Designing Energy Efficient Approximate Multipliers for Neural Acceleration

Sayandip De, Jos Huisken and Henk Corporaal
Department of Electrical Engineering, Eindhoven University of Technology
Postbus 513, 5600 MB Eindhoven, The Netherlands
Email: {sayandip.de, j.a.huisken, h.corporaal}@tue.nl

Abstract—**Many error resilient applications can be approximated using multi-layer perceptrons (MLPs) with insignificant degradation in output quality. Faster and energy efficient execution of such an application is achieved using a neural accelerator (NA). This work exploits the error resilience characteristics of a MLP by approximating the accelerator itself. An error resilience analysis of the MLP is performed to obtain key constraints which are used for designing energy efficient approximate multipliers. A systematic methodology for the design of approximate multipliers is used. A graph based netlist modification approach is considered. Approximate versions of basic standard cells are generated and these are used to replace accurate cells in the synthesized netlist in a systematic quality controlled manner. These approximate multipliers are further used for approximating the multiply and accumulate (MAC) units in the neural accelerator (NA). The results are validated by considering approximate neural replication of a robotic application, *inversek2j*. System level energy savings of upto 14% is obtained for less than 7% degradation in output quality. Average application speedup of 24% is obtained over accurate neural accelerator (NA). The results are compared with state-of-the-art approximate multipliers and a comparison with truncation (bit-wise scaling) is performed. Moreover, error healing capability of MLPs is shown by studying the impact of retraining on networks with approximate multipliers.**

Keywords— **Approximate Computing, Machine Learning, Neural Networks, Low Power Design.**

I. Introduction

Over the past 50 years, the world has experienced a computing revolution. This is evident with the transition from mainframe computers that filled up a room to today's mobile computing devices which can be fitted in a pocket. Transistor technology scaling has been the major driving force behind this revolution which led to exponential performance improvements in line with Moore's law [1] predictions. Lately, diminishing returns from technology scaling due to the breaking of Dennard Scaling [2] has led to a switch from single core to multi-core designs. However, performance gains from continuous increase in the no. of cores are constrained by the level of parallelism in the applications and the exponential increase in leakage current, thus leading to the "Dark Silicon" [3] era. To keep up with the performance improvement trend, as predicted by Moore's law, in the "Dark Silicon" era, there is a need to focus on newer computing paradigms which are realistic and economically viable. *Approximate Computing* has been proposed as a potential solution to these challenges. It exploits the inherent error resilience of applications and trades-off quality of desired output for energy/performance improvements.

Research efforts on application resilience analysis [4] show that machine learning applications using artificial neural networks (ANNs) are highly error resilient. Modern computing trends focus on implementation of ANNs on low power embedded systems. Approximate computing can play a significant role in designing energy efficient ANNs by trading-off the error resiliency of learning algorithms for energy improvements. Optimizing ANNs for energy efficiency can be done at various levels, namely, neurons, interconnects, learning algorithms, memory access etc [5]. In this paper, we focus on design of well-optimized energy efficient ANNs by improving the energy efficiency of the computation performed within the neurons.

Recent studies have shown that error resilient applications can be entirely or partially replaced by neural networks for faster and energy efficient execution [6]–[8]. The neural network model used for this neural approximation is multi-layer perceptron (MLP). MLPs are first trained to mimic a compute intensive and error resilient code segment in the compilation phase. During the execution phase, this code segment is replaced with the invocation of a low power neural accelerator.

A MLP architecture is itself error-resilient and can tolerate faults when the network is retrained [9]. This motivates us to extend the neural approximation procedure from the algorithmic level to the circuit level. The neural accelerator is first approximated by replacing the multipliers within the MLP with approximate ones. The approximated MLP is then retrained to obtain energy efficient designs with little loss of quality. Rather than manually designing approximate multipliers, this work uses a systematic netlist modification approach for introducing approximations. First, gate level approximate versions of selected standard cells are generated. These approximate cells are then used to replace the accurate cells in the netlist of the multiplier. The replacement is guided by error metrics which are obtained from application specific quality constraints.

The key contributions of this work are:

- An error resilience analysis of a MLP is performed to obtain the key constraints for designing approximate multipliers (see section III).

- A systematic automated methodology for designing energy efficient multipliers is used. A comparative analysis of the proposed multipliers with state-of-the-art approximate multipliers is done (see section IV & V).
- An energy efficient approximate MLP using approximate multipliers is designed. A study of retraining on approximate MLP is also shown (see section V).
- Proposed approaches are validated for execution of an error resilient robotic application, inverse kinematics (*inversek2j*) (see section V).

The rest of the paper is organized as follows: Section II gives a background on the basic idea of neural acceleration and an overview on the existing neural accelerator (NA) architectures in the literature. This section also outlines the state-of-the-art approximate multiplier designs in the literature. Section III provides a study on error tolerance of a MLP trained for *inversek2j*. The methodology used for the design of approximate multipliers is described in section IV. Section V provides the results for our proposed techniques. Finally, section VI concludes the paper with a brief summary of our findings and a glimpse on possible future approaches.

II. Preliminaries & Related Work

A. Multilayer Perceptron (MLP)

A multilayer perceptron (MLP) is a fully connected feed-forward neural network model. It consists of an input layer, an output layer and atleast one hidden layer, shown in Fig. 1.

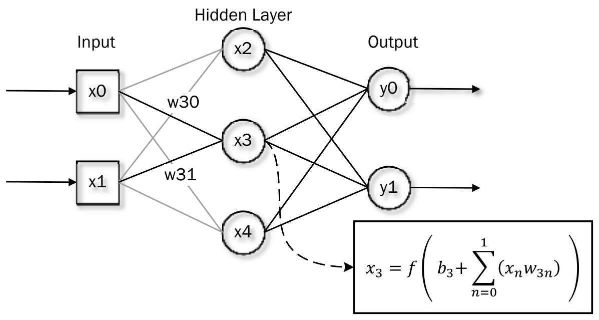


Fig. 1: A typical multilayer perceptron (MLP).

The neurons in a MLP perform weighted sum of the inputs **x** from the previous layer to produce an output **y** (see Eq. 1). The weights are denoted by **w** and the bias is denoted by **b**. The number of neuron in the previous layer is denoted by **l**. This is followed by a non-linear activation function. This paper uses a rectified linear unit (relu) as the non-linear activation function. The mathematical representation of relu is shown in equation 2.

$$y_i = f(b_i + \sum_{n=1}^{l} x_i \times w_{in})) \tag{1}$$

$$f(x) = \left\{ \begin{array}{l} x \ if \ x \geq 0 \\ 0 \ otherwise \end{array} \right. \tag{2}$$

B. Neural Acceleration and Proposed Architectures

The basic idea of neural acceleration is to mimic and replace a compute intensive error resilient kernel of an application by a neural network [6]. It consists of three main stages: annotation, compilation and execution (see Fig. 2). In the annotation phase, the source code is analysed to determine compute intensive approximable kernels with well-defined inputs and outputs. These kernels are annotated and corresponding input/output pairs are recorded to obtain the training dataset. A MLP is trained in the compilation phase and proper code binaries are generated along with the configuration for the neural accelerator (NA). In the execution phase, the source code is executed on a closely coupled CPU and neural accelerator (NA) pair to obtain better speed up and higher energy efficiency over CPU only compute.

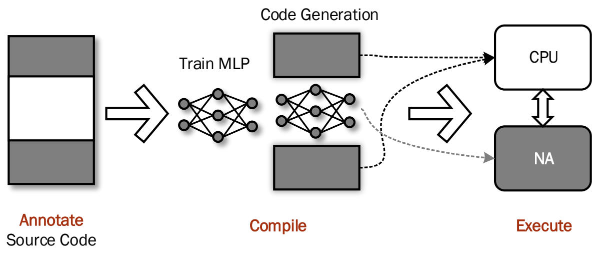
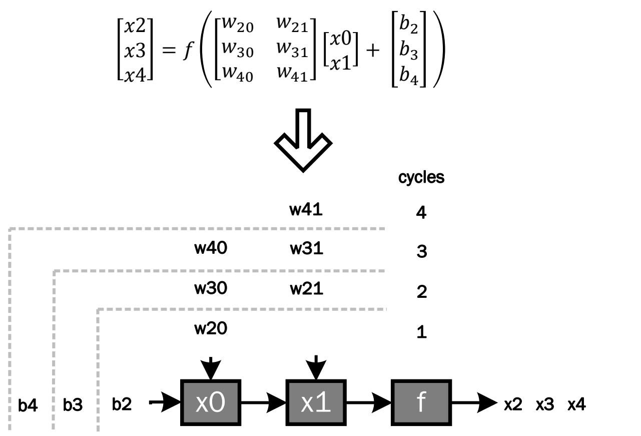


Fig. 2: Neural acceleration at a glance: source code is annotated and corresponding input/output pairs are recorded, a MLP is trained using the recorded input/output pairs, finally, the compiled code is executed on a closely coupled CPU and neural accelerator (NA) pair.

Numerous different architectures have been proposed in the literature for a neural accelerator (NA). However, most of the existing architectures have two common implementation styles - spatially expanded and time multiplexed [8]. Spatially expanded neural accelerators are customized for targeted applications. For instance, Temam et al. [9] proposed a spatially expanded MLP with 90 inputs, 10 outputs and one hidden layer with 10 neurons. The proposed hardware is geared towards the study defect tolerance in ANNs. Their fixed structure, however, limits applicability to a wide range of applications. Time multiplexed neural accelerators compute each MLP layer successively over time. They can run different MLP topologies and are most suited for general purpose code acceleration.

Existing time multiplexed neural accelerators like RNA [10] address the mismatch problem between the available computational resources on the accelerator and the size of the MLP to be scheduled. RNA supports a scheduling framework that assigns the best computational pattern to each MLP layer. SNNAP [7] proposes a one-dimensional systolic array based neural accelerator targeted towards Xilinx Zync SoC. In this work, the architecture proposed in [7] is adopted as a baseline for performing approximations. Fig. 3 shows the computation of a single MLP layer (see Fig. 1) on this architecture.

Proposed work extends the neural approximation approach from the algorithmic level to circuit level by modifying the neural accelerator using approximate multipliers. Further fine tuning using retraining is performed to gain added benefits.



$$\begin{bmatrix} x2 \\ x3 \\ x4 \end{bmatrix} = f\left(\begin{bmatrix} w_{20} & w_{21} \\ w_{30} & w_{31} \\ w_{40} & w_{41} \end{bmatrix}\begin{bmatrix} x0 \\ x1 \end{bmatrix} + \begin{bmatrix} b_2 \\ b_3 \\ b_4 \end{bmatrix}\right)$$
cycles
w41
4
w40
w31
3
w30
w21
2
w20
1
b4
b3
b2
x0
x1
f
x2 x3 x4

Fig. 3: 1D systolic array based neural accelerator: two MAC units followed by a relu unit.

C. State-of-the-Art Approximate Multipliers

The focus of this work is to exploit the inherent error-resilience of MLPs by using functionally incorrect multipliers. Previous work on approximate multiplier design, such as, [11] shows a architectural space exploration methodology for approximate multipliers. Approximate versions of 2x2 multiplier blocks are generated and these blocks are iteratively used to generate multiplier of higher bitwidth. A rounding based approximate multiplier (RoBA) is proposed in [12] which rounds the operands to the nearest exponent of two. EvoApprox8b [13] propose a library of approximate adders and multipliers. These approximate designs are evolved by a multi-objective Cartesian Genetic Programming (CGP). [5] proposes a methodology for the design of well-optimized power efficient ANNs with a uniform structure. Approximate multipliers generated using CGP are used to approximate the ANNs and the results are evaluated for classification problems (MNIST).

This work significantly differs from the existing research efforts. Proposed approach focuses on modification of the netlist using approximate standard cells rather than modifying the behavioural description of the multiplier. This approach can be applied in sync, or independent of the existing approaches.

III. Error Analysis of MLPs

This section outlines the error tolerance analysis for a MLP. A robotic application, **inverse kinematics**, is chosen as our benchmark application for this analysis. Critical design constraints are obtained from the application end which are systematically translated to circuit level constraints for designing approximate multipliers.

Inverse Kinematics: Inverse kinematics (*inversek2j*) is a robotic application which uses kinematic equations to calculate the angles of a 2-joint robotic arm. As shown in Fig. 4, input dataset is the position of the target (X, Y) and the output is the two angles of the robot-arm (THETA1, THETA2). The length of the 2-joint robot arms are denoted by L1 and L2 respectively. For our experiments, L1 is assumed to be 10 units and L2 as 7 units. Also, THETA1 is restricted to the

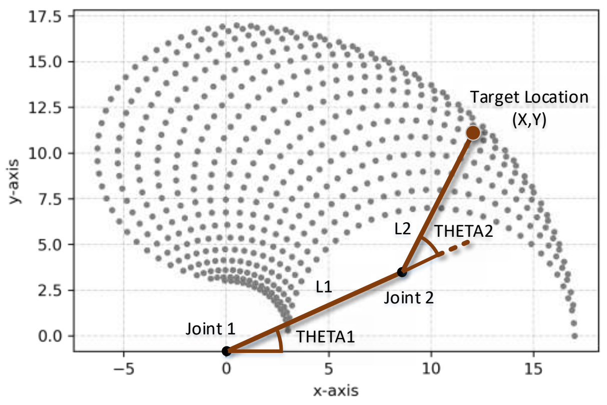


Fig. 4: Inverse kinematics for 2-joint robot arm: L1 and L2 denote the two joint robot arms. The grey points denote the different coordinates (X,Y) used for training the MLP.

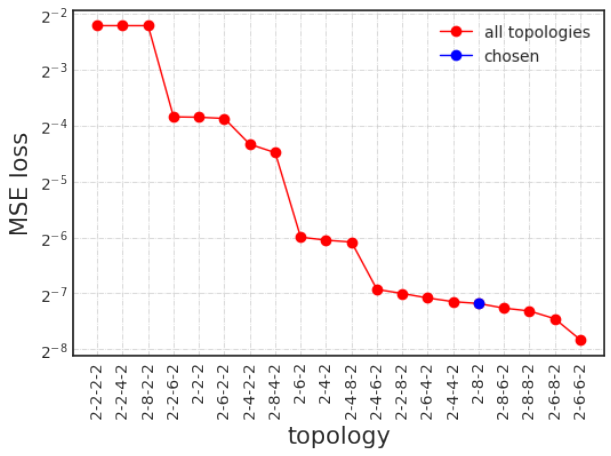


Fig. 5: MLP topological search for inversek2j: The format for MLP topology is (no. of inputs)-(no. of neurons in hidden layer1)-(no. of neurons in hidden layer2)-(no. of outputs). MSE loss is the mean squared error between MLP output and the actual training output.

range 0 to $\frac{\pi}{2}$, whereas THETA2 is restricted to the range 0 to π.

Training and Topology Selection: For generating the training dataset, forward kinematic equations are used. Different valid target positions (X,Y) are generated by sweeping THETA1 and THETA2 in steps of 0.1π across their chosen ranges (see Fig. 4). This dataset is then used to train different MLP topologies using Pytorch framework[1]. The backpropagation algorithm is used for training the MLPs. Mean squared error (MSE) between the actual and predicted outputs is used as the loss function. An exploration search over different MLP topologies is performed as shown in Fig. 5. The search space is limited by restricting the MLPs to 2 hidden layers and at most 8 neurons per hidden layer. Proposed work considers a time-multiplexed NA architecture which computes the results one layer at a time. For better energy efficiency and speed up, MLP topology [2-8-2] is chosen for all future experiments. As shown in Fig. 5, it gives a good trade-off between MSE loss and the number of MAC operations required per input sample (∼1% increase in MSE loss for 4 less MAC operations

[1]https://github.com/pytorch/pytorch

compared to [2-6-6-2]).

Error Evaluation of MLP: In the inference stage, MLP [2-8-2] is tested using new test datasets obtained by sweeping THETA1 and THETA2 across their chosen ranges in random steps. It gives a mean positional deviation[2] of 4.3% when precise 8-bit signed multipliers are used. Probing across the precise multipliers, and plotting output distribution shows a peak over zero (see Fig. 6(a)). This denotes that majority of the multplier inputs are zero. The sensitivity of MLP [2-8-2] to errors is analyzed by emulating imprecise multiplication using a error bias δ. Three different cases are considered:

(a) *bidirectional:* $mul_{\text{approx}}(a, b) = a \times b \pm \delta$
(b) *overestimation:* $mul_{\text{approx}}(a, b) = a \times b + \delta$
(c) *underestimation:* $mul_{\text{approx}}(a, b) = a \times b - \delta$

A [2-8-2] MLP has two levels of multiply and accumulate (MAC) operation. Level 1 corresponds to generation of hidden neurons whereas level 2 corresponds to the generation of final outputs. A study is performed by varying the error bias δ in the above mentioned three cases, and then introducing them to both level1 & level2 separately as well as simultaneously. Fig. 6(b) shows the impact of introducing bidirectional errors on both levels of the MLP. Fig. 6(c) plots error behaviour of the MLP for all three cases across x-axis, y-axis & diagonal of Fig. 6(b). Y-axis shows the positional deviation normalized with respect to the sum of lengths of the two robot arms. The timed multiplexed 1D systolic array architecture (see Fig. 3) adopted in this work allows only diagonal traversal along Fig. 6(b), as the multiplier units are reused over time.

Following conclusions are drawn from the error analysis:

(a) Errors in level2, which generates the output, are more critical than errors in level1 (see Fig. 6(c)).
(b) Errors in the multipliers should be bounded within 5% of the maximum multiplier output ($2^{2\times7}$) to obtain tolerate positional deviation ($\leq$ 25%), as shown in Fig. 6(b).
(c) Inexact multipliers which perform correct multiplication by zero needs to be chosen (see Fig. 6(a)).
(d) Approximate multipliers with bidirectional errors are more suitable for MLP approximation as the errors are more likely to compensate over multiple cycles of accumulation (see Fig. 6(c)).

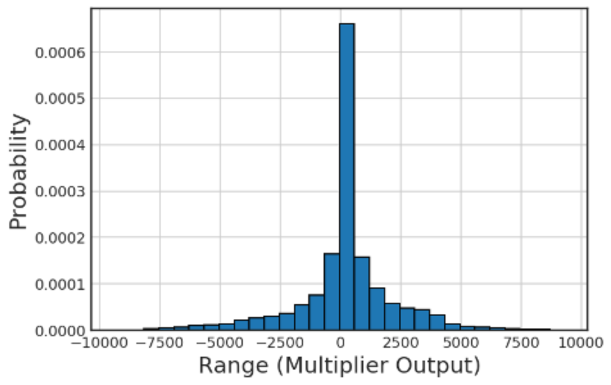
IV. Design Methodology for Approximate Multipliers

A systematic methodology for approximation of multipliers is used which replaces and modifies the netlist of the multiplier using approximate standard cells. The key steps involved in the process are outlined in this section.

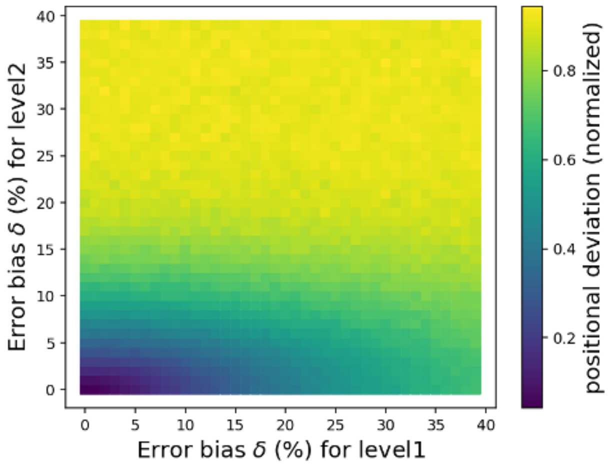
A. Design and characterization of approximate standard cells

The design process starts with manually building approximate versions of standard cells. Approximate versions of standard cells are designed by modifying the truth table entries of the accurate cells. Manual creation of approximate versions

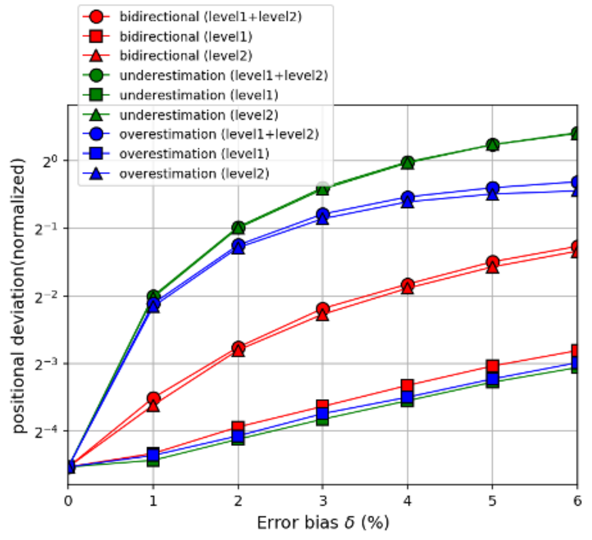
[2]euclidean distance between input coordinates (X,Y) and obtained coordinates (X',Y')



(a) Output distribution of accurate multipliers.



(b) Bidirectional error response on positional deviation.



(c) Error bias versus positional deviation (corner cases).

Fig. 6: Error analysis of MLP [2-8-2].

of all the cells in the design is a tedious task. Therefore, early design-space reduction is performed by properly analysing the energy distribution of the accurate multiplier as shown in Fig. 7. As evident from Fig. 7, 4-to-2 compressors and 1-bit full adders account for more than 70% of the total energy consumption of the 8-bit multiplier circuit. Therefore, multiple inexact versions of these 4-to-2 compressor and 1-bit full-

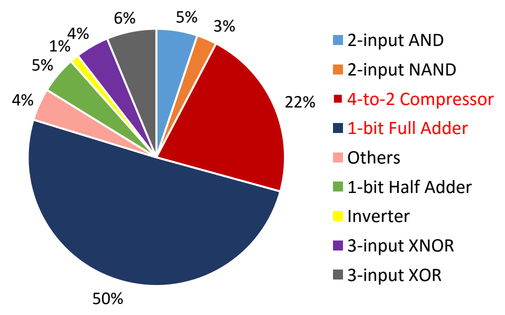


Fig. 7: Standard cell based energy distribution for 8-bit signed multiplier (tool synthesized using TSMC 40 nm).

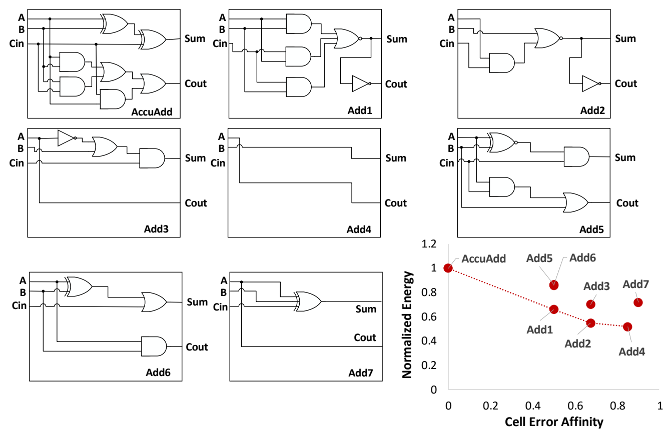


Fig. 8: Design and characterization of 1-bit full adder cells.

adder cells are generated manually. This is followed by further pruning of inefficient cell designs in the pareto-frontier of energy vs accuracy.

Design of Standard Cells: Fig. 8 shows different variants of approximate full adder cell designs. Gate-level implementation of an accurate 1-bit full-adder cell is given in AccuAdd (see Fig. 8). Add1 - Add5 show state-of-the-art approximate designs implemented in gate-level according to their truth tables reported in [14] whereas Add6 & Add7 are additional low-power designs adopted from [11]. Different gate-level implementations of 4-to-2 compressor cells are shown in Fig. 9. Accurate compressor cell is first implemented (as shown in AccuComp, Fig. 9). An approximate low-power compressor cell implementation is adopted from [15] (shown in Comp1, Fig. 9). Additionally, five new low power compressor designs are considered (Comp2 - Comp6, Fig. 9) with variable quality characteristics.

Characterization: For proper characterization of the above mentioned designs, a cost function, *Cell Error Affinity* (CEA), is used. It is a weighted sum of three quality metrics [11] [16]: (a) Error Count (EC) (b) Maximum Error Distance (MaxED) (c) Mean Error Distance (MED). The formulation of these quality metrics is given in equations 1, 2 & 3, where M' is the approximate result and M is the accurate result. Eq. 4 gives Cell Error Affinity (CEA) where $\widehat{EC}$, $\widehat{MaxED}$ & $\widehat{MaxED}$ represent the normalized forms of the respective

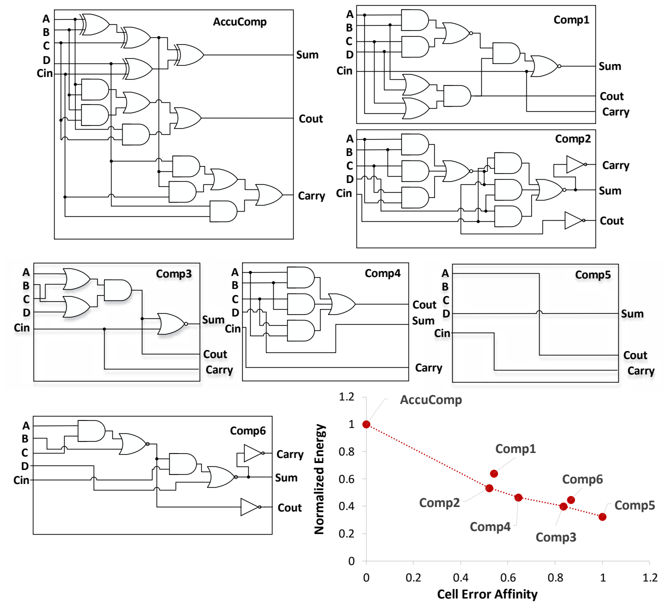


Fig. 9: Design and characterization of 4-to-2 compressor cells.

metrics (w.r.t to the worst case possible) while α, β, γ are their corresponding weights satisfying $\alpha + \beta + \gamma = 1$.

$$EC = \sum \begin{matrix} 1 \; if \;\; M' \neq M \\ 0 \; otherwise \end{matrix} \quad \forall \; val \;\; \in T \tag{3}$$

$$MaxED = max\{|M' - M| \; \forall \; val \;\; \in T\} \tag{4}$$

$$MED = \frac{1}{|T|} \sum_{n=1}^{|T|} |M' - M| \tag{5}$$

$$where \; T = \{val | val \in \mathbb{Z}^{+}, |val| < 2^{input\_count}\}$$

$$CEA = \alpha \times \widehat{EC} + \beta \times \widehat{MaxED} + \gamma \times \widehat{MED} \tag{6}$$

A weight configuration of $\alpha = 0.2$, $\beta = 0.3$, $\gamma = 0.5$ is chosen for the pareto-optimality study shown in Fig. 8 & Fig. 9. However, any weight configuration can be selected depending on the preferences of the designer. The cells were synthesized using Cadence Encounter RTL Compiler and mapped to a 40nm TSMC technology library. Design space reduction is performed by pruning inefficient designs which are non-optimal on the pareto-froniter as shown in Fig 8 & Fig. 9. In case of full-adder cells, add3, add5, add6 & add7 are pruned whereas comp1 & comp6 are pruned in case of compressor cells.

B. Replacement strategy of standard cells

A graph based systematic cell replacement strategy is used in this work which allows controlled modulation of quality. At first, the target multiplier design is synthesized and mapped to a technology library to get the gate-level netlist. This gate-level netlist is converted into a directed acyclic graph (DAG), where the nodes represent standard cell instances and the edges represent wires. A multiplier circuit follows a notion of bit significance where each bit has a two times higher significance than its previous bit, when moving from LSB to MSB. Therefore, cell replacement is started from the LSB which moves toward the MSB until the quality constraint is violated. For a particular primary output bit, a backward dependency search is done using reverse graph traversal. This extracts the standard cell types and instance names responsible for corresponding bit generation. Once the cell instance names are known, they are replaced with approximate cells in the library having the same type to give an approximate netlist. Proper combination of approximate cells is treated as a configurable knob which is applied to obtain different approximate versions of the multiplier having varying accuracy.

V. Experimental Results

The first part of this section outlines the effectiveness of used methodology for the generation of approximate multipliers. The second part focuses on approximation of the systolic neural accelerator using proposed multipliers.

A. Approximate multipliers using netlist modification

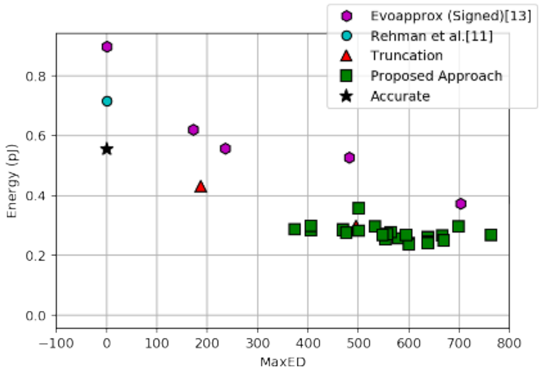
Quality constraints obtained from the error analysis of MLP (see section III) are used to generate approximate mutipliers. Maximum error bound of 5% is chosen. All the multiplier circuits under test were synthesized using Cadence Encounter RTL Compiler at nominal operating conditions, mapped to a 40nm TSMC technology library and functionally verified using Cadence Incisive Enterprise Simulator.

In order to properly evaluate the designs under test, three different quality metrics are considered, Maximum Error Distance (MaxED), Mean Error Distance (MED) and Mean Relative Error Distance (MRED). Equations 3-5 & 7 gives the formulation of these quality metrics. A maximum error bound of 5% (see section III) translates to MaxED = 819, MED = 408 and MRED = 0.25. All the approximate multipliers generated by our proposed method adhere to these constraints. Also, the designs were tested using actual input data obtained from traces of *inversek2j* kernel.

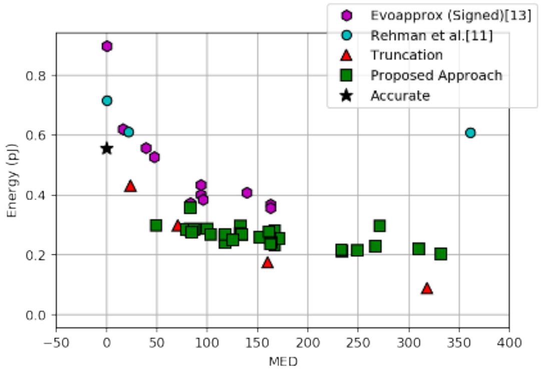
$$MRED = \frac{1}{|T|}\sum_{n=1}^{|T|}\frac{|M'-M|}{M} \tag{7}$$

$$where\ T = \{val|val \in \mathbb{Z}^{+}, |val|<2^{input\_count}\}$$

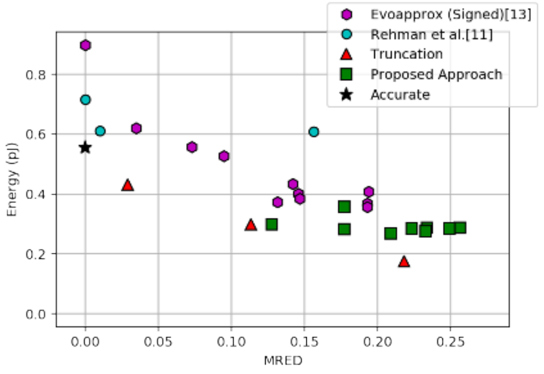
Fig. 10 shows the energy improvements obtained for a 8-bit signed multiplier. Accurate tool synthesized multiplier at relaxed frequency is chosen as the baseline. Multiple approximate design points are synthesized and tested while adhering to the mentioned quality constraints. Proposed designs achieves energy savings of 2.5× with respect to the accurate design for hardly any quality degradation (MaxED $\sim$ 600, MED $\sim$ 240 & MRED $\sim$ 0.22), as can be seen in Fig. 10. Comparative analysis with a state-of-the-art low power approximate multipliers [11] shows upto 3× improvements in energy, for almost similar qualtiy degradation (MED between 300 to 400)(see Fig. 10(b)). Similarly, a comparison with



(a) Energy versus MaxED.



(b) Energy versus MED.



(c) Energy versus MRED.

Fig. 10: Comparative analysis of proposed multipliers with state-of-the-art multipliers.

EvoApprox8 [13][3] shows that our designs benefit by 1.7× in terms of energy, for similar quality loss (MED $\sim$ 160)(see Fig. 10(b)).

Observations: The accurate tool synthesized multiplier chosen as baseline in this work dominates the approximate designs proposed in [11] & [13] (see Fig. 10). *This stresses the fact that a proper baseline choice is very essential.* Also, comparison with input bit-width scaling/ truncation of the multiplier suggests that the truncated designs are pareto-dominant compared to both the state-of-the-art designs proposed in [11] and [13] as well as the proposed designs. *The reason behind this observation is that the quality metrics (MaxED, MED & MRED) are calculated using absolute value of the*

[3]the multipliers are converted to signed for proper comparison

errors and truncated designs provide a lower average absolute error. However, for analysis on approximate neural replication of inversek2j, we have to implement these designs in the 1D systolic MLP accelerator. Truncated designs chosen in this study always produce underestimating error, whereas the proposed designs produce bidirectional errors. This makes the proposed designs more suitable for approximation of MLP as justified in section III.

B. Approximate neural acceleration of inversek2j

A 1D systolic array architecture with 8 MAC units is chosen as the baseline for all our experiments. The design is synthesized using Cadence Encounter RTL Compiler at nominal operating conditions. It is mapped to a 40nm TSMC technology library and functionally verified using Cadence Incisive Enterprise Simulator. A relaxed operating frequency of 200 MHz is chosen for analysis.

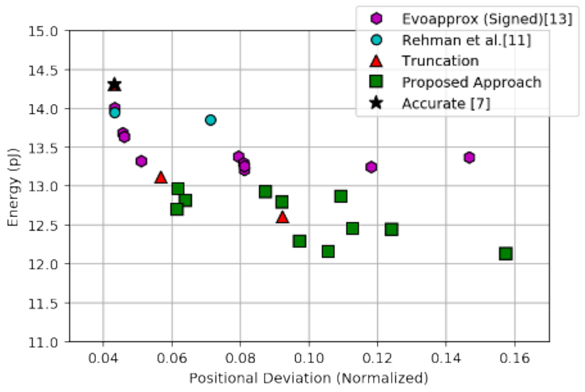


Fig. 11: Neural replication of *inversek2j* using approximate multipliers.

Fig. 11 shows the impact of using approximate multipliers on the neural acceleration of *inversek2j*. X-axis gives the positional deviation of the input co-ordinates (X, Y) due to approximate calculation of the robot-arm angles (THETA1, THETA2). Y-axis gives the system level energy consumption, which includes both the MAC compute units and the control units. Our multiplier designs dominate the state-of-the-art implementations on the pareto-front giving better energy efficiency (1.1×-1.2×) for similar positional deviation ($\sim$ 0.12). Also, proposed approximate designs improve upon the system level energy consumption of the accurate design by 10.5% for less than 7% positional deviation. A comparative analysis with truncation shows that the proposed multipliers outperform the truncated ones in the pareto frontier. This further justifies the statement that designs with bidirectional error behaviour are more suited for MLP approximation than the ones with underestimating behaviour (see section III).

All the designs plotted in Fig. 11 are synthesized for the same relaxed frequency of 200 MHz. The observations in this work overlook the fact that using approximate multipliers reduces the critical path latency of the design. This reduced latency can be used for voltage scaling (keeping the frequency constant), which gives potential for improved power savings. Also, the reduced latency can be used to operate the design at higher frequencies, thus, obtaining better speed up as shown in Fig. 12.

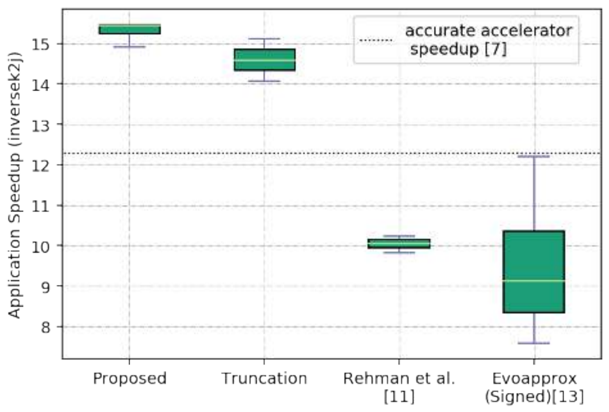


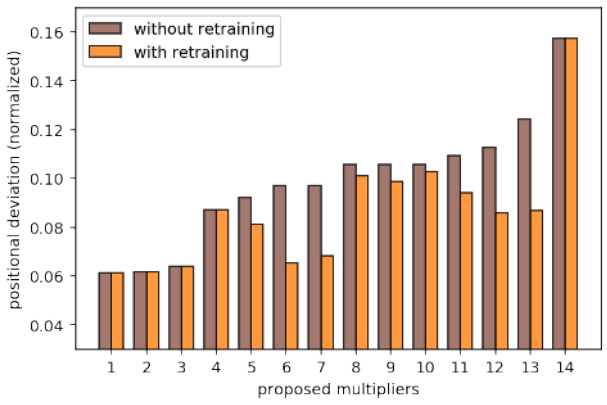
Fig. 12: Speed up obtained using approximate MLP accelerator w.r.t to the software only execution.

Retraining of approximate MLP designs: Proposed MLP accelerator designs with approximate multipliers are retrained to obtain lower normalized positional deviation, thus, giving better energy versus accuracy trade off. For retraining, fixed point training is performed on the MLP network as suggested in [17] & [18]. Since the weights in the MLP network have only discrete values, the challenge lies in updating the weights after backpropagation due to limited precision. To overcome this, updates are performed on the full precision floating point weights, called shadow weights. These shadow weights are further converted to fixed point during forward propagation.

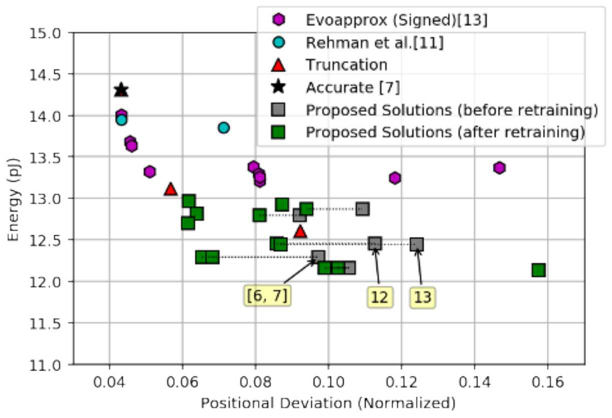
Fig. 13(a) shows the improvements in the normalized positional deviation obtained with retraining of the MLP network. On an average, 12% improvement in normalized positional deviation is obtained for all proposed multipliers. Best case improvement of upto 32% is obtained (see Fig. 13(a), col 6). Fig. 13(b) shows four best case improvements obtained from retraining (see yellow boxes). Also, Fig. 13(b) shows that our proposed approximate designs improve upon the system level energy efficiency of the accurate accelerator by upto 14% for less than 7% positional deviation.

VI. Conclusion

Error resiliency of applications can be exploited by approximating them using multi-layer perceptrons (MLPs) with insignificant degradation in output quality. Faster and energy efficient execution of such an application can be achieved using a MLP accelerator. This work approximates the MLP accelerator by approximating the MAC compute units using energy efficient approximate multipliers. An error resilience analysis of the MLP is performed to obtain key constraints which are used for designing energy efficient approximate multipliers. A systematic methodology for the design of approximate multipliers is used. A graph based netlist modification approach is considered. Approximate versions of basic standard cells are generated which are used to replace accurate cells in the netlist in a systematic quality controlled manner. On an average, 2.5× savings in energy is observed



(a) Improvements in positional deviation (normalized) with retraining for proposed multipliers.



(b) Pareto-optimality study of retrained proposed solutions with respect to the state-of-the-art approximate multipliers.

Fig. 13: A study of retraining on approximate MLP accelerator.

for hardly any quality degradation. Proposed approximate multipliers are further used for approximating MLPs. The results are validated by using approximate neural replication of a robotic application, *inversek2j*. System level energy savings of upto 14% is obtained for less the 7% degradation in output quality. Average application speedup of 24% is obtained over accurate MLP accelerator. The results are compared with state-of-the-art approximate multipliers and a comparison with truncation (bit-wise scaling) is performed. Moreover, error healing capability of MLPs is shown by studying the impact of retraining on networks with approximate multipliers. Future work will aim to consider voltage scaling (keeping the operating frequency fixed) to take advantage of the fact that using approximate multipliers reduces the critical path latency of the design. Also, trade off analysis in terms of opportunities for approximation between spatially expanded and time multiplexed MLP accelerators will be considered in future.

Acknowledgment

This research has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under grant agreement no 674875 (oCPS).

References

[1] C. A. MacK, "Fifty years of Moore's law," in *IEEE Transactions on Semiconductor Manufacturing*, vol. 24, no. 2, 2011, pp. 202–207.

[2] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, Oct 1974.

[3] M. Shafique and S. Garg, "Computing in the dark silicon era: Current trends and research challenges," *IEEE Design and Test*, vol. 34, no. 2, pp. 8–23, 2017.

[4] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," *Proceedings of the 50th Annual Design Automation Conference on - DAC '13*, no. i, p. 1, 2013.

[5] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," *Proceedings of the 35th International Conference on Computer-Aided Design - ICCAD '16*, pp. 1–7, 2016.

[6] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '12*, p. 301, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2150976.2151008

[7] T. Moreau, M. Wyse, J. Nelson, A. Sampson, H. Esmaeilzadeh, L. Ceze, and M. Oskin, "SNNAP: Approximate computing on programmable SoCs via neural acceleration," *2015 IEEE 21st International Symposium on High Performance Computer Architecture, HPCA 2015*, pp. 603–614, 2015.

[8] F. Tu, S. Yin, P. Ouyang, L. Liu, and S. Wei, "Reconfigurable Architecture for Neural Approximation in Multimedia Computing," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 4, pp. 1–14, 2018.

[9] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, June 2012, pp. 356–367.

[10] F. Tu, S. Yin, P. Ouyang, L. Liu, and S. Wei, "Neural approximating architecture targeting multiple application domains," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 2509–2512.

[11] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel, "Architectural-space exploration of approximate multipliers," in *Proceedings of the 35th International Conference on Computer-Aided Design - ICCAD '16*, 2016, pp. 1–8.

[12] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "RoBA Multiplier: A Rounding-Based Approximate Multiplier for High-Speed yet Energy-Efficient Digital Signal Processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 2, pp. 393–401, 2017.

[13] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapproxsb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 258–261.

[14] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, Jan 2013.

[15] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 984–994, 2015.

[16] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, pp. 60:1–60:34, Aug. 2017.

[17] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, pp. 3123–3131.

[18] P. Gysel, "Ristretto: Hardware-oriented approximation of convolutional neural networks," *CoRR*, vol. abs/1605.06402, 2016. [Online]. Available: http://arxiv.org/abs/1605.06402