ApproxANN: An Approximate Computing Framework for Artificial Neural Network

Qian Zhang, Ting Wang, Ye Tian, Feng Yuan and Qiang Xu
CUhk REliable computing laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: {qzhang,twang,tianye,fyuan,qxu}@cse.cuhk.edu.hk

ABSTRACT

Artificial Neural networks (ANNs) are one of the most well-established machine learning techniques and have a wide range of applications, such as Recognition, Mining and Synthesis (RMS). As many of these applications are inherently error-tolerant, in this work, we propose a novel approximate computing framework for ANN, namely ApproxANN. When compared to existing solutions, ApproxANN considers approximation for both computation and memory accesses, thereby achieving more energy savings. To be specific, ApproxANN characterizes the impact of neurons on the output quality in an effective and efficient manner, and judiciously determine how to approximate the computation and memory accesses of certain less critical neurons to achieve the maximum energy efficiency gain under a given quality constraint. Experimental results on various ANN applications with different datasets demonstrate the efficacy of the proposed solution.

1. INTRODUCTION

Inspired by animals' central nervous systems (in particular the brain), Artificial Neural Networks (ANNs), in computer science and related fields, are constructed as systems of interconnected computation nodes (namely "*neurons*"), which can compute output values from inputs by feeding information through the network [1]. While the growing interests in ANNs were hindered with the advent of support vector machines (SVMs) in mid-90s, there was a major resurgence with the introduction of deep networks in recent years [2].

ANNs are computationally expensive for data-intensive applications, which may contain a large amount of neurons and several orders of magnitude larger number of parameters. Consequently, a significant amount of research effort has been spent to implement hardware neural networks in order to achieve high energy-efficiency [3, 4]. Given the fact that emerging RMS applications are inherently error-tolerant with noisy input datasets and/or involving human interfaces with limited perceptual capability, *approximate computing* has been advocated for these applications [5–8]. These works have shown that, by relaxing the computational exactness requirement for neurons in ANNs, we can still achieve minor accuracy loss at the application level while gaining significant energy savings.

In this work, we propose a novel approximate computing framework for ANN, namely *ApproxANN*. To be specific, we first characterize the criticality of each neuron under approximation by jointly considering its impact on output quality and energy consumption, and then utilize an efficient and effective algorithm to determine the approximation for the ANN under a given quality constraint.

The main contributions of this work include:

- ApproxANN considers approximation for both computation and memory accesses, thereby achieving more energy savings when compared to existing approximate ANN designs (e.g., [5, 8]);
- We present a theoretical neuron criticality analysis technique, which can be efficiently applied to neural networks with any topologies;
- We propose a new optimization procedure for approximate ANN construction, wherein error-tolerance capability and energy consumption are jointly considered to achieve the optimized energy savings under a given quality constraint.

The remainder of this paper is organized as follows. In Section 2, we present preliminaries and motivation for this work. Section 3 and Section 4 detail the proposed methodology and experimental results, respectively. Finally, Section 5 concludes this paper.

2. PRELIMINARIES

2.1 Artificial Neural Network

ANNs are usually presented as systems of interconnected computation nodes called *neurons*. Each neuron generates a single output by operating on a vector of inputs, denoted by $\mathbf{x}$ here (usually x_0 is the bias). The input vector is associated with a weight vector (denoted by $\mathbf{w}$), to indicate each entry's numerical significance. In its mathematical representation (see Eq. 1), the neurons will first compute a weighted sum, and then perform a non-linear activation function (e.g., sigmoid function) on the weighted sum to generate the output $g(x)$.

$$g(x) = f(\sum_{i=1}^{d} x_i w_i + w_0) = f(w^t x) \qquad (1)$$

$$f(s) = \begin{cases} 1, \ if \ s \geq 0 \\ -1, \ if \ s < 0 \end{cases} .$$

Typically, an artificial neural network is defined by three types of parameters [1]: (i)The interconnection pattern between the different layers of neurons; (ii) The activation function that converts a neuron's weighted input to its output activation; (iii) The learning process for updating the weights of the interconnections.

2.2 Related Work and Motivation

ANNs are notoriously computationally-intensive, and hence a number of hardware ANN accelerators were proposed in the literature [9, 11, 12]. ANNs, by their nature, are error-resilient. This is because: (i). RMS applications running on ANNs are inherently error-tolerant, which may take noisy dataset as inputs and/or involve human interfaces with limited perceptual capability; (ii). ANN architecture itself is defect-tolerant, as shown by Temam in [13], which is able to tolerate permanent faults provided that the neural network is retrained.

Approximate computing is an emerging design paradigm that is able to tradeoff computation quality (e.g., accuracy) and

978-3-9815370-4-8/DATE15/©2015 EDAA

computational effort (e.g., energy) by exploiting the error-resilience properties of applications [14]. Various approximate computing techniques, spanning from transistor-level designs [15] to architecture-level designs [6], were proposed to achieve improved performance and/or energy efficiency.

With the above, it is natural to apply approximate computing for ANNs to achieve energy savings without sacrificing much solution quality. With many approximate arithmetic building blocks presented in the literature (e.g., [16]), some recent works proposed to design approximate ANNs with approximate neuron designs. In [5], Du *et al.* first designed approximate neurons with approximate multipliers, and then search a subset of substitutions to choose the "best" configuration for the approximate ANN design. However, this is a trial-and-error process instead of a systematic solution.

Recently, Venkataramani *et al.* [8] proposed a systematic approximate ANN design methodology, namely *AxNN*. In this work, the final error of the NN is first back propagated to get error apportions of each individual neuron. Neurons are then sorted based on the magnitude of their average error contribution, and classified as resilient or sensitive ones with a pre-determined threshold. Those resilient neurons are then designed as approximate ones for energy savings. Finally, retraining is performed to mitigate the impact of inexact hardware on the solution quality. One problem with the above neuron criticality analysis technique is that the solutions to calculate error apportions on each neuron are not unique. Figure 1 is a simplified (last) 2-layer neural network, and the error contributions of single neuron must satisfy $W^T e = o - t$, where k is the number of neurons in relative first layer, e is the errors, o and t are the outputs and golden results, respectively. As weight matrix W is a $4 \times k$ matrix, the rank of W is at most 4, which is definitely less than k. Consequently, it yields a large number of solutions for e and the criticality ranking for neurons may not be trustworthy. However, due to the error-resilience capability of ANN and the effective retraining procedure, the experimental results in [8] are quite encouraging.

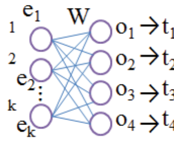


Figure 1: Calculating the Error Apportions.

The few existing approximate ANN designs[1] try to approximate computational unit in neurons only, while memory access may consume a significant portion of energy in ANNs [9]. Moreover, their optimization policies are rather ad-hoc (e.g., using a pre-defined threshold).

The above has motivated the proposed ApproxANN framework presented in this paper.

3. PROPOSED DESIGN METHODOLOGY

3.1 Neuron Criticality Analysis

We say one neuron is critical, if small jitter on this neuron's computation introduces large final output quality degradation; otherwise, it is resilient. To determine each neuron's criticality, intuitively, we can inject random error on each neuron and record its influence on final output. However, this method is not practical because single neuron's effect is too small to be observed for large scale networks. In this section, we will present our efficient theoretical-based criticality analysis for neurons in output layer and hidden layers.

[1]Note that, there are also some research efforts that attempt to improve the energy-efficiency of ANNs by optimizing the bit-width when mapping the neuromorphic system on FPGA platform(e.g., [3, 4]), which can be seen as an implicit approximate solution for ANNs.

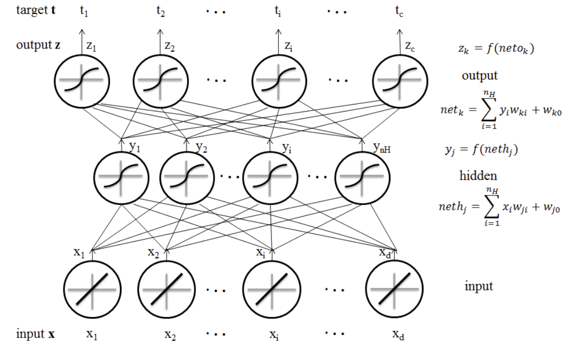


Figure 2: Notations in Neural Network.

To support our criticality analysis, we have the notations illustrated in Figure 2. And the final network's cost function can be described as:

$$E = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|t - z\|^2$$

Consequently, the $i-th$ neuron's criticality, denoted by nc_i (includes nco_i and nch_i for output neurons and hidden neurons, respectively), can be represented by the derivative of E with respect to net_i, as illustrated by Eq.2.

$$nc_i = \frac{\partial E}{\partial net_i}. \tag{2}$$

For the neurons in output layer:

$$\begin{aligned} nco_k &= \frac{\partial E}{\partial neto_k} \\ &= \frac{\partial \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2}{\partial z_k} \cdot \frac{\partial z_k}{\partial neto_k} \\ &= -(t_k - z_k) \cdot f'(neto_k) \end{aligned} \tag{3}$$

For the neurons in hidden layers:

$$\begin{aligned} \frac{\partial E}{\partial y_j} &= \frac{\partial}{\partial y_j}[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2] \\ &= -\sum_{k=1}^{c}(t_k - z_k) \cdot \frac{\partial z_k}{\partial neto_k} \cdot \frac{\partial neto_k}{\partial y_j} \\ &= -\sum_{k=1}^{c} nco_k w_{kj} \\ nch_j &= \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial neth_j} \\ &= -f'(neth_j) \cdot \sum_{k=1}^{c} nco_k w_{kj} \end{aligned} \tag{4}$$

As illustrated by Eq.3 and Eq.4, we can efficiently calculate the criticality factor of neurons in each output layer and hidden layers after training phase fixes all the weights, even for large-scale neural networks.

Based on this definition, we say a neuron is less "critical" if its accuracy requirement relaxation leads to less final quality degradation. A less "critical" neuron will have a higher priority to be approximated. Based on this analysis, we sort all the output and hidden neurons in ascending order, and finally get the criticality ranking vector $s = \{s_1, s_2, ..., s_n\}$ for a given network. Every entry s_m indicates that the corresponding neuron has the $m-th$ priority to be approximated.

3.2 Approximate Artificial Neural Network Architecture

3.2.1 Neural Network Architecture Overview

As presented in Sec. 2.1, each hidden/output neuron computes dot product of weight vector w and input vector x, followed by an activation function. Neurons in the same layer share the same input vector x, and their different weight vectors ws form the weight matrix between the current layer and the pervious layer. In our implementation, we use the widely-used hardware architecture ([17]) shown in Figure 3, in which each processing element (PE), working as a neuron, is comprised of a series of arithmetic units and a local memory (LM), and it only reads the corresponding weight vector from off-chip memory to its local memory. By streaming vector x through each neuron in the same layer, matrix-vector multiplication is performed naturally (with each neuron computing the dot product of two vectors).

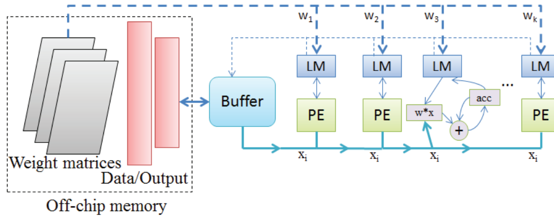


Figure 3: Mapping Neuron to PE.

3.2.2 Approximate Design Choices

For a given $d - n_H - c$ network shown in Figure 2 with n data samples, the number of off-chip to on-chip data transfer DT is given as

$$DT = dn_H + n_H c + n(d + c).$$

And the computation workload CW, in terms of the number of multiply-accumulate operations is

$$CW = (dn_H + n_H c)n.$$

Suppose energy consumption for read/write 1unit data[2] and 1unit ALU operation are A and B respectively. The ratio α between memory energy consumption and computation energy consumption can be represented as

$$\begin{aligned} \alpha &= \frac{(dn_H + n_H c + n(d + c)) * A}{(dn_H + n_H c)n * B} \\ &= (\frac{1}{n} + \frac{1}{n_H}) * \frac{A}{B}, \end{aligned} \quad (5)$$

which is determined by the number of samples in the dataset n, network structure n_H, as well as hardware characteristics A and B. As such ratio vary a lot with different applications, we need to consider both memory accesses and computations when applying approximate computing. Albeit any approximate design techniques can be integrated into ApproxANN, in this work, we use the following three techniques to illustrate our proposed design methodology.

Memory Access Skipping. As the individual PEs can decide whether an operations can be skipped or not, to save energy consumption on memory access and improve overall performance, we can easily skip several neurons (i.e., just skip reading specific rows in weight matrix), if they are regarded as uncritical in our previous criticality analysis.

[2]1unit means basic read/write and operation component, for example, 16 bits data

Precision Scaling. The most intuitive method for energy-accuracy tradeoff in ANN is to control the data bit-width. When simply discarding a specific number of least significant bits (LSBs) of the data, computational quality is slightly degraded, but energy saving and performance can be significantly improved.

Approximate Arithmetic Building Blocks. Another method is to replace accurate arithmetic untis in a neuron with approximate ones. Without loss of generality, we resort to the approximate multiplier proposed in [19] for neuron approximation [3]. This multiplier has a tunable output of $(n + k)$ bits, where n represents the bit-width of input data. By using different values of parameter k, we can tradeoff computational accuracy and energy consumption.

3.3 Optimization Procedure

3.3.1 Problem Formulation

Denoting En_i is the energy consumption of $i - th$ neuron in the criticality ranking vector, $\triangle Q$ is the quality degradation on final output (i.e., growth on network's cost function), E is the error function on final output given in Sec. 3.1, and C is the maximum percentage of allowed quality degradation, our problem can be formulated as:

$$\begin{aligned} &min &&En = \sum_{i=1}^{n} En_i \\ &s.t. &&\triangle Q \leq E * C \\ & &&s_1 \geq s_2 \geq ... \geq s_{n*(k+1)} \end{aligned} \quad (6)$$

If neurons are independent of each other, the above problem can be transformed to the well-known knapsack problem and then solved by many efficient algorithms (e.g. dynamic programming).

However, neurons are tightly inter-connected in a given network. As a result, the criticality ranking usually changes after approximation of every single neuron, and re-sorting after each replacement will definitely cause unaffordable overhead. To tackle this problem, in this paper, we adopt iterative heuristics (detailed in the following subsections) to determine which neurons to be approximated, and how to approximate them.

3.3.2 Adaptive Batch Replacement

Inspired by adaptive step size in gradient methods (e.g., Gradient Descent, etc.), an adaptive batch replacement policy is proposed to solve the formulated problem.

As shown in Figure 4, at earlier steps we have a relatively larger budget for quality degradation, and we can relax computation or read/write requirements for more neurons in each step. When we are approaching the convergence of the algorithm, the budget becomes smaller and we can only approximate few neurons. Therefore, the batch size should be adaptive based on the changing quality budget.

Detailed description of this algorithm is illustrated in Algo. 1, in which Lines 5-7 determine the exit condition of the program when we have no quality budget. Line 8 guarantees to target neurons which haven't been selected yet in the previous steps. Lines 9-12 perform the neuron approximation and add the selected neurons to result list. After the error healing process in Line 13, Line 14 determines whether we need an extra iteration to refine the result or not.

In this proposed algorithm, we first determine the batch size at the beginning of each iteration. Selecting the optimized batch size cnt plays a vital role in the whole procedure, because the efficiency of our algorithm is highly dependent on it in each iteration. If the size is optimal in each step, the total iteration

[3]Similar to [5], we do not consider to apply approximation for the adder due to the small energy gains.

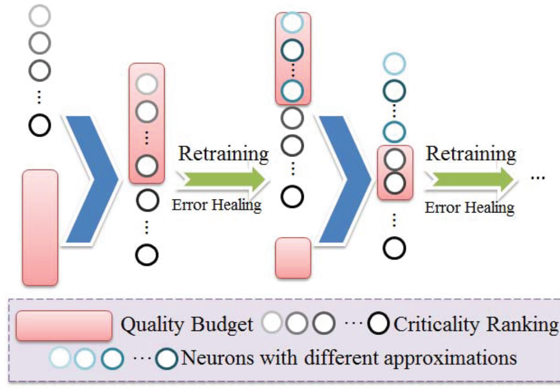


Figure 4: Adaptive Batch Replacement.

```
    Function: BatchReplacement
    input   : N, C, s
    output  : Result Vector r
 1  CurrentDegradation← 0;
 2  ExitFlag← 0;
 3  while CurrentDegradation<E*C do
 4  |  [cnt,tempACModes] ← DetermineBatchSize
    |  (C,s,CurrentDegradation,r);
 5  |  if cnt == 0 then
 6  |  |  ExitFlag← 1;
 7  |  end
 8  |  tempNeurons ← fist cnt accurate neurons from s.top() ;
 9  |  for i←1, i≤ cnt, i++ do
10  |  |  replace tempNeurons.at(i) with
    |  |  tempACModes.at(i) approximation mode;
11  |  |  r.add(tempNeuron.at(i), tempACModes.at(i));
12  |  end
13  |  perform retraining and testing;
14  |  CurrentDegradation← △E;
15  |  if ExitFlag = 1 then
16  |  |  return r;
17  |  end
18  end
```

Algorithm 1: Proposed Algorithm Framework

numbers and the overhead we described in Sec. 3.3.1 will be minimized consequently. In the following, we discuss how to adjust the batch size on-the-fly.

3.3.3 Batch Size Determination

Our proposed adaptive batch size selection algorithm is shown in Algo. 2. Line 1 gives the current quality budget. To perform a greedy search procedure, in lines 11-13 we take the largest batch size each time that satisfies the quality constraint. According to the classification nature of neural networks, total error introduced by n neurons is not greater than the sum of errors introduced by each of these n neurons. Therefore we can use the sum of quality impact of a batch of neurons for replacement guidance. And the iterative procedure in Algo. 1 will fix the effects of estimated errors on batches.

Besides getting the number of neurons that can be approximated, each neuron's approximate mode must be determined wisely. Related works in the literature usually set a threshold manually, then replace all the neurons under this threshold by the same kind of approximate counterpart, without optimizing the energy saving under a given quality constraint. To achieve this target and select approximate mode adaptively, we first characterize different approximate configurations to get their impacts on energy saving and quality degradation, and then propose our novel concept of "saving per loss".

Denoting all the possible approximate design candidates as $A = \{AC_0, AC_1, ..., AC_k\}$, where AC_0 represents no memory read/write operation while the others are different kinds of circuit-level approximate neuron designs with various accuracy-energy tradeoffs. We define "quality impact" and "energy impact" for each configuration as follows:

```
    Function: DetermineBatchSize
    input   : CurrentDegradation,r
    output  : Batch Size t, Approximation modes related to
              each neuron to be replaced Modes
 1  Capacity = E*C-CurrentDegradation;
 2  do criticality analysis;
 3  sum ← 0;
 4  t ← 0;
 5  for i←,i≤ s.lengh(),i++ do
 6  |  if r.found(s.at(i)) then
 7  |  |  continue;
 8  |  end
 9  |  [tempLoss, tempMode]←ModeSelection (s.at(i));
10  |  sum += tempLoss;
11  |  if sum>Capacity then
12  |  |  break;
13  |  end
14  |  t++;
15  |  Modes.add(tempMode);
16  end
17  return [t,Modes]
```

Algorithm 2: Batch Size Selection

Table 1: Hardware Characterization Illustration

Impact \ Configuration	AC_0	AC_1	...	AC_n
quality impact	ϵ_0	ϵ_1	...	ϵ_n
energy impact	en_0	en_1	...	en_n

Definition 1 *For any $AC_i \in AC$, the "quality impact", denoted by ϵ_i, is defined as the relative difference between accurate and approximate computations for dot product of two vectors.*

Definition 2 *For any $AC_i \in AC$, the "energy impact", denoted by en_i, is defined as the energy savings between accurate computation and this specific approximate computation on a single neuron computation.*

By running representative workload, naturally we can estimate the above impacts of each hardware configurations, and obtain Table 1

Now we propose our novel measure by combining both "quality impact" and "energy impact" to achieve optimized energy savings per quality degradation.

Definition 3 *The "saving per loss", denoted by sp for neuron i with approximation candidate AC_j, is defined as energy savings per quality degradation of final output of the network, and has the following form:*

$$sp_i^{AC_j} = \frac{en_j}{\epsilon_i * nc_j}$$

Based on this definition, Line 3-4 in Algo. 3 show the calculation of our defined "saving per loss". And Line 5-9 return the most energy saving mode for this given neuron under per quality degradation.

With all of the procedures above, we can construct an approximate hardware neural network effectively and efficiently with optimized energy savings under given quality constraint.

4. EXPERIMENTAL RESULTS

```
    Function: ModeSelection
    input    : neuron j
    output   : Final quality loss loss, Optimal approximation
               modes related to this neuron mode
 1  sp←0; loss←0;
 2  for i←0,i≤ AC.lengh(),i++ do
 3  |   tempLoss←nc_j*ε_i;
 4  |   tempsp←en_i/loss;
 5  |   if sp<tempsp then
 6  |   |   sp←tempsp;
 7  |   |   mode← i;
 8  |   |   loss←tempLoss;
 9  |   end
10  end
11  return [loss,mode]
```

Algorithm 3: Neuron Approximation Mode Selection

4.1 Experimental Setup

To evaluate the effectiveness and efficiency of the proposed $ApproxANN$, we performed the simulation based on $45nm$ standard cell library with 1V supply voltage by commercial Synopsys EDA tools. The energy includes both the dynamic and leakage power. The power values of different approximate components were obtained by using Synopsys PrimeTime, and energy consumptions on memory read/write were obtained by CACTI [18].

Then we perform experiments on 4 representative networks with various datasets. Table 2 describes the detailed information of these applications and datasets, wherein the first one is a typical learning algorithm for neural networks on hand writing digit recognition, followed by two convolutional neural networks on face recognition and pedestrian detection, while the last one is neural network based method for financial analysis applications.

4.2 Energy Benefits of ApproxANN

Our first experiment is conducted to compare original energy consumptions on memory and computation, and to evaluate the energy benefits of ApproxANN against the accurate implementation.

The numerical results for our used examples with different quality loss constraint are presented in Table 3, wherein "M/C" is the ratio of energy consumptions between memory and computation without approximation, "Constraint" gives the predefined constraint on final output quality loss, "Q. Loss" is the real quality loss, "$E.B.(C)$", "$E.B.(M)$", and "$E.B.(T)$" indicate energy savings on computation part, on memory part, and on total application, respectively.

First of all, our designed hardware neural network can satisfy the given output constraints with notable energy gains. As illustrated in the rightmost column in Table 3, the detailed energy savings depend on the applications and datasets, and we obtain 35.28% and 51.72% energy benefits for the recognition on digit dataset and face dataset, respectively. And second, for different combination of network topology and dataset, the energy consumptions on memory and computation vary significantly. The ratio between these two parts (column "M/C") is 0.152 for the pedestrian detection, while it becomes 71.630 for the financial prediction. Here we want to emphasis that it is necessary to consider both memory and computation to get better energy efficiency on hardware neural network designs.

4.3 Results Comparison

Our second experiment is performed to compare the energy/quality tradeoff of the proposed $ApproxANN$ with the conventional strategy in [5], and the latest approach $AxNN$ proposed in [8].

Table 4 shows the results by using conventional approach ([5]) in terms of energy savings and quality loss tradeoff. In such conventional strategy, all of the neurons are replaced by same approximate versions (i.e., all neurons with single mode approximation) without considering memory's impact. For fair comparison, the same network topology and parameter initialization are used, and 6 different approximate versions are compared in each of the 4 applications. Configuration PS_k in the "Config." column means the calculations are performed on data represented with k bits, while MP_j means the multiplier output contains j bits for 16 bit-input.

By comparing the results of conventional (Table 4) and our (Table 3) approaches, we have the following observations. First, we can hardly predict the final quality loss by replacing all of the neurons(i.e., column "Q.Loss" in Table 4), which means the conventional approach cannot provide any guarantee on final output quality and hence is not a viable solution. If 10% is the desired quality loss for $CALTECH$, we can see that all of the configurations have violated the constraint except MP_{24}, however, our designed network can meet any predefined quality constraint (Table 3 and Figure 5). Second, the energy savings on computation and the energy savings on total application are two different stories. For instance, the application $MNIST$ with MP_{24} configuration can achieve 45.92% energy savings on computation, while it becomes 5.63% on the total application, because Table 3 shows that the energy consumptions on memory for this application is around 7.16 times on computation. Third, for computation-dominated networks, conventional approach can get notable energy savings on total application (e.g., column $E.B.(T)$ for $CALTECH$ in Table 4). However, our designed network ($CALTECH$ in Table 3) can provide better guarantee on final output quality.

To tackle the quality constraint problem in conventional approach, $AxNN$ in [8] will first sort the resilience capability of all the neurons, and then only a fraction of neurons will be replaced to satisfy a pre-defined quality loss threshold. Figure 5 compares the results with [8] in terms of energy saving percentages under different quality constraints, and notable more energy savings, especially for memory-dominated networks ($MNIST$ and $NASDAQ$), than $AxNN$ are reported.

5. CONCLUSION

Artificial neural networks are state-of-the-art machine learning techniques for a wide range of applications. Many of these applications are error-resilient, where approximate computing are naturally used to achieve great energy savings with minor quality degradation. In this paper, ApproxANN approximates the computation and memory accesses of certain less critical neurons based the criticality analysis to obtain energy efficiency under quality constraint. With the proposed solution, ApproxANN achieves 34.11% $\sim$ 51.72% energy benefit with less than 5% quality loss for various neural network applications used in our experiments.

6. ACKNOWLEDGEMENT

This work was supported in part by the Hong Kong S.A.R. General Research Fund (GRF) under Grant 418112, and in part by National Natural Science Foundation of China (NSFC) under Grant 61432017.

7. REFERENCES

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 9, pp. 533–536, 1986.

[2] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[3] Y. Lee, Y. Choi, S.-B. Ko, and M. H. Lee, "Performance analysis of bit-width reduced floating-point arithmetic units in FPGAs: a case study of neural network-based

Table 2: Datasets and Parameters Settings

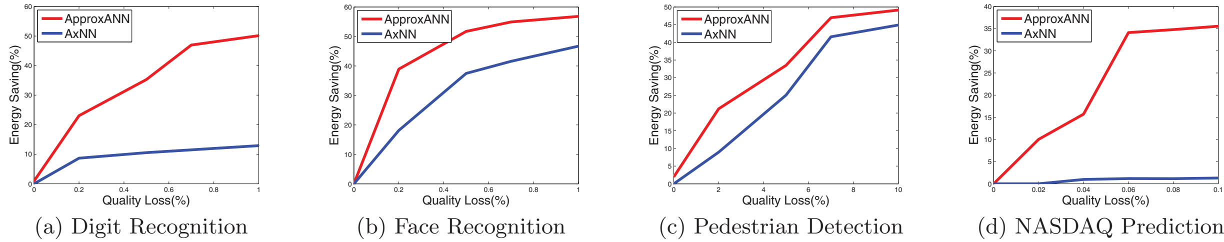
Dataset	Application	Source	Samples	Feature Dimension
The MNIST Database	Hand Writing Digit Recognition	NYU	10,000	400(20*20 image)
The CIFAR-10 Dataset	Face Recognition	UToronto	60,000	1,024(32*32 image)
Caltech	Pedestrian Detection	CALTECH	100,000	7,056(84*28*3 image)
NASDAQ	Financial Prediction	Yahoo!	10,799	10

Table 3: Experimental Statistics

Application	M/C	Constraint	Q. Loss	E.B.(C)	E.B.(M)	E. B.(T)
MNIST Hand Writing Recognition	7.160	0.5%	0.4891%	83.32%	28.57%	35.28%
CIFAR-10 Face Recognition	1.005	0.5%	0.5000%	70.85%	32.67%	51.72%
Pedestrian Detection	0.152	5%	4.977%	52.89%	19.78%	48.51%
Nasdaq Prediction	71.630	0.06%	0.0499%	89.96%	33.33%	34.11%

Table 4: Results: Conventional Single Mode Approximation Approach

Dataset	Fixed Bit-Width Implementation					Approximate Multiplier Implementation				
	Config.	Q. Loss	E. B.(C)	E.B.(M)	E.B.(T)	Config.	Q. Loss	E. B.(C)	E.B.(M)	E.B.(T)
MNIST	PS_8	0.7483%	75.08%	0	9.20%	MP_24	0.4627%	45.92%	0	5.63%
	PS_6	2.2039%	85.87%	0	10.52%	MP_20	1.1114%	70.66%	0	8.66%
	PS_4	4.1893%	93.66%	0	11.48%	MP_18	7.6201%	83.35%	0	10.21%
CIFAR-10	PS_8	1.0882%	75.08%	0	37.45%	MP_24	0.6480%	45.92%	0	22.90%
	PS_6	1.7211%	85.87%	0	42.83%	MP_20	1.2884%	70.66%	0	35.24%
	PS_4	3.8422%	93.66%	0	46.71%	MP_18	5.4001%	83.35%	0	41.57%
CALTECH	PS_8	13.6417%	75.08%	0	73.39%	MP_24	9.0667%	45.92%	0	44.89%
	PS_6	19.8667%	85.87%	0	83.94%	MP_20	18.95%	70.66%	0	69.07%
	PS_4	30.925%	93.66%	0	91.55%	MP_18	27.5083%	83.35%	0	81.47%
NASDAQ	PS_8	0.1179%	75.08%	0	1.03%	MP_24	0.0730%	45.92%	0	0.63%
	PS_6	0.1766%	85.87%	0	1.18%	MP_20	0.1558%	70.66%	0	0.97%
	PS_4	0.3102%	93.66%	0	1.29%	MP_18	0.2088%	83.35%	0	1.15%



(a) Digit Recognition (b) Face Recognition (c) Pedestrian Detection (d) NASDAQ Prediction

Figure 5: Comparison Between ApproxANN and AxNN [8]

face detector," *EURASIP Journal on Embedded Systems*, pp. 4, 2009.

[4] L.-W. Kim, S. Asaad, and R. Linsker, "A fully pipelined fpga architecture of a factored restricted boltzmann machine artificial neural network," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, no. 1, pp. 5:1–5:23, 2014.

[5] Z. Du et al. "Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators," in *Proc. of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 201–206, IEEE, 2014.

[6] H. Esmaeilzadeh et al. "Neural acceleration for general-purpose approximate programs," in *Proc. of International Symposium on Microarchitecture (Micro)*, pp. 449–460, IEEE Computer Society, 2012.

[7] Y. Kim, Y. Zhang, and P. Li, "An energy efficient approximate adder with carry skip for error resilient neuromorphic vlsi systems," in *Proc. of International Conference on Computer-Aided Design (ICCAD)*, pp. 130–137, IEEE Press, 2013.

[8] S. Venkataramani et al. "Axnn: Energy-efficient neuromorphic systems using approximate computing," in *Proc. of The International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 27–32, ACM, 2014.

[9] T. Chen et al. "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. of Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 269–284, ACM, 2014.

[10] C. Farabet et al. "Hardware accelerated convolutional neural networks for synthetic vision systems," in *Proc. International Symposium on Computer Architecture (ISCA)*, pp. 257–260, IEEE, 2010.

[11] B. Li, et al. "Training itself: Mixed-signal training acceleration for memristor-based neural network," in *Proc. of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 361–366, 2014.

[12] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," in *Proc. International Symposium on Computer Architecture (ISCA)*, pp. 356–367, 2012.

[13] Q. Zhang, F. Yuan, R. Ye, and Q. Xu, "ApproxIt: An Approximate Computing Framework for Iterative Methods," in *Proc. Design Automation Conference (DAC)*, pp. 1–6, 2014.

[14] V. Gupta et al. "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design*, vol. 32, no. 1, pp. 124–137, 2013.

[15] R. Ye, T. Wang, F. Yuan, R. Kumar and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *Proc. of International Conference on Computer-Aided Design (ICCAD)*, pp. 48–54, 2013.

[16] A. Majumdar et al. "A massively parallel, energy efficient programmable accelerator for learning and classification," in *ACM Transactions on Architecture and Code Optimization*, vol. 9, no. 1, 2012.

[17] H. Labs. `http://www.hpl.hp.com/research/cacti/`.

[18] E. J. King and E. Swartzlander, "Data-dependent truncation scheme for parallel multipliers," in *Proc. Asilomar Conference on Signals, Systems & Computers*, vol. 2, pp. 1178–1182, 1997.