APPROXIMATE FULLY CONNECTED NEURAL NETWORK GENERATION

Tuba Ayhan, Mustafa Altun

Istanbul Technical University, Faculty of Electrical and Electronics Engineering,
Electronics and Communication Engineering Department, 34469-Istanbul, Turkey
{tuba.ayhan, altunmus}@itu.edu.tr

ABSTRACT

Approximate computing is exploited in implementation of fully connected networks for classification problems. A multiplier structure whose area is scalable over accuracy through approximate computing is proposed. In order to employ the multipliers in a network, an area reduction algorithm is formed. It can adjust the approximation level of multipliers while still maintaining the target classification performance, without prior information on the value of network weights. Implementing on a Spartan6 FPGA, up to 79% area saving is recorded for various performance targets.

Index Terms— Approximate computing, fully connected network, area reduction

I. INTRODUCTION

In the last decade machine learning and artificial intelligence era had a great burst with the increasing use of SIMD processors and cloud computing. Thanks to these powerful computation platforms, high performance is achieved in applications such as object detection, segmentation, and image recognition, where computational overhead is due to the requirement of large-amount data processing [1]. Relatively expensive tools such as GPU or remote solutions like cloud computing are used for artificial intelligence applications. Area and power aware hardware designs should be presented in order to benefit from artificial intelligence tools in low-resource devices such as mobile devices and sensor nodes with limited computation power. Therefore, a small, fast, and power efficient hardware implementation of Convolutional Neural Networks (ConvNets) is a timely research topic. Power and area reduction can be preformed by accelerators [2], [3], [4], or approximate computing methods can be used to trade off system accuracy for area or power. For example in [5], error critical neurons are detected during training. Similarly, in [6] an optimization procedure for approximate network implementation to reduce energy consumption under given quality constraint is proposed. These approaches tie training phase and implementation phase of a network together. Considering that Caffe like learning frameworks are preferred to train large networks, they are not useful for who wants to implement and test their network on FGPA in as quick and as easy as possible.

978-1-5386-5153-7/18/$31.00 © 2018 IEEE

To isolate training and implementation platforms from each other, a framework which estimates the error resiliency of layers rather than individual neurons is proposed. Given the classification error tolerance of the application, the tolerable approximation level of layers is estimated. Moreover, an approximate Fully Connected Network (FCN) whose area can be traded-off for accuracy is implemented. The layer-based implementation matches with the area saving method which runs a nonlinear programming solver with constraints. Implementation and area reduction algorithm are combined under an easy-to-use framework.

Framework at a glance is illustrated in Figure 1 with two main parts: an area reduction block, and a network generation block. Verilog HDL code for FCN implementation is generated by the network generation block. The block uses the predefined approximate multipliers, and network architecture template as well as the user defined weights and bias. The network architecture and approximate layers are explained in Section II. The area reduction algorithm receives basic FCN information, such as the number and size of layers, and error tolerance set by the user. This error is propagated through the layers to obtain an approximation model as explained in Section III. The framework is tested under different conditions and reported in Section IV. Finally the paper is concluded in Section V.

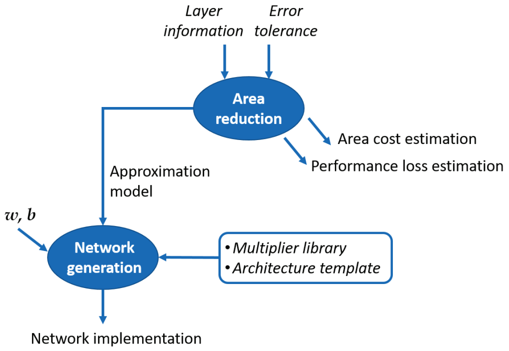

Fig. 1. Proposed framework contains two main parts: an area reduction algorithm, and a network generation code.

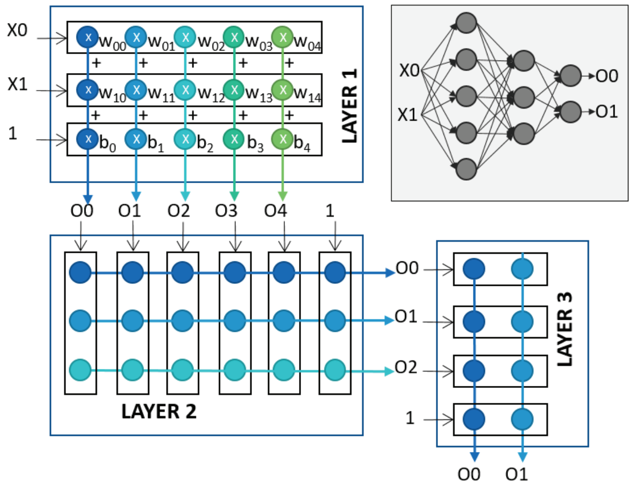
X0
X1
1
w_{00} w_{01} w_{02} w_{03} w_{04}
w_{10} w_{11} w_{12} w_{13} w_{14}
b_0 b_1 b_2 b_3 b_4
LAYER 1
O0 O1 O2 O3 O4 1
LAYER 2
O0
O1
O2
1
LAYER 3
O0 O1
X0
X1
O0
O1

Fig. 2. Simplified network architecture.

II. NETWORK ARCHITECTURE

The network structure is explained on an example given in Fig. 2. In this example, the network has three layers with 5, 3 and 2 neurons from input layer to output layer, as given in the grey box. Layers employ constant multipliers followed by accumulators. Multiplication scheme is illustrated for layer 1. Layer 1 has two inputs, moreover the bias is interpreted as a constant 1 input. Then, layer 1 process 3 inputs in total. Joining bias into data input is followed in the other layers, as well. Each input of the layer 1 is multiplied with 5 weights, since the layer has 5 neurons. Similarly, constant 1 is multiplied by 5 bias of the neurons. The multipliers are combined under Constant Multiplier Blocks (CMBs), each having 5 outputs. Further information on CMBs is in Section II-A. Related outputs are accumulated to result the layer outputs. For example, the first arrow on the left of the figure is an adder calculating O_0 of layer 1:

$$O_0 = w_{01}x_0 + w_{10}x_1 + b_0. \tag{1}$$

Large amount of registers to store weights and biases are avoided by using constant multipliers. On the other hand, register blocks can be inserted between the consecutive layers in order to provide pipeline. The control circuity that stages reading input, pipelined layers and printing output is not shown in the figure, to keep it concise on this paper's scope. Moreover, this structure do not show any activation function such as ReLu or sigmoid, because the activation function is not relevant with the area reduction technique analysed in this work.

II-A. Constant multiplier block

In Fig. 3, CMB for layer 1 is shown. This block calculates 5 outputs: 5 multiplications with input x_i. The output j is $w_{ij}x_i$, which is approximated by sum of N shifted versions of the input x_i:

$$O_j = \sum_{n=1}^{K} a_n x_i, \tag{2}$$

where a_n is either 0 or power of 2. Multiplication with power of 2 means calculating the shifted version of the input. Shift operation is implemented by reorganizing the input bits. For an 8-bit long input 8 shifted versions are generated, as shown in the example. The output j is result of an adder with K addends. This structure allow us to generate CMBs with different approximation levels: accuracy of multiplication increases with the number addends.

The shifted versions which are not used as an addend will be trimmed by the circuit compiler, in a later stage. Number of registers in CMB does not depend on the accuracy score of the block but, increasing the number of addends increase the accuracy as well as the combinational circuit area. In this work, networks are implemented on a Spartan6 FPGA, so area of a circuit is measured by number of LUTs. The error of an approximate multiplier is reported as the standard deviation on error over all possible multiplications. For signed computations, mean error for shift-add approximate multiplication will be 0. In the example, an exact multiplication is performed with 8 addends, consuming 120 LUTs on FPGA. We can reduce the cost of combinational circuitry by decreasing the number of addends down to 2, as illustrated in Fig. 4.

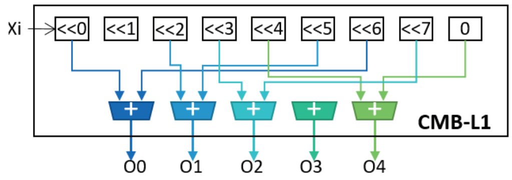

Fig. 3. Constant multiplier block with two addends.

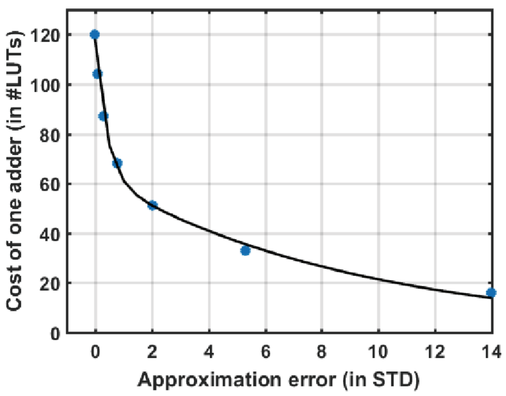

Fig. 4. Error of multiplication is inversly proportional to the area of combinational adder in the proposed CMB.

III. AREA REDUCTION

Thanks to the CMB structure, area reduction is possible in exchange with accuracy loss. In this section, area vs accuracy trade-off for FCNs is analysed in order to reduce area consumption when an ultimate error tolerance is set by the user. The ultimate error tolerance, e_T is defined as an acceptable error margin on classification performance. Therefore, unit for e_T is percent, i.e. if classification performance of an FCN is already 95%, but 90% classification performance is acceptable for some sub-application, then that network has an ultimate error tolerance $e_T = 5\%$.

In Section III-A, e_T is propagated through the FCN in order to calculate the tolerable computation error of network layers. e_T can be distributed over layers in many different ways, so a distribution to reduce the total network size is to be found. A method to select the smallest CMB combination while maintaining e_T is offered in Section III-B.

III-A. Error propagation through layers

According to the layer structure given in Section II, each layer is composed of only one type of neuron, in other words accuracy of layers are in concern rather than accuracy of individual neurons. Assume an FCN with L layers has N_i neurons in layer i and each layer has an input size of M_i. Layer i is implemented with same type of CMBs: M_i CMBs are required for layer i. As explained with Figure 4, we can construct as many CBMs with different approximation levels as the input word length of the layer. The approximation level of a CBM is quantified with the standard deviation on error and shown with A_i for layer i. Therefore, the ultimate error tolerance e_T in percent has to be converted into approximation error in std. To demonstrate this conversion, binary classification is considered, because binary classification can then be generalized for multiple class problems.

Binary classification uses a threshold to interpret the FCN output. If the FCN output is close to the threshold, computation error may cause classification error. Ideally, FCN output shows a distribution away from the threshold. If the deviation on the output distribution increase, the chance of crossing the threshold increases. Therefore, e_T is associated with the standard deviation of the output neurons. Moreover, this distribution depends on the noise injected to the system, by approximate CMBs. Let's call the standard deviation of noise infused at layer i as H_i. Then, H_L, the error injection at the output layer cannot override e_T:

$$H_L \leq \| \sqrt{e_T} \| \tag{3}$$

Layer receive erroneous inputs from the preceding approximate layers. However, if the mean of approximation error is 0, as in our approximate CMBs, then noise is moderated as the number of neurons increase:

$$H_i \cong A_i + \frac{H_{i-1}}{\sqrt{2}N_{i-1}}. \tag{4}$$

As (4) shows, the error contribution of layer i is related with both the approximate CMBs, and the noise inherited from the previous layer.

III-B. Area reduction algorithm

The goal of this algorithm is to minimize the area of combinational computation circuits while still maintaining the target performance in terms of classification rate. It should be noted that the area reported by this algorithm is not a global minimum: smaller network may be possible with different weights and architecture. However, the motivating applications of this work require separating FCN training from circuit implementation.

An optimization problem with linear constraints is constructed and solved by a built-in nonlinear programming solver of Matlab (*fmincon*). The optimizer finds the best combination of CMBs. Therefore, the result returned by the solver is a vector x with approximation levels of a CBMs in all layers from 1 to L. This vector is denoted as $x = [A_1\ A_2\ ....\ A_L]$. The objective function to minimize the total combinational computation area of the network:

$$f = \sum_{i=1}^{L} N_i C_i, \tag{5}$$

where C_i is the combinational area of the CMB adder used in layer i. C_i is a function of A_i, the cost of a CMB changes with computation accuracy as in Fig. 4. An exponential function in the form of $C_i = a\exp(bA_i) + c\exp(dA_i)$ is fitted on the curve, resulting in the following parameters: $a = 54.93\ b = -2.505\ c = 63.13\ d = -0.1082$.

The constraints are derived by minimum and maximum possible adder area. Lower bound is set to a 0 vector of size $(1, L)$ for exact CMB adder. Upper bound depends on the word length, maximum error is 14 when input word length is 8-bit for this work. The last linear constraint is derived by using the error propagation conditions (3) and (4). For a three layer network, error contribution of the output layer is

$$H_L = H_3 = A_3 + \left(A_2 + \frac{A_1}{\sqrt{2}N_1}\right)\frac{1}{\sqrt{2}N_2}, \tag{6}$$

which has to ensure (3). Then, an inequality constraint

$$\left[\frac{1}{2N_1N_2}\ \frac{1}{\sqrt{2}N_2}\ 1\right] x^T \leq \| \sqrt{e_T} \| \tag{7}$$

is found.

IV. SIMULATION RESULTS

Experimental set up which is simplified on Fig. 5 is constructed with three layers. The input data precision is limited to 8 bits. Implementation is performed on Xilinx Spartan6 FPGA with ISE 14.7 design tools. LUT counts of these networks are acquired from place & route reports.

Networks with different sizes are considered for test: neuron counts of the networks vary between 12 and 120. The

Network	Exact	$e_T = 0.5$	$e_T = 5$	$e_T = 10$
120 Neurons	1826	1521	513	380
96 Neurons	1463	1431	529	328
72 Neurons	1026	927	497	286
60 Neurons	745	617	314	216
36 Neurons	614	292	165	163
12 Neurons	252	252	106	104

Table I. Area utilization of three network architectures with three tolerance settings.

optimizer is run for 3 ultimate error tolerance values: $e_T = \{0.5,\ 5\ 10\}$. For each network architecture and problem, 3 approximate and 1 exact implementations are possible. Weights of networks depend on the classification problem (5 linearly separable and 5 linearly non-separable problems), so 10 weight sets has to be used for each approximate and exact implementation. That makes in total 40 implementations per network architecture. In Table I, each cell shows the average LUT count of the related 10 implementations. The generated binary classification problems can be solved with 0 % to 2,4 % classification rate with exact networks. It should be noted that classification error rates of the implementations are calculated to be lower than the tolerated values (e_T). LUT counts show a decreasing pattern as the ultimate error tolerance e_T increase. An area saving up to 52.44%, 73.13%, and 79.19% is obtained for 0.5%, 5%, and 10% error tolerance, respectively. Exact implementation area is significantly larger than any approximate one, except for the smallest network architecture. Benefits of approximate computation for area saving can be quantified more easily as the network layers get larger. Exact networks and networks with 2-addends are investigated in Figure 6. Increasing the number of neurons does not necessarily increase the LUT count, because the weights are carved in a way that require less addends, for some larger networks. Regardless of this deduction, approximate version of a network consumes less LUT than the exact network.

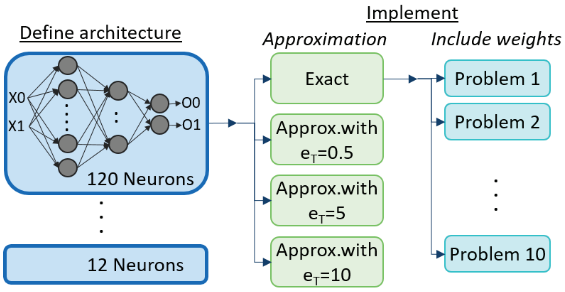

Fig. 5. The experiments are carried out as follows: first a network architecture is defined with 12∽120 neurons, then each architecture is implemented 40 times, in total.

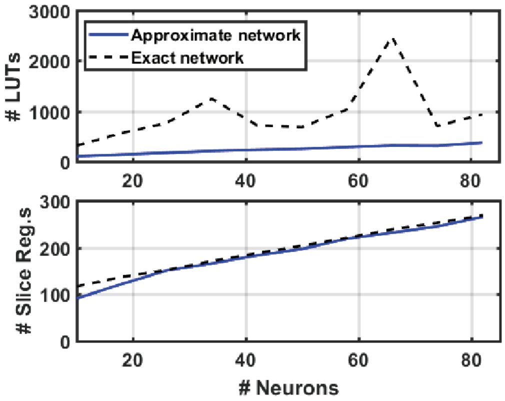

Fig. 6. Area reduction with 2-addend CMBs is more evident as the layer sizes insrease.

V. CONCLUSION

In this work, an approximate FCN generation framework, which includes an FCN structure and an area reduction algorithm utilizing the proposed structure is presented. By the proposed FCN structure employing CMBs, the approximation error of the multipliers could be gradually increased. Moreover, best approximation levels are determined to reduce the area consumption of the network, yet the network can provide targeted classification performance. The area optimization can be completely detached from FCN training. Thus, without prior information on network constants, the framework can still achieve up to 73% area saving with less than 5% loss in classification accuracy.

ACKNOWLEDGEMENTS

This project is funded by TUBITAK (The Scientific and Technological Research Council of Turkey), with the grant number 117E078.

VI. REFERENCES

[1] R. Ranjan, S. Sankaranarayanan, A. Bansal, N. Bodla, J. C. Chen, V. M. Patel, C. D. Castillo, and R. Chellappa, "Deep learning for understanding faces: Machines may be just as good, or better, than humans," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 66–83, Jan 2018.

[2] J. H. Kim, B. Grady, R. Lian, J. Brothers, and J. H. Anderson, "Fpga-based cnn inference accelerator synthesized from multi-threaded c software," in *2017 30th IEEE International System-on-Chip Conference (SOCC)*, Sept 2017, pp. 268–273.

[3] Y. H. Chen, T. Krishna, J. Emer, and V. Sze, "14.5 eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, Jan 2016, pp. 262–263.

[4] B. Moons and M. Verhelst, "A 0.3-2.6 tops/w precision-scalable processor for real-time large-scale convnets," in *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, June 2016, pp. 1–2.

[5] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2016, pp. 1–7.

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