Design of Power-Efficient Approximate Multipliers for Approximate Artificial Neural Networks

Vojtech Mrazek[1]
imrazek@fit.vutbr.cz

Syed Shakib Sarwar[2]
sarwar@purdue.edu

Lukas Sekanina[1]
sekanina@fit.vutbr.cz

Zdenek Vasicek[1]
vasicek@fit.vutbr.cz

Kaushik Roy[2]
kaushik@purdue.edu

[1]Faculty of Information Technology, Centre of Excellence IT4Innovations
Brno University of Technology
Brno, Czech Republic

[2]School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA

ABSTRACT

Artificial neural networks (NN) have shown a significant promise in difficult tasks like image classification or speech recognition. Even well-optimized hardware implementations of digital NNs show significant power consumption. It is mainly due to non-uniform pipeline structures and inherent redundancy of numerous arithmetic operations that have to be performed to produce each single output vector. This paper provides a methodology for the design of well-optimized power-efficient NNs with a uniform structure suitable for hardware implementation. An error resilience analysis was performed in order to determine key constraints for the design of approximate multipliers that are employed in the resulting structure of NN. By means of a search based approximation method, approximate multipliers showing desired tradeoffs between the accuracy and implementation cost were created. Resulting approximate NNs, containing the approximate multipliers, were evaluated using standard benchmarks (MNIST dataset) and a real-world classification problem of Street-View House Numbers. Significant improvement in power efficiency was obtained in both cases with respect to regular NNs. In some cases, 91% power reduction of multiplication led to classification accuracy degradation of less than 2.80%. Moreover, the paper showed the capability of the back propagation learning algorithm to adapt with NNs containing the approximate multipliers.

Categories and Subject Descriptors

B.6.3 [**Hardware**]: Logic Design—*Automatic synthesis*; I.2.6 [**Computing Methodologies**]: Artificial Intelligence

ICCAD '16, November 07-10, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2966986.2967021

1. INTRODUCTION

Recent advances in artificial intelligence methods and a huge amount of computing resources available on a single chip have led to a renewed interest in efficient implementations of complex neuromorphic systems based on artificial neural networks (NNs). Implementing complex NNs in low power embedded systems requires careful optimization strategies at various levels including neurons, interconnects, learning algorithms, data storage and memory access. This work is focused on reducing power consumption of computations performed in neurons, which is of the same importance as optimizing the data storage and memory access [7].

Inexact or approximate computing has been adopted in recent years as a viable approach to reduce power consumption and improve the overall efficiency of computers. In approximate computing, circuits are not implemented exactly according to the specification, but they are simplified in order to reduce power consumption or increase operation frequency. It is assumed that the errors occurring in simplified circuits are acceptable, which is typical for error resilient application domains such as multimedia, classification and data mining. Applications based on NNs have proven to be highly error resilient [2].

This paper provides a methodology for the design of well-optimized power-efficient NNs that have a uniform structure (i.e. all nodes are identical in all layers) which is thus suitable for hardware implementation. An error resilience analysis is performed in order to determine key constraints for the design of approximate multipliers that are employed in the resulting structure of NN. In order to avoid a manual approximation of accurate multipliers, systematic methods capable of performing approximations have been introduced recently [21, 20, 14]. These methods typically start with a gate-level description of the accurate circuit and an error constraint that specifies the type of error that can be accepted. The approximation algorithm is typically constructed as a design space exploration algorithm directly approximating some parts of the circuit [11] or the whole circuit [18]. The search is guided by an error metric such as the average error magnitude or maximum arithmetic error.

In addition to developing highly-optimized power efficient

NNs, an automated design space exploration method is proposed. The method is capable to design approximate multipliers in such a way that the resulting multipliers satisfy not only a given error, but also a set of other application-specific constraints.

2. ARTIFICIAL NEURAL NETWORKS

In machine learning, artificial neural networks are a family of models inspired by biological neural networks. A typical artificial neural network consists of an input layer of neurons, several hidden layers of neurons and an output layer of neurons.

2.1 Artificial Neuron

A typical structure of neuron is as follows [4, 22]. The output h_i of neuron i is defined as $h_i = \sigma(\sum_{j=1}^{N} w_{ij}x_j - \theta)$, where $\sigma(\cdot)$ is an activation function, N is the number of inputs of the neuron, w_{ij} is weight of the link, x_j is the j-th input and θ is a threshold or bias. The purpose of the activation function is (in addition to introducing non-linearity into NN) to map the resulting values into the interval $(-1, 1)$ or $(0, 1)$. The activation can be a threshold function, semi-linear or non-linear function. A common example of the non-linear function, which is used in this work, is sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$.

2.2 Architecture and learning

The NNs are classified into feed-forward neural networks (FNNs), recurrent neural networks (RNN) and their combination. In RNNs, there is at least one feedback connection. The earliest and the simplest architecture is the perception model which utilizes just one layer of output neurons that are connected with all the inputs. The extended version, the multilayer perception model (MLP), uses one or more layers (a.k.a. hidden layers) of neurons between the input and output layers. In the hidden layer, each neuron is directly linked to the outputs of the previous layer. An important contribution to the state of the art in NNs has been the development of large-scale NNs such as the convolutional NNs introduced by LeCun [9], where more types of layers (e.g. convolutional layers) are employed. Another type of layers is the average pooling layer which is used for weighted subsampling. Nowadays there are many different application-specific layers intended for, e.g., image classification [8], segmentation [1], speech processing [6] etc.

Learning is the most important capability of neural networks. It is performed by an algorithm that iteratively updates the synapses (weights) and other parameters of neural network. Determining the most suitable parameters and weights of NN can be viewed as a complex nonlinear optimization problem. Learning methods are usually divided into supervised, unsupervised, reinforcement, and evolutionary methods [4]. The most popular algorithms for supervised learning, which we will employ, are the least mean squares method and back propagation algorithm [4].

2.3 Approximations in NNs

As neural networks are inherently error-resilient, various approaches have been proposed to approximate them [13].

Venkataramani et al. [19] proposed a methodology of identifying error-resilient neurons based on the backpropagation gradients. For the error-resilient neurons, an approximation using precision modification and piecewise-linear approximation of activation function was applied to create an approximate neural network. Since training is by itself an error-healing process, after creating the approximate version, the NN is retrained. They also proposed a neuromorphic processing engine platform to determine the best tradeoff between the precision and energy.

Zhang et al. [24] used a different approach for critical neuron identification. A neuron is considered as critical, if small jitters on the neuron's computation introduce large output quality degradation; otherwise, the neuron is resilient. They presented a theoretical approach for finding the critical neurons. The least critical neurons are candidates for approximation. Due to the tight interconnection between the neurons, the ranking of candidate neurons is updated after approximation of each neuron. Hence, an iterative algorithm for the criticality ranking and approximation was developed. Three approximation strategies were used – precision scaling, memory access skipping and approximating the multiplier circuits.

Du et al. [5] proposed an inexact Neural Network accelerator showing that it is possible to use inexact multipliers in NNs. The multipliers were designed using an inexact logic minimization algorithm [11]. For small fully connected neural networks, their strategies were able to find good configurations. They exploited the fact that the output layer has a small number of neurons and since there is no synaptic weight after these neurons, lowering the errors through retraining is difficult [13].

Judd et al. [7] showed that computations and memory accesses significantly contribute to power consumption. Hence they used bit-precise weights reduction in standard multiplication and reduced memory accesses of standard memories in their implementation for GPUs and ASIC.

Power consumption of the synaptic weight memory was optimized by Srinivasan et al. [17] who applied a conventional 6T SRAM that is known to be susceptible to bit-cell failures due to voltage over-scaling. A significance driven hybrid 8T-6T SRAM was proposed wherein the sensitive MSBs are stored in robust 8T bit-cells. The memory access power reduction was exchanged for a small loss in the classification accuracy.

Sarwar et al. [16] introduced approximate multipliers based on alphabet-set multiplication. The weights were divided into parts having 4 bits. Multiplication by each 4-bit part of the weight was implemented by shifting a precomputed input value and followed by summation. Authors showed that reducing the set of precomputed values has a significant impact on power consumption and a small impact on the total accuracy of neural network. This architecture is suitable for an efficient hardware implementation because the resulting NN shows a uniform structure and each neuron has the same architecture.

In summary, the first four approaches presented in this section have shown that it is possible to approximate some neurons. The resulting NNs can be characterized as non-uniform NNs. However, for an efficient VLSI implementation and for implementing a general-purpose NN (not an application specific one), all (or almost all) neurons have to be uniform. Moreover, the selected components were approximated manually and independently of a target NN. It was also shown that not only multiplication but also the

memory access has a significant impact on the total power consumption.

2.4 Approximate multipliers in NNs

Since NN contains hundreds of thousands multiplications, it seems to be useful to introduce approximate multipliers to reduce power consumption. In order to determine the impact of inexact multiplication on NNs' accuracy, the following sensitivity analysis has been carried out.

A non-trivial MLP network (1 hidden layer, 100 hidden neurons) trained for recognizing handwritten numbers of MNIST dataset (described in Section 4.2.1) was chosen as our benchmark problem and evaluated using *DeepLearnToolbox*.[1]. Its accurate implementation shows the classification accuracy 94.16% when precise 8-bit multipliers are used.

To emulate imprecise multiplication, a jitter function $\Delta : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is introduced. Let the output of inexact multiplier m be defined as $m(a,b) = a \cdot b + \Delta(a,b)$. To ensure that the relative worst-case error of 8-bit multiplier m is 5.2%, the range of the jitter function Δ is bounded by ± 852, calculated as $5.2\% \cdot 2^{2 \cdot 7}$. Note that this worst-case error was chosen according to approximate multipliers proposed in [18].

When function m is used instead of accurate multiplication and no retraining is applied, the classification accuracy of the network decreased to 10.77%. A detailed analysis revealed that there are more than 80% cases where one of the input operands of multiplication is zero. The random jitter then provides a non-zero output value and this error is accumulated. Hence we hypothesized that the multiplication must be accurate if at least one of the operands is zero.

To investigate this hypothesis, we re-defined the approximate multiplier m to m', where:

$$m'(a,b) = \begin{cases} m(a,b) & \text{if } a \cdot b \neq 0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Now the original NN which employs approximate multipliers m' exhibits the classification accuracy of 94.20%. Although the impact of approximate multipliers on the accuracy is application-specific, this benchmark showed that it is necessary to have the accurate multiplication by 0. Figure 1 shows the absolute difference between outputs of the same neurons in the case that approximate multipliers provide (a) inexact and (b) exact multiplication by 0.

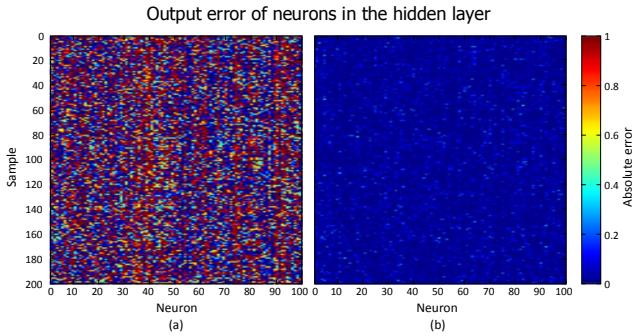

Figure 1: The error of the output neurons in the approximate NN in comparison with the original NN. The approximate NN utilizes approximate multipliers showing a 5.2% error and (a) inaccurate and (b) accurate multiplication by zero

3. PROPOSED DESIGN METHOD

The proposed method is based on uniform NNs that utilize approximate multipliers. In this section, we will define feasible approximate multipliers, describe a design space exploration search method for obtaining the feasible multipliers, and introduce the overall methodology for NN approximation.

3.1 Constraints and cost function

A digital combinational circuit with n inputs and m outputs computes a completely-specified Boolean function $\mathcal{F} : B^n \to B^m$, $B = \{0,1\}$, that maps n-input Boolean vector $x = \langle x_1, x_2, \dots x_n \rangle$ to an m-output Boolean vector $y = \langle y_1, y_2, \dots y_m \rangle$ with associated hardware cost. Let n-bit accurate multiplier be represented by a function $\mathcal{M} : B^n \times B^n \to B^{n+n}$ and let $\delta : B^m \to \mathbb{N}$ assign a natural number to an m-bit Boolean vector.

The error metric is defined as maximal relative error ε, i.e. it is requested that the maximal arithmetic error of multiplication for each combination of operands is lower than ε on the whole output range (which is $0 \dots (2^{2n}-1)$). This error ε will be one of the input parameters of the algorithm designing approximate multiplies.

A candidate approximate multiplier $\mathcal{M}'$ is a *feasible* solution is two conditions hold. (i) The error is acceptable:

$$\forall_{(a,b) \in B^n \times B^n} : |\delta(\mathcal{M}(a,b)) - \delta(\mathcal{M}'(a,b))| \leq \varepsilon \cdot (2^{2n}-1). \quad (2)$$

and (ii) multiplication by 0 is accurate:

$$\forall_{a \in B^n} : \quad \mathcal{M}(a,\{0\}^n) = \mathcal{M}'(a,\{0\}^n) \quad \wedge \quad \mathcal{M}(\{0\}^n,a) = \mathcal{M}'(\{0\}^n,a). \quad (3)$$

In the approximation process, the implementation cost of multiplier $S_{\mathcal{M}'}$ will be estimated as the number of used gates. The number of two-input gates is a sufficient metric because the circuits are relatively simple (as it will be seen in Section 5). The number of used gates $S_{\mathcal{M}'}$ is determined recursively as follows: (1) the gate is used if its output is connected to output of the circuit; (2) the gate is connected if its output is connected to an input of any used gate.

The cost function for the approximation process is defined as

$$\mathbf{C}_{\mathcal{M}'} = \begin{cases} S_{\mathcal{M}'} & \text{if constraints (2) and (3) are met} \\ \infty & \text{otherwise} \end{cases}. \quad (4)$$

3.2 Approximate multiplier design

In order to approximate an accurate multiplier, various approaches have been proposed. In this work, we employ Cartesian Genetic Programming (CGP) [12] because it can easily handle constraints given on candidate circuits, the method is naturally multi-objective and high-quality approximate circuits have already been obtained with it [18].

The standard CGP is a branch of genetic programming which represents candidate designs using directed acyclic graphs [12]. A candidate circuit is modeled using a 2D array of programmable nodes with n_c columns and n_r rows. In our case, the nodes will be 2-input Boolean functions, where Γ is the set of available functions. The circuit utilizes n_i primary

[1]https://github.com/rasmusbergpalm/DeepLearnToolbox

inputs and n_o primary outputs. Feedback connections are not enabled.

The primary inputs and the outputs of the nodes are labeled $0, 1 \ldots n_c \cdot n_r + n_i - 1$ and considered as addresses which the node inputs can be connected to. A candidate solution is represented in the so-called chromosome (which is, in fact, a netlist) by $n_r \cdot n_c$ triplets (x_1, x_2, ψ) determining for each node its function ψ ($\psi \in \Gamma$) and input connections. The last part of the chromosome contains n_o integers specifying the nodes where the primary outputs are connected to. While the chromosome size s is constant $s = n_c n_r (n_a + 1) + n_o$, the circuit size is variable and measured as the number of active (i.e. used) nodes. See an example in Fig. 2. The set of valid chromosomes (netlists) represents the whole search space.

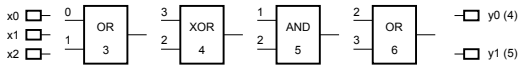

Figure 2: Example of a circuit in CGP with parameters: $n_i = 3$, $n_o = 2$, $n_c = 4$, $n_r = 1$, $\Gamma = \{0^{and}, 1^{or}, 2^{xor}\}$. Chromosome: 0, 1, 1; 3, 2, 2; 1, 2, 0; 2, 3, 1; 4, 5. Gate 6 is not used. Logic behavior of the circuit is:

$$y_0 = ((x_0 \text{ or } x_1) \text{ xor } x_2);\ y_1 = x_1 \text{ and } x_2.$$

CGP employs a simple search method . In our case, the initial population P of CGP contains one of various implementations of accurate multipliers and a few circuits generated using mutation of the accurate multiplier. Creating the accurate multiplier in the initial population is trivial as there is a one-to-one mapping between multiplier netlists and CGP chromosomes. The next step consists in the evaluation of candidate circuits using the fitness function. Each member of P then receives the so-called fitness score and the highest-scored individual becomes a new parent of the next population. From this parent, λ candidate solutions are generated using mutation. The termination criterion is given by the number of iterations.

Despite many attempts to propose a suitable crossover operator to CGP, the mutation is still used as the crucial genetic operator. The mutation operator modifies up to h randomly chosen genes (integers) of the chromosome. Their new values are generated randomly, but it is checked whether the new values are valid. One mutation can affect either the gate function, gate input connection, or primary output connection.

In order to approximate multipliers, the fitness is defined as $fitness(\mathcal{M}') = -\mathbb{C}_{\mathcal{M}'}$ and Γ = {NAND, NOR, XNOR, AND,OR, XOR, NOT, identity}.

3.3 Evaluation platform

This section describes the evaluation platform used for simulations of the proposed approximate NNs. The software framework is based on C++ project *tiny-cnn*[2]. We have implemented two new types of layers to NNs: the approximate fully connected layer and approximate convolution layer. In the software simulation, the approximate multiplication was realized using a lookup table. The framework uses weights and inputs with double floating point precision. We rounded them to the fixed point representation in the range $\langle -1, 1 \rangle$. All numbers are unsigned, the sign is determined after the computation.

[2]https://github.com/nyanp/tiny-cnn

We have also synthetised multipliers for neural network. The multipliers were implemented at the Register-Transfer Level (RTL) in Verilog and mapped to the IBM 45nm technology using Synopsys Design Compiler Ultra. The hardware multiplication unit utilizes a combinational approximate unsigned multiplier circuit and logic for the sign extension. We have utilized the one's complement method which is easy to calculate ($4n$ XOR gates), but provides lower accuracy w.r.t. the two's complement method (extra three one-subtractors) used in standard applications. The framework can estimate energy consumption and area under iso-speed conditions. The clock frequency for 8 bit neurons is 3 GHz and 2.5 GHz for 12 bit neurons.

There are equal count of multiplications and additions and one activation function in the neuron computational model. Since the count of operations is big (tens or hundreds) and the multiplication consumes significantly more energy than addition, the multiplication is the most consuming part and power reduction of this part significantly contributes to the overall power consumption reduction.

3.4 Overall design methodology

Finally, the overall methodology for design of approximate multipliers that will be used in approximate NNs is presented in Figure 3. The inputs to the methodology are the accurate neural network (with accuracy J), training and testing data, quality constraint Q, accurate multiplier and initial error ε. The whole procedure is as follows. The CGP algorithm is utilized for creating a set of approximate multipliers from the accurate one. The approximate multipliers are used in the pretrained network. In order to achieve the best quality results, the network is retrained. The NN implementation showing the best accuracy K is selected. The accuracy K is checked if it meets the quality constraint Q w.r.t. accurate neural network with accuracy J. If the constraint is not met, the relative maximal approximation error ε is decreased and next iteration is performed. Due to non-deterministic generation of approximate multipliers by CGP, it is necessary to generate several approximate multipliers and then re-evaluate the accuracy of NN.

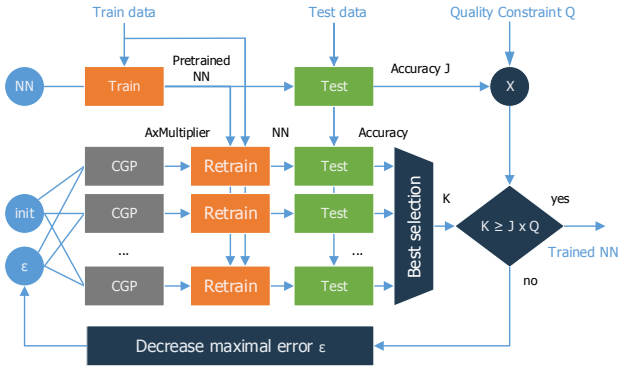

Figure 3: Overview of approximate multiplier design for approximate NNs

4. EXPERIMENTAL SETUP

The goal of the experiments is to investigate the impact of proposed approximation methods on the accuracy and power

consumption of NNs. This section provides the experimental setup and benchmark problems description.

4.1 CGP configuration

CGP will be used to design 7 bit and 11 bit unsigned multipliers. The sign extension, i.e. 8 bit and 12 bit-width multipliers, will be designed manually using the one's complement method. The maximum target arithmetic error ε of approximate multipliers is taken from the set {0.5%, 1%, 2%, 5%, 10%, 15%, 20%}. We did not employ arithmetic error beyond 20% for approximate multipliers since the classification accuracy drops significantly. The approximation process starts with accurate multipliers (Ripple Carry Array, Carry Save Array with RCA and CSA adders, Wallace Tree with RCA, CSA and CLA adders [23]) which constitute the so-called initial population. Considering two bit widths, 7 target errors and 6 types of initial multiplier architectures, there are 84 initial configurations in total.

4.2 NN Accuracy analysis

The accurate multipliers are replaced with candidate approximate multipliers in the NN which is then retrained in the supervised learning scenario. Two types of NN and two classification datasets are utilized for the accuracy analysis.

4.2.1 Handwritten numbers

The first dataset is MNIST (Mixed National Institute of Standards and Technology) database of handwritten numbers [10] which consists of two sets of data. The first one is the training data set containing 60,000 28×28 images and their labels. The second one contains 10,000 test pairs. The digits are normalized and centered in fixed-sized images. The dataset is very popular for quantifying the accuracy of classification methods. It was shown that neural networks are able to provide the error rate as low as 0.27% using convolutional networks [3]. In this case, we used a MLP network with 28×28 input neurons, 300 neurons in the hidden layer and 10 output neurons whose outputs are interpreted as the probability of each of 10 target classes $(0-9)$.

4.2.2 House numbers

The second dataset is SVHN (Street View House Numbers) which is obtained from house numbers in Google Street View images [15]. The images come from a significantly harder, unsolved, real-world environment. The dataset contains 73,257 digits for training and 26,032 digits for testing. Each digit is represented as a pair of 32×32 RGB image and label. While MLP does not provide good accuracy in this case, LeNet-6 (a 6 layer NN in Figure 4 [9]) is able to classify the images with a very small error. The network consumes a 32×32 grayscale image as an input. In order to reduce the complexity, we transformed original RGB images to grayscale using an equation $Y = 0.299R + 0.587G + 0.114B$.

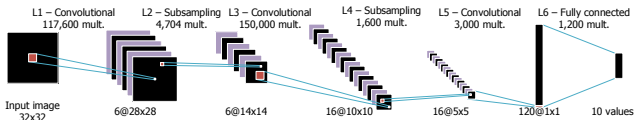

Figure 4: LeNet structure, where L1, L3, L5 and L6 contain approximate multipliers, i.e. 98 % of multiplications are approximated.

Layers L1, L3 and L5 perform the convolution. The L3 employs a special table that indicates which feature map from 6 previous maps is used for generating each of 16 output feature maps. Last layer (L6) connects all 120 values with each neuron of the output layer. Convolutional and fully connected layers represent 98 % of all multiplications performed in the network. Hence, the approximation was applied only for this layers. Layers L2 and L4 perform a subsampling by weighted average, but this process was not approximated because it has a small impact on power consumption.

5. RESULTS

The first part of this section is devoted to the results of the proposed CGP-based approximation of multipliers. The second part deals with approximate NNs. We also report detailed parameters of approximate multipliers.

5.1 Multiplier approximation with CGP

CGP is used with settings given in Section 3.2. Five circuits ($\lambda = 5$) are evaluated in each iteration and new circuits are created by modifying just 1 integer in the chromosome ($h = 1$) of the parent circuit. CGP operates with $n_c \times n_r = 900$ and $n_c \times n_r = 300$ (respectively) nodes for 11-bit and 7-bit multiplier (respectively). The evaluation of a candidate 11 bit approximate multiplier requires evaluation of 256x more test vectors than for the 7 bit multiplier (2^{22} vs. 2^{14}). Hence, the maximal time for CGP was set to 120 minutes for 11 bit multipliers and 30 minutes for 7 bit multipliers. CGP performed 1,343 (and 122,773) iterations on average for 11 (and 7) bit multiplier. The setting of CGP corresponds with typical values used in the literature [12, 18].

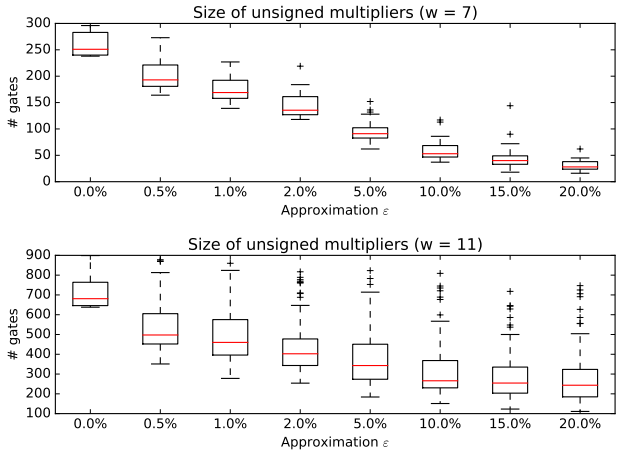

Figure 5: The number of gates in approximate multipliers

Figure 5 gives the number of gates in approximate multipliers as boxplots showing the results from 60 independent runs for a given error ε. If the error is zero only 6 values are presented which corresponds with gate counts in our accurate multipliers. In addition to obtaining many different tradeoffs between the error and the number of gates, the proposed method guarantees the exact multiplication by zero in all approximate multipliers. The spread in obtained gate counts is high especially for the 11-bit multipliers. Please

note that the approximate multipliers do not prolong delay of the original accurate multipliers.

5.2 Approximate NNs

For constructing the approximate NNs, each of designed approximate multipliers was utilized. In total, we thus obtained 2×852 NNs (LeNet6 and MLP with (28×28)-100-10 layers) using 852 approximate multipliers which were subsequently extended to signed versions using the one's complement. The accuracy of approximate NNs is presented in Figure 6 for a pretrained network (column *initial*) and then for 5 and 10 retrains, respectively, using the backpropagation algorithm. Each boxplot represents 60 multipliers, e.g. in MNIST $w = 8, \varepsilon = 15\%$, there is one multiplier leading to the accuracy 20% and another to 97% in the initial placement in pretrained neural network.

Because it is infeasible to estimate power consumption of each of 852 circuits, we did a precise power analysis in the following way. We have selected circuits for each error ε and bit-width w that have one of top three best accuracies for SVHN. Then we selected a circuits from the top three sets that provide the best tradeoff between MNIST accuracy and the number of used gates. The accuracy of NNs utilizing the selected approximate multipliers is shown in Figure 7. The accuracy is normalized w.r.t. a circuit with the same bit-width and $\varepsilon = 0\%$. We have followed the alphabet-reduced

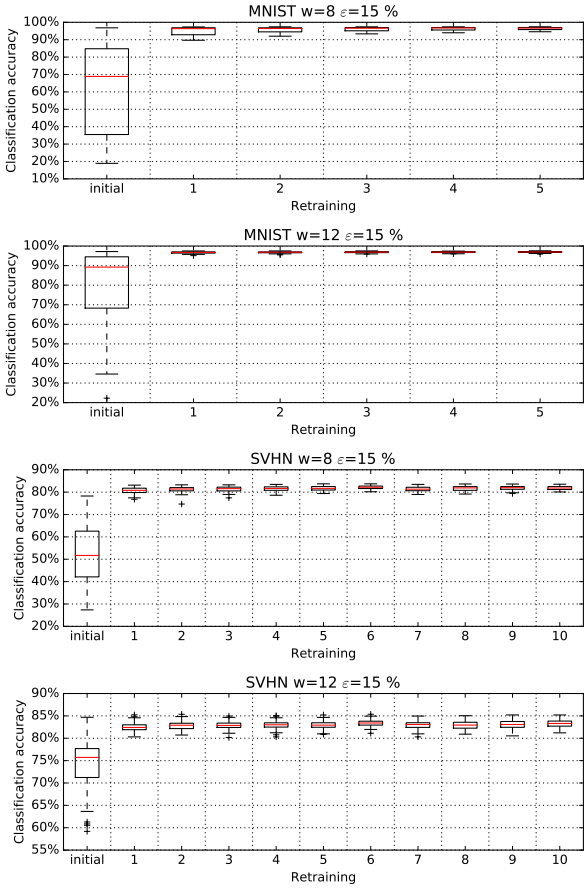

Figure 6: Accuracy of NNs for several configurations during the retraining process. The data shows statistical information for all designed multipliers with selected w and ε.

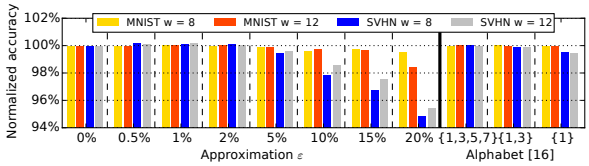

Figure 7: Normalized accuracy of NNs utilizing the best approximate multipliers developed by the proposed method for a given ε and its comparison with [16]. For each configuration, the accuracy is normalized w.r.t. NN employing accurate multipliers ($\varepsilon = 0$).

approach proposed in [16] and perform the simulation. The reduced alphabet $\{1\}$ enables to employ just 40 out of 256 weights for $w = 8$ and 200 out of $4,096$ weights for $w = 12$. It can be seen that results from [16] are very similar for the proposed approach when $\varepsilon = 5\%$.

Error ε	**Power** μW	**Area** μm	**Accuracy SVHN**	**Accuracy MNIST**
0 %	250.0	440.0	87.00	97.67
0.5 %	201.0	367.7	87.15	97.66
1 %	175.0	316.6	87.08	97.68
2 %	107.0	218.3	87.07	97.65
5 %	58.6	129.9	86.54	97.58
10 %	45.2	109.5	85.11	97.31
15 %	22.3	63.2	84.20	97.42
20 %	22.9	65.8	82.52	97.22

(a)

Error ε	**Power** μW	**Area** μm	**Accuracy SVHN**	**Accuracy MNIST**
0 %	831.0	1175.0	87.04	97.70
0.5 %	417.0	664.9	87.15	97.69
1 %	475.0	720.8	87.22	97.71
2 %	284.0	523.8	87.06	97.71
5 %	247.0	483.0	86.68	97.61
10 %	125.0	285.0	85.81	97.48
15 %	115.0	262.4	84.95	97.38
20 %	111.0	252.5	83.06	96.18

(b)

Table 1: Power consumption and area of (a) 8-bit and (b) 12-bit sign-extended approximate multipliers and the absolute accuracy of NNs utilizing these multipliers.

Table 1 gives power consumption of selected approximate multipliers (in IBM 45nm process) and the accuracy of NNs that are utilizing these multipliers in two classification tasks. In comparison with the original NNs (which utilize the accurate multiplication), one can observe that approximate NN ($w = 8, \varepsilon = 10\%$) provides 81.9% power reduction of multiplication process while its accuracy decreases by 1.89% for SVHN and 0.36% for MNIST. If the error of multiplication remains below 20% the accuracy is only slightly decreased (in some cases it is even improved, but the improvement is negligible) for the MNIST problem. The NN trained for the SVHN dataset is more sensitive to approximations. The reason is that SVHN is a significantly harder classification problem than MNIST, because SVHN contains natural scene images with a high variability. However, the accuracy degradation of NN is around 1% if $\varepsilon \leq 5\%$. And finally, for example, 91% multiplier power reduction ($w = 8, \varepsilon = 15\%$) corresponds with the accuracy degradation of NN less than 2.80%.

To summarise the results, it was shown in Section 3.3 that the multiplication has a significant impact on total power consumption of calculation. When the calculation of LeNet consumes approximately 44% [7], 91% reduction of multiplication power leads to a significant total power consumption reduction of the NN.

6. CONCLUSION

This paper provided a methodology for the design of power-efficient NNs with approximate multipliers. An analysis of error resiliency of neural networks showed the feasibility of using the proposed multipliers to achieve trade-off between classification accuracy versus energy consumption. By means of CGP, approximate multipliers were designed to achieve the desired tradeoffs between the accuracy and implementation cost. Resulting approximate NNs, containing the approximate multipliers, were evaluated using standard benchmarks (MNIST dataset) and a real-world classification problem of Street-View House Numbers (SVHN). A significant improvement in power efficiency was obtained compared to the exact (or original) NNs. In some cases, 91% power reduction of multiplication was obtained with classification accuracy degradation less than 2.80% for SVHN dataset.

7. ACKNOWLEDGMENTS

This work was supported by the Czech science foundation project 14-04197S and by The Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602. Syed Shakib Sarwar and Kaushik Roy's research were funded in part by National Science Foundation.

8. REFERENCES

[1] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected CRFs. *CoRR*, abs/1412.7062, 2014.

[2] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *DAC '13*, pages 113:1–113:9, 2013.

[3] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Convolutional neural network committees for handwritten character classification. In *ICDAR*, 2011.

[4] K.-L. Du and M. Swamy. *Neural Networks in a Softcomputing Framework*. Springer London, 2006.

[5] Z. Du, K. Palem, A. Lingamneni, O. Temam, Y. Chen, and C. Wu. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In *ASP-DAC '14*, 2014.

[6] G. Hinton, L. Deng, D. Yu, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov 2012.

[7] P. Judd, J. Albericio, T. H. Hetherington, T. M. Aamodt, N. D. E. Jerger, R. Urtasun, and A. Moshovos. Reduced-precision strategies for bounded memory in deep neural nets. In *HiPEAC WAPCO '16*, 2016.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *the IEEE*, 86(11):2278–2324, Nov 1998.

[10] Y. LeCun, C. Cortes, and C. J. Burges. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[11] A. Lingamneni, A. Basu, C. Enz, K. V. Palem, and C. Piguet. Improving energy gains of inexact dsp hardware through reciprocative error compensation. In *DAC '13*, 2013.

[12] J. F. Miller. *Cartesian Genetic Programming*. Springer-Verlag, 2011.

[13] S. Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4), Mar. 2016.

[14] K. Nepal, Y. Li, R. I. Bahar, and S. Reda. Abacus: A technique for automated behavioral synthesis of approximate computing circuits. In *DATE '14*, 2014.

[15] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop 2011*, 2011.

[16] S. S. Sarwar, S. Venkataramani, A. Raghunathan, and K. Roy. Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing. In *DATE '16*, pages 145–150, 2016.

[17] G. Srinivasan, P. Wijesinghe, S. S. Sarwar, A. Jaiswal, and K. Roy. Significance driven hybrid 8T-6T SRAM for energy-efficient synaptic storage in artificial neural networks. In *DATE '16*, pages 151–156, 2016.

[18] Z. Vasicek and L. Sekanina. Evolutionary approach to approximate digital circuits design. *IEEE Tr. on Evolutionary Computation*, 19(3):432–444, 2015.

[19] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan. Axnn: Energy-efficient neuromorphic systems using approximate computing. In *ISLPED '15*, 2014.

[20] S. Venkataramani, K. Roy, and A. Raghunathan. Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits. In *DATE'13*, 2013.

[21] S. Venkataramani, A. Sabne, V. J. Kozhikkottu, K. Roy, and A. Raghunathan. Salsa: systematic logic synthesis of approximate circuits. In *DAC '12*, pages 796–801.

[22] S.-C. Wang. *Interdisciplinary Computing in Java Programming*, chapter Artificial Neural Network, pages 81–100. Springer US, Boston, MA, 2003.

[23] N. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, 4th edition, 2010.

[24] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu. Approxann: An approximate computing framework for artificial neural network. In *DATE '15*, 2015.