW)	103.80	95.47	92.40
	Perf (GOPS)	196.92	288.1801	347.51
Image	Acc w/	95.24	95.71	79.76
	Acc w/o	95.47	30.71	27.86
	Power	105.43	96.97	93.85
	Perf	200.0	292.68	352.94
Iosp	Acc w/	94.28	94.25	91.42
	Acc w/o	87.14	35.71	35.41
	Power	83.77	77.10	74.65
	Perf	157.51	230.50	277.96
Satimage	Acc w/	86.32	85.85	76.77
	Acc w/o	88.96	67.52	40.64
	Power	106.99	98.37	95.19
	Perf	203.81	298.26	359.66
MNIST	Acc w/	98.21	98.20	98.1
	Acc w/o	98.0	88.94	77.89
	Power	118.68	109.71	106.41
	Perf	211.95	310.18	374.04

works at higher frequencies. Besides, the energy efficiency also improves by 58.97% and 97.95% on average. For the accuracy part, AxTrain helps to raise the inference accuracy by 34.44% and 36.36% on average in two approximate modes compared with the BP-trained NN models. To further evaluate the potential of the proposed co-design, we evaluate the design incorporating both the approximate multipliers and lower-than-nominal voltage based SRAM weight storage. The accuracy and power results for four different approximation combinations (two low voltage levels with two approximate modes for multipliers) are listed in Table IV. We observe that further with incorporating low voltage based approximation, the inference accuracy degrades only by 2.47% and 1.448% on average for approximate multipliers in M1 and M2 modes, correspondingly. These results prove that proposed AxTrain could help to tolerate the exacerbated noise from the combined noise source. And This observation also indicates with a further relaxed accuracy constraint (2≈3% more allowed degradation), the NN accelerator could achieve lower power consumption with more aggressive approximation, in this case the NN accelerator with the combined approximations consumes 66.85% less power on average than the accelerator with approximate multipliers only. Concluded from this experiment, the accuracy-scalable accelerator with AxTrain could achieve a better trade-off between energy efficiency and inference accuracy, and AxTrain could also be applied to the exacerbated noise scenarios which have a combined noise source.

VII. CONCLUSION

Approximate computing leverages the intrinsic error tolerance of a neural network for improved energy efficiency.

TABLE IV
POWER AND ACCURACY OF ACCELERATOR INFERENCE COMBINING BOTH THE VARIABLE-LATENCY MULTIPLIERS AND LOW VOLTAGE SRAM STORAGE WITH AXTRAIN-GUIDED MODELS

Applications		M1 Con	M1 Agg	M2 Con	M2 Agg
Breast	Acc mean	97.81	96.00	96.35	95.21
	Acc std	0.7571	2.256	0.3248	2.332
	Power	55.4857	50.59	52.41	47.52
Image	Acc mean	96.09	94.08	79.71	77.44
	Acc std	0.5875	0.8451	0.3377	1.4023
	Power	67.157	56.37	64.04	53.2468
Iosp	Acc mean	91.11	88.71	90.43	90.86
	Acc std	1.268	2.602	1.010	2.101
	Power	53.52	44.94	51.06	42.48
Satimage	Acc mean	84.40	81.57	77.09	73.47
	Acc std	0.7000	0.8221	0.3856	0.9340
	Power	68.00	57.02	64.82	53.84
MNIST	Acc mean	97.22	92.81	97.53	90.75
	Acc std	0.0882	0.1666	0.0608	0.2133
	Power	78.15	66.74	74.85	63.43

The most critical challenge of this technique is to how to effectively maintain "good enough" accuracy while at the same time exploiting aggressive approximation. In order to tackle this problem, NN training process should be aware of the functionality of inference hardware instead of just conducting hardware-agnostic training that is solely driven by maximizing accuracy under ideal hardware conditions. In this paper, we propose the AxTrain framework, which trains networks for both accuracy and noise resilience. With active training for sensitivity reduction and passive learning the characteristics of underly approximate hardwares, AxTrain reduces NN's noise sensitivity and improves its resilience against approximation. Moreover, the advantages from AxTrain motivate the design of an accuracy-scalable neural network accelerator which has the flexibility to reconfigure to different accuracy modes. Hence based on the given scalable accuracy requirement, the neural network accelerator can adapt to a certain mode which maximizes energy efficiency. Experimental results for five applications with near threshold voltage SRAM and approximate multiplier based approximate computing techniques demonstrate that our proposed AxTrain framework could lead to more robust networks than conventional hardware-agnostic training frameworks and can be generally applied to diverse approximation techniques. Leveraging this proven AxTrain-guided model, our proposed accuracy-scalable accelerator can achieve better trade-off between inference accuracy and energy efficiency, as supported by our system-level performance, power, and accuracy simulation results.

ACKNOWLEDGMENT

This work was supported in part by Natural Science Foundation Award #1657562 and National Natural Science Foundation of China under Grant No. 61572470.

REFERENCES

[1] P. Dubey, "Recognition, mining and synthesis moves computers to the era of tera," *Technology@ Intel Magazine*, vol. 9, no. 2, pp. 1–10, 2005.

[2] M. Magno, L. Benini, C. Spagnol, and E. Popovici, "Wearable low power dry surface wireless sensor node for healthcare monitoring application," in *IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2013, pp. 189–195.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *computer vision and pattern recognition (CVPR)*, 2016, pp. 770–778.

[4] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[5] S. Han, H. Mao, and W. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *International Conference on Learning Representations (ICLR)*, 2016.

[6] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *International Symposium on Computer Architecture (ISCA)*, 2016.

[7] L. Chen and et al, "Accelerator-friendly neural-network training: Learning variations and defects in rram crossbar," in *Design, Automation and Test in Europe Conference Exhibition (DATE)*, 2017, pp. 19–24.

[8] L. Xia, T. Tang, W. Huangfu, M. Cheng, X. Yin, B. Li, Y. Wang, and H. Yang, "Switched by input: Power efficient structure for rram-based convolutional neural network," in *Design Automation Conference (DAC)*, 2016, p. 125.

[9] J. Miao, K. He, A. Gerstlauer, and M. Orshansky, "Modeling and synthesis of quality-energy optimal approximate adders," in *International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 728–735.

[10] V. Vassiliadis, J. Riehme, J. Deussen, K. Parasyris, C. D. Antonopoulos, N. Bellas, S. Lalis, and U. Naumann, "Towards automatic significance analysis for approximate computing," in *Code Generation and Optimization (CGO)*, 2016, pp. 182–193.

[11] Y. Han, Y. Wang, H. Li, and X. Li, "Enabling near-threshold voltage (ntv) operation in multi-vdd cache for power reduction," in *International Symposium on Circuits and Systems (ISCAS)*, 2013, pp. 337–340.

[12] A. Raha and V. Raghunathan, "qlut: Input-aware quantized table lookup for energy-efficient approximate accelerators," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, p. 130, 2017.

[13] B. Reagen, P. Whatmough, Robert, and et al, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *International Symposium on Computer Architecture (ISCA)*, 2016, pp. 267–278.

[14] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," in *International Symposium on Computer Architecture (ISCA)*, 2012.

[15] Z. Du, A. Lingamneni, Y. Chen, K. Palem, O. Temam, and C. Wu, "Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators," in *ASP-DAC*, 2014, pp. 201–206.

[16] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "Axnn: Energy-efficient neuromorphic systems using approximate computing," in *international symposium on Low power electronics and design (ISLPED)*, 2014, pp. 27–32.

[17] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "Approxann: an approximate computing framework for artificial neural network," in *Design, Automation and Test in Europe Conference Exhibition (DATE)*, 2015, pp. 701–706.

[18] B. Pearlmutter, "Fast exact multiplication by the hessian," *Neural Computation*, vol. 6, no. 1, pp. 147–160, 1994.

[19] A. Weigend, D. Rumelhart, and B. Huberman, "Generalization by weight-elimination with application to forecasting," in *Advances in Neural Information Processing Systems (NIPS)*, 1991, pp. 875–882.

[20] Z. Du, A. Lingamneni, Y. Chen, K. Palem, O. Temam, and C. Wu, "Leveraging the error resilience of neural networks for designing highly energy efficient accelerators," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 34, no. 8, 2015, pp. 1223–1235.

[21] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, 2010.

[22] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *International Symposium on Computer Architecture (ISCA)*, 2016, pp. 27–39.

[23] J. D. M Courbariaux, Y Bengio, "Training deep neural networks with low precision multiplications," in *International Conference on Learning Representations workshop (ICRL)*, 2015.

[24] G. Hinton, "Neural networks for machine learning." Coursera, 2012.

[25] Y. Bengio, N. Lonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," in *arXiv preprint*, 2013.
[26] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks," in *HPCA*, 2017, pp. 553–564.
[27] Y. Wang, Y. Han, H. Li, and X. Li, "Vanuca: Enabling near-threshold voltage operation in large-capacity cache," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 3, pp. 858–870, 2016.
[28] S. R. S Hashemi, R Bahar, "Drum: A dynamic range unbiased multiplier for approximate applications," in *International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 418–425.
[29] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *International Conference on Computer-Aided Design (ICCAD)*, 2011, pp. 694–701.
[30] M. Lichman, "UCI machine learning repository," 2013.
[31] Y. Lecun and C. Cortes, "The mnist database of handwritten digits," 1998.

Xin He received the BEng degree in software engineering from Sichuan University, Chengdu, China, in 2011, and the PhD degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 2017. He is currently a postdoctoral research associate in Washington University in St. Louis. His research interests include computer architecture especially on application specific acceleration, deep learning, neural network accelerator, and approximate computing. He is a member of the IEEE.

Wenyan Lu received the B.Eng. degree in Electrical and Information Engineering from China Agricultural University, in 2012, and the M. Eng. degree in Electronics and Communication Engineering from Beijing Institute of Technology, in 2014. He is currently a Ph.D. Candidate at the State Key Lab. of Computer Architecture, Institute of Computing Technology (ICT), Chinese Academy of Science (CAS). His research interests include heterogeneous computing and deep learning accelerator. He is a student member of IEEE/CCF.

Guihai Yan received the BSc degree in electronics and software engineering (dual-degree) from Peking University, Beijing, China, in 2005, and the PhD degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 2011. He is currently an associate professor with the ICT, CAS. His research interests include computer architecture, heterogeneous computing, domain-specific micro-systems, and computational finance. He is a member of the IEEE and ACM.

Xuan Zhang (S08, M15) is an Assistant Professor in the Preston M. Green Department of Electrical and Systems Engineering at Washington University in St. Louis. She received her B. Eng. degree in Electrical Engineering in 2006 from Tsinghua University in China, and her MS and PhD degree in Electrical and Computer Engineering from Cornell University in 2009 and 2012 respectively. She works across the fields of VLSI, computer architecture, and cyber physical systems and her research interests include adaptive learning hardware for autonomous systems, hardware/software co-design of efficient power delivery and distribution, and ubiquitous self-powered IoT devices.