Leveraging the Error Resilience of Machine-Learning Applications for Designing Highly Energy Efficient Accelerators

Zidong Du[†]
duzidong@ict.ac.cn

Avinash Lingamneni[‡]
avinash.l@rice.edu

Yunji Chen[†]
cyj@ict.ac.cn

Krishna Palem[‡]
palem@rice.edu

Olivier Temam[§]
olivier.temam@inria.fr

Chengyong Wu[†]
cwu@ict.ac.cn

[†]CARCH*, ICT, CAS, China

[‡]Rice University, USA

[§]INRIA, France

Abstract— In recent years, *inexact computing* has been increasingly regarded as one of the most promising approaches for reducing energy consumption in many applications that can tolerate a degree of inaccuracy. Driven by the principle of trading *tolerable* amounts of application accuracy in return for significant resource savings—the energy consumed, the (critical path) delay and the (silicon) area being the resources— this approach has been limited to certain application domains. In this paper, we propose to expand the application scope, error tolerance as well as the energy savings of inexact computing systems through neural network architectures. Such neural networks are fast emerging as popular candidate accelerators for future heterogeneous multi-core platforms, and have flexible error tolerance limits owing to their ability to be *trained*. Our results based on simulated 65nm technology designs demonstrate that the proposed inexact neural network accelerator could achieve 43.91%-62.49% savings in energy consumption (with corresponding delay and area savings being 18.79% and 31.44% respectively) when compared to existing baseline neural network implementation, at the cost of an accuracy loss (quantified as the Mean Square Error (MSE) which increases from 0.14 to 0.20 on average).

I. Introduction

Due to stringent energy constraints, researchers have started to accept that holistic approaches for saving energy are required, which span all computing system aspects, from circuits to applications. One of the most promising approaches for saving energy is to trade application accuracy for energy savings, popularly referred to as *inexact* (see [13] for additional references and [22] for early examples)or *approximate* [8] computing. The driving philosophy behind this approach is that some if not many applications do not require the degree of accuracy imposed by current circuit or programming methods; for instance, video decoding can tolerate a fair degree of inaccuracy because the image variations may not be perceived by the human eye. Some have started to take advantage of that observation by reducing the amount of logic, i.e., the logic operators' accuracy, used for the computations. These techniques advocated the reduction of logic density by either pruning/deletion of components that are not so significant [17], or by transformations to lesser power consuming yet similar logic that is ”close” to the original design [6, 18].

While this concept is attractive, in practice, when applied to ASICs taking advantage of the specific interplay between logic operators within a given circuit datapath for the targeted application error [13], these ASICs (or programs run on them) have limited error tolerance to inexact computations as currently designed. Furthermore, the concept of approximate computing has had limited application to general-purpose processors because the control logic is highly susceptible to faults [8].

We seek to overcome these limitations by leveraging two trends: one in architecture and the other in applications. The trend in architecture is driven by increasing energy constraints: it is projected that multi-core architectures (embedded and high-performance) will increasingly rely on *accelerators* for most computationally intensive tasks (heterogeneous multi-cores) owing to the lack of voltage scaling leading to an effect known as Dark Silicon [20]. Beyond traditional accelerator solutions such as GPUs, FPGAs and ASICs, researchers are increasingly investigating accelerators or co-processors with an intermediate scope in terms of generality, covering more applications than ASICs but less than general-purpose processors, in order to reap significantly higher energy benefits than GPUs or FPGAs [23, 10]. Considering multiple such accelerators can be implemented within a chip today, the union of the application scopes of these accelerators can be very broad indeed.

The second trend is related to the nature of emerging computing-intensive applications. A few years ago, Intel has called the attention of the architecture community to the fact that its benchmarks were ill-designed, and encouraged the consideration of Recognition, Mining and Synthesis (RMS) applications [7] as more representative of emerging high-performance applications. As we know, this push later led to the PARSEC benchmarks [4], a collaboration between Intel and Princeton University.

Recently, it has been highlighted [24] that the combination of these two trends can lead to an atypical but highly attractive accelerator option based on *a hardware neural network*. Chen et al. [5] have shown that for a significant portion of the PARSEC benchmarks, 90% or more of their execution time could potentially be offloaded to a hardware neural network accelerator. Therefore, neural networks can be regarded as accelerators based on which a broad set of applications can be implemented. While several other accelerator options exist for supporting the PARSEC benchmarks, from our perspective, neural networks are particularly attractive because of their ability to inherently

*State Key Laboratory of Computer Architecture

978-1-4799-2816-3/14/$31.00 ©2014 IEEE

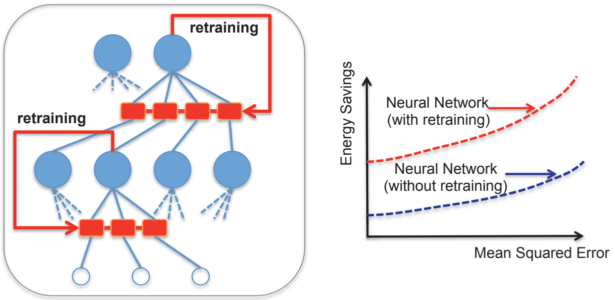


Fig. 1. Illustrating the benefit of retraining for neural network error tolerance.

tolerate errors. For instance, it was recently shown that a hardware neural network accelerator would be capable of tolerating transistor-level faults [24].

With this as a backdrop, we start from the following observations: (1) contrary to common wisdom, the application scope of a hardware neural network accelerator is very broad in light of emerging computing-intensive applications, (2) in spite of that broad application scope, a hardware neural network is just one large datapath circuit, with no complex control as in a core, and as a result, it can directly benefit from inexact arithmetic circuit optimizations, and (3) it implements an algorithm which has an unique capability of tolerating errors due to retraining.

Specifically, we take advantage of these three observations to extend the reach of inexact computing. First, by applying inexact computing to a hardware neural network accelerator, we demonstrate the energy savings for applications which can take advantage of that accelerator explicitly. Second, we take advantage of the error tolerance capability of neural networks to improve the degree of inaccuracy that can be tolerated and thus, resulting in greater energy savings. This capability derives from the learning/training algorithm used for neural networks (qualitatively shown in Figure 1). Using a hardware neural network accelerator with inexact logic operators, and a set of machine-learning tasks from the UCI Machine-Learning repository, we show that it is possible to achieve energy savings of 53.75% on average, and an area reduction of 31%, compared to a hardware neural network accelerator with standard logic operators, at a loss of only 0.06 in Mean Squared Error (MSE).

In Section II, we outline the main concepts of inexact computing and hardware neural networks, and we show how both of them can be combined, which represent the goal and the results of this paper. In Section III, we explore the design space formed by inexact hardware neural networks accelerators, the potential energy, delay and area savings, and the concomitant impact on task accuracy. We describe some of the related work that is most relevant in Section IV, and conclude in Section V.

II. *Inexact* Hardware Neural Networks

In this section, we first present a brief primer on inexact computing and neural networks before proceeding to demonstrate how they can be combined to improve the energy efficiency of hardware neural networks without substantially degrading their accuracy. These techniques and concepts are based on previously known work and are thus, not claimed to be novel. The novel contribution of this paper is the characterization of the benefits and associated costs represented by increased error, derived by combining inexact computing with Neural Network (hardware) accelerators as summarized in Section III.B.

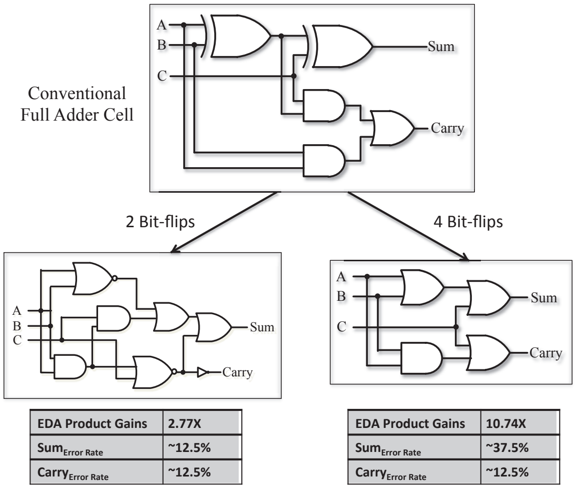
A. A Primer on Inexact Computing

Let us consider a circuit that computes a completely-specified boolean function $\mathcal{F} : B^n \rightarrow B^m$ that maps n-input boolean vector of the form $\mathbf{x} = [x_1, x_2, \dots x_n]$ to a vector $\mathbf{y} = [y_1, y_2, \dots y_m]$ with an associated hardware cost function $\mathbf{C}_{\mathcal{F}}$. The goal of (heuristic) inexact logic minimization (paraphrased from [1] is to find a Boolean function $\mathcal{F}' : B^{n'} \rightarrow B^{m'}$ where $n' \leq n$ and $m' \leq m$, such that its cost $\mathbf{C}_{\mathcal{F}'}$ is heuristically minimized subject to the following constraint:

$$\sum_{\forall\, \vec{i} \in \mathbf{I}} \frac{|\mathcal{F}(\,\vec{i}\,) - \mathcal{F}'(\,\vec{i}\,)|}{L} \leq Er_{th} \qquad (1)$$

where $\mathbf{I}$ is a set of L-testbench vectors to this circuit and Er_{th} represents the average error (see [1] for a formulation).

Such inexact logic minimization (ILM) can be achieved by logic-level alteration of boolean function using the notion of intentional *bit flips*, which implies an intentional forcing of the output from $0 \rightarrow 1$ or $1 \rightarrow 0$ for certain input vectors. Some application-specific contexts where this idea has been explored can be found in [6, 1, 2], among others. One example of such a hardware building block synthesized and studied is a 3-input 2-output full adder cell and is shown in Figure 2 (from [2]).



EDA Product Gains	2.77X
Sum$_{\text{Error Rate}}$	~12.5%
Carry$_{\text{Error Rate}}$	~12.5%

EDA Product Gains	10.74X
Sum$_{\text{Error Rate}}$	~37.5%
Carry$_{\text{Error Rate}}$	~12.5%

Fig. 2. Inexact design through inexact logic minimization: few examples from [2].

In this paper, we will use a portion of the inexact logic minimization approach from [1]. Specifically, to tackle the combinatorial explosion of the solution space for a given accuracy constraint from Equation (1) above, we use a *significance*-guided greedy heuristic combined with stochastic exploration to narrow down the search space. Given the richness of the solution space, even this heuristic-based exploration resulted in good design points. In our approach, as we are targeting datapath circuit design, we use an output-significance driven ranking of the nodes similar to previous works [18], i.e., nodes feeding outputs with higher (binary) significance are assigned higher rank and correspondingly are allotted lesser inexact configurations. As in the previous work [1], this significance or rank

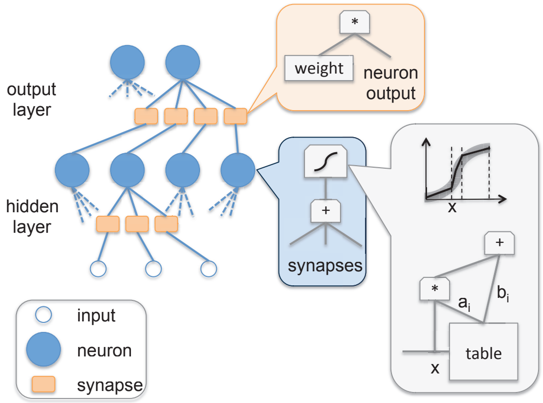


Fig. 3. Structure and logic operators of a 2-layer MLP, including the sigmoid interpolation.

is used to guide the decision on determining as to which elements of the circuit can accept more error in a relative sense. We then use this significance array as the template for guiding the amount of error introduced at each node.

We have performed such a heuristic-based exploration for the operators found to be most critical in a neural network, and built a library of these operators. The nature of the operators and the libary are described in Section II.D and in Section III.A.

B. A Primer on Hardware Neural Networks

Several types of artificial neural networks are used in machine-learning. A broadly used type of neural networks are Multi-Layer Perceptrons (MLPs) [12]. Convolutional Neural Networks (CNNs) [16] are also popular, and recently, a form of artificial neural networks, called Deep Belief Networks (DBNs), have been shown to be very powerful techniques again, outperforming Support Vector Machines (SVMs) and other techniques [15]. DBNs have a structure similar to MLPs or CNNs with broader and more layers. Here, we focus on MLPs, and the techniques proposed in this paper are compatible with any width or number of layers.

MLP. A typical MLP is a 2-layer feed-forward network: information flows come from an *input layer* (which contains no neurons) through a *hidden layer*, to an *output layer*. Following [12], each neuron is fully connected to the neurons of the previous layer, and each connection carries a synaptic weight. Each evaluation of one neuron j implements $\sum_{i=0}^{N} f(w_{ji} \times y_i)$ where y_i the output of neuron i of previous layer, w_{ji} is the synaptic weight between neurons i and j, and f is an activation function (sigmoid), typically $f(x) = \frac{1}{1+exp^{-x}}$. Following conventional approaches, in hardware, the activation function can be efficiently implemented using the piecewise linear approximation ($f(x) = a_i \times x + b_i$), through a small look-up table to store coefficients (a_i, b_i), an adder and a multiplier (Figure 3). Using a 16-segment approximation, we observed that the maximum sigmoid accuracy loss was 2.3% and this had no noticeable degradation of the MSE of the neural network.

Feed-Forward hardware. We use back-propagation [12], the most popular training algorithm for neural networks. However, we do not implement back-propagation in hardware; the hardware neural network only contains the feed-forward path. There is a frequent and important misconception that *on-line* training is necessary for many applications. On the contrary, for many industrial applications *off-line* training is sufficient; for example, trained on handwritten digits, license plate numbers, a number of faces or objects to recognize, etc. The network can be taken offline periodically and retrained. We advocate this offline training approach in this paper.

C. Melding Inexact Computing and Neural Networks

In spite of its atypical nature, a hardware neural network is just one large datapath circuit, predominantly composed of multipliers (to compute the synaptic weight × neuron output, and activation function), and adders (sum of neuron inputs and the approximation to the activation function).

However, a neural network's desirable feature over other application contexts such as JPEG decompression, is its ability to lower the effects of faulty logic gates by retraining the circuit. As noted earlier, even a large number of faults due to the fabrication process can be compensated for by retraining [24]. In contrast to [24], our approach here is to *voluntarily* introduce introduce errors through inexactness in order to save energy, and to use the retraining capability of the neural network to lower the effects of the resulting functional errors. Retraining is done automatically re-balancing the synaptic weights so that the neurons producing the most errors are suppressed, or at least, their impact is diminished.

D. Sensitivity and Complexity of Exploration

[Dove applicare l'AC nelle NN?]

The total exploration space of inexact neural networks is huge. An exhaustive consideration of all the parameters would result in 1.07×10^{68} different network configurations. Considering we need to train each network configuration in order to evaluate its quality in our experience, the evaluation of each design point is about 100 seconds. So, not only is an exhaustive exploration impossible, but in one year it would only be possible to evaluate a minuscule fraction of about $2.95 \times 10^{-61}\%$ of all possible configurations using a single workstation. At the same time, arbitrarily limiting the exploration to a tiny subset of the design space risks resulting in far too pessimistic solutions.

We address this issue by taking the specific nature of neural networks into consideration. We note that the adders accumulate the weight×output products in each neuron (see Figure 3), are potentially more susceptible to errors than each of the many synaptic weight multipliers which feed these adders. At the same time, the hardware cost of the adders is usually much smaller than the cost of multipliers, and as a result, the area/energy gains that can be expected from making adders inexact is not likely to be large even though their impact on the neuron output is significant. Similarly, the role of the multiplier and the adder used to approximate the activation function is likely to be significant from the standpoint of error, while providing savings equivalent to one multiplier at most. Based on this reasoning, we conclude that the most fruitful portions of the architecture for introducing inexactness are the synaptic weight multipliers. Restricting our attention to these multipliers reduces the total number of configurations to be explored as well.

Exploration can be further reduced by focusing more on the hidden layers of the neural network. Since there is no synaptic weight after the neurons of the output layer, it is more difficult for the training algorithm to lower the error of these neurons, though training can reshuffle the role of each output and thus, still provide some form of error tolerance. Fortunately, most neural networks require large hidden layers, while the number of neurons in the output layer is usually small since the number of classes in a classification problem being solved for example is related to it. So, we bias the design space exploration towards the hidden layer.

To recapitulate and using common knowledge about the error sensitivity of the elements of the neural network architecture and focusing inexactness on the value of information, it is opportune that the significant gains can be realized by restricting the design space exploration to the less error sensitive elements such as the synaptic weight multipliers of the hidden layers while not considering the more error vulnerable adders or synaptic weight multipliers of the output layer.

III. Performance Evaluation

In this section, we describe the methodology, the design space which is explored, and the energy/accuracy tradeoffs achieved using the inexact hardware based neural networks. The material in Section III.B constitute the original contribution of this paper.

A. Methodology

Tool flow. We have explored a set of 7 inexact multipliers using the principles described in Section II.A, using a baseline of standard n-bit truncated multipliers with correction constant from [14]; an n-bit truncated multiplier has n-bit inputs and n-bit outputs, and as a result, it induces an error due to the truncation of output (as well as the corresponding logic generating these truncated outputs) from the full $2n$ bits down to n bits. The characteristics of these 7 inexact multipliers are shown in Table I and correspond to various tradeoffs in accuracy and efficiency.

The functional models of these inexact multipliers has been implemented as C++ subroutines (with corresponding area savings estimates) and are plugged into a C++ based neural network simulator. Specifically, we modified the implementation of the multi-layer perceptron in order to replace the calls to the standard multiplication with calls to the inexact multipliers, for all multiplications between synaptic weights and neuron inputs. This modified software neural network model is used to assess the impact of the inexact multiplier on the neural network accuracy before and after retraining.

The inexact multipliers have also been implemented as configurable Verilog HDL, along with the full neural network, in order to assess the energy, delay and area values. We synthesized the neural network using the Synopsys Design Compiler and the TSMC 65nm standard-V_{th} library, place and route using Synopsys IC compiler, and simulated using the Synopsys VCS and estimated the power using PrimeTime PX tool. The hardware neural network is a 2-stage pipelined architecture with one stage each for the hidden and output layers respectively.

TABLE I
Inexact 16-bit multipliers.

Type	Rel.Err (%)	Area (um^2)	Power (uW)	Delay (ns)
#0 (truncated)	0.09	2068.20	931.41	11.47
#1 (inexact)	0.99	1883.52	725.50	10.46
#2 (inexact)	5.23	1642.32	558.70	8.48
#3 (inexact)	10.70	1537.92	494.59	7.99
#4 (inexact)	17.16	1520.64	468.02	8.84
#5 (inexact)	27.51	1384.92	380.86	7.03
#6 (inexact)	41.65	1298.88	370.81	7.03
#7 (inexact)	98.94	1249.92	364.10	7.03

Benchmarks and the neural network dimension. We evaluated the accuracy of the neural networks on tasks obtained from the UCI Machine-Learning repository [3]. The example cases in that repository are contributed by researchers and engineers from various domains in order to stimulate machine-learning research and thus, they are both diverse in nature and correspond to actual applications. In [24], it was shown that a neural network of 90 inputs, 10 hidden neurons, and 10 outputs could compute 90% of the tasks of the UCI repository and provide good classification results. We implemented a neural network with the same configuration for the 7 benchmarks mentioned in [24]. The benchmarks we used include: *glass, ionosphere, iris, robot, sonar, vehicle* and *wine*. Note that the *accuracy* of a neural network is typically reported as the mean squared classification error (MSE).

B. Design Space Exploration

The configurations we explored were determined by forcing the exploration on the synaptic multipliers of the hidden layers (Figure 3), due to the analysis of Section II.D. While the baseline configuration uses 16-bit operators, we considered bit widths of 8, 10, 12, 14 and 16; bit widths smaller than 8 generally yielded poor MSE. Any of the multipliers in the network can be replaced by any of the 8 multipliers (7 inexact and 1 exact). Overall, we explored 24500 configurations selected randomly out of the total possible configurations.

For each configuration, we estimate the MSE using the C++ neural network simulator on all 7 benchmarks. We separately train the neural network on each benchmark using 10-fold cross-validation, and repeat the experiments 5 times to account for statistical variations where the initial weights of the neural network are randomly set with an uniform distribution. We only retain the configurations for which the MSE is less than 2 times the MSE of the exact neural network, and estimate the area, delay and power cost of a configuration by simply extrapolating the resource cost of individual hardware objects such as multipliers, adders, registers, etc.

TABLE II
Hardware characteristics for baseline, best, bit-width.

-	Best	Bit-Width	Baseline
Area (mm^2)	1.73	2.24	2.61
Delay (ns)	24.97	26.99	29.89
Power (W)	1.02	1.38	1.48
Energy (nJ)	25.40	37.15	44.34
Average MSE	0.20	0.15	0.14

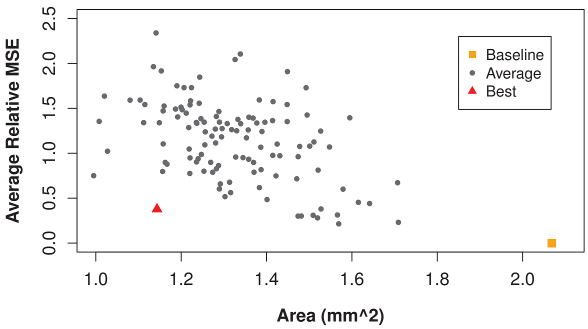


Fig. 4. Solutions of exploration for all 7 benchmarks.

In Figure 4, the relationship between the estimated area (from our C++ framework) and the accuracy across benchmarks is shown. In Table II, we selected a *best* configuration as the one on the area/MSE pareto-optimal front closest to the median between MSE and area (other area/MSE tradeoffs are naturally possible).

We observe that in this particular *best* configuration, there are no 8-bit nor 10-bit synaptic weight or multipliers operands, 44.56%, 44.56% of synaptic weights use 12-bit and 14-bit operands respectively, while the remaining are 16-bit synaptic weights. Among the multipliers in the hidden layer, 41.30% use 14-bit operands, 28.70% use 12-bit operands and the rest 30% use 16-bit operands. Also, we observed that all the types of inexact multipliers expect for the type 0 (standard truncated multiplier) listed in I are used in this *best* configuration (multiplier types 2, 4 and 6 are more dominant than other types). Another interesting observation in this *best* configuration is that all the multipliers in the output layer use 16-bit operands, suggesting that they were indeed more susceptible to errors, which was in line with the analysis of Section II.D.

Inexact Logic Minimization vs. Truncation. We also experimentally determined the impact of inexact multipliers over the more standard bit width truncation approach. In our exploration, the bit width truncation is applied by reducing the size of the operators, from 16 bits down to 8 bits.

In Table II, we showed the results of the *bit-width* configuration where the bit width of operators of the exact multipliers are being varied as freely as in the standard exploration. We can observe that varying the bit width of the operators provides only a small improvement over the baseline in terms of energy and area. On the other hand, introducing inexact multipliers provides more significant improvements, with only a small impact on accuracy.

C. Energy, Area and MSE of Inexact Hardware Neural Networks

In Figure 5, we compare the MSE of the baseline configuration against the MSE of the best configuration before and after training for that configuration; "before training" means that the synaptic weights of the baseline configuration are used. The MSE before training shows that inexact operators introduce a very significant loss of accuracy. This magnitude of accuracy loss would be unacceptable, but the training algorithm of neural networks can compensate for it. As a result, the MSE after training (0.20) is closer to the MSE of the baseline (0.14). This implies that the impact on accuracy of the inexact operators can be mostly overcome.

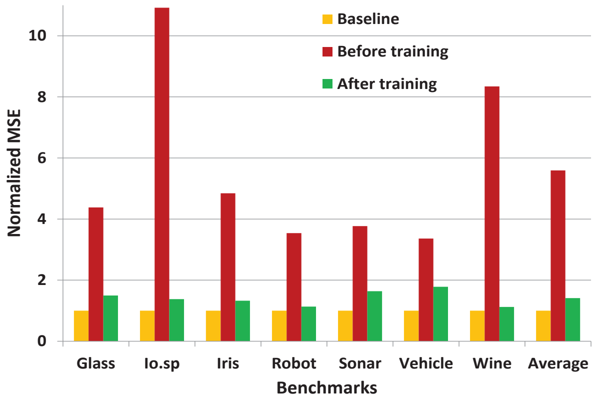
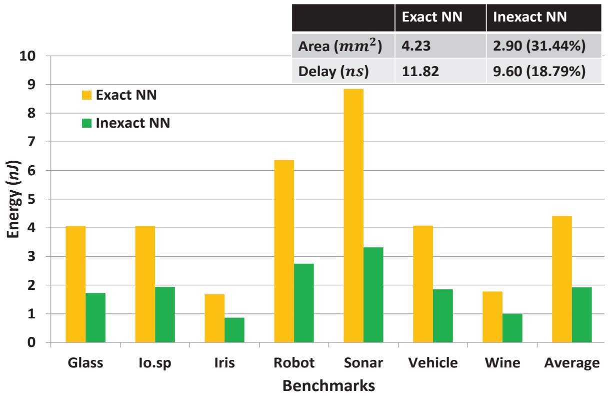


Fig. 5. Impact of inexact logic operators on error.



	Exact NN	Inexact NN
Area (mm^2)	4.23	2.90 (31.44%)
Delay (ns)	11.82	9.60 (18.79%)

Fig. 6. Energy, delay and area benefits of inexact NN vs. exact NN.

At the same time, the energy and area savings results achieved by using inexact architectures are very significant, as shown in Figure 6. For the 7 benchmarks, the energy savings is at least 43.91%, and up to 62.49%. Clock and registers correspond to 7%-24% of the total energy cost, while most of the energy is consumed by the combinational logic.

The area cost also drops from 4.23 mm^2 to 2.90 mm^2 implying an improvement of 31.44%. Finally, thanks to ILM, the critical path delay has been reduced from 11.82ns to 9.60ns. We note that 900 multiplications are performed in the hidden layer with 90 inputs and 10 neurons within these 9.60ns.

IV. Related Work

The early foundations of the principle of trading the hardware accuracy of the circuit for energy gains can be found in in [22] and since then, its application in many error-tolerant applications, in particular those whose output quality is judged by human sensory perception including vision or audition, has been exploited in a plethora of papers (for a partial summary, see [13]). Moving beyond datapaths to processors with control, the fact that the control in processors cannot be susceptible to errors proves to be a stringent limitation [8]. Mahdiani et al. study [19] considered the implementation of fuzzy logic and neural networks using imprecise adders and multipliers, but they sequentially execute all neurons on one single hardware neuron. Not only does this miss out on many of the performance and energy benefits of hardware neural networks in highly parallel implementations, but it creates the same susceptibility to control logic as in processors.

In this paper, we limit ourselves to previous work that is related only to inexact logic minimization that has been used here. These build on innovations at the logic- and architectural-level [6, 17, 2], which have emerged as a promising option providing *zero* (hardware) overhead. The readers are referred to earlier references for a discussion of other techniques with greater than zero overheads relying on voltage overscaling, for example. Also, we refer to some of the early examples where the notion of "significance" or value driven computation in conjunction with other techniques for energy minimization was done to realize inexact designs [11, 25].

Architectural approaches based on hardware neural networks are becoming increasingly popular [24] including considering concepts of using approximations at the computational level (in contrast to the datapath hardware level presented in this paper) and concerns of energy efficiency [9]. In the context of heterogeneous multi-cores, hardware neural networks are at the crossroad of multiple converging trends: the quest for both fault-tolerant and energy efficient accelerators, the need for accelerators with a broad application scope [5], and recent progress in machine-learning [15]. This convergence of trends has been acknowledged by companies such as IBM, whose hardware neural network accelerators [21] are now being developed.

V. Conclusions

In this article, we have proposed an approach for applying inexact computing to a broad set of applications by taking advantage of the inherent feature of hardware neural networks. Specifically, we take advantage of three distinct aspects of these neural networks: (a) many emerging high-performance applications can be competitively implemented using hardware neural networks, (b) these neural networks have much higher error tolerance owing to their learning/training ability, and (c) they can be realized as a circuit datapath using inexact logic operators. Using machine-learning benchmarks, we demonstrate that the proposed inexact neural network accelerator, implemented in 65nm technology, could achieve 43.91%-62.49% savings in energy consumption (with accompanying delay and area savings of 18.79% and 31.44% respectively) when compared to an existing baseline neural network implementation. These savings were achieved at only a slight cost in accuracy of output quantified through the MSE metric. Additionally, we also propose approaches to narrow down the huge search space of the inexact neural networks through a selective assignment of inexact logic operators and by identifying operators which yield higher energy savings and introducing relatively low error.

VI. Acknowledgment

This work is supported by the Intel ICRI-CI lab, a Google Faculty Research Award, ANR projects MHANN and NEMESIS, and the Inria joint team YOUHUA. And this work is supported by the Strategic Priority Research Program of the Chinese Academy of Sciences (under Grant XDA06010403), the National Natural Science Foundation of China (under Grants 61003064, 61100163, 61222204, 61303158, 61050002, 61173006, and 61133004), the National 973 Program of China (under Grant 20118034), and the National 863 Program of China (under Grants 2012AA012202).

References

[1] A. Lingamneni et al. Improving energy gains of inexact DSP hardware through reciprocative error compensation. *in the 50th Design Automation Conference*, pages 20:1–20:8, June 2013.

[2] A. Lingamneni et al. Synthesizing parsimonious inexact circuits through probabilistic design techniques. *in the ACM Transactions on Embedded Computing Systems*, 12(2s):93:1–93:26, May 2013.

[3] A. Asuncion and D. J. Newman. {UCI} Machine Learning Repository, 2007.

[4] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *International Conference on Parallel Architectures and Compilation Techniques*, New York, New York, USA, 2008. ACM Press.

[5] T. Chen, Y. Chen, M. Duranton, Q. Guo, A. Hashmi, M. Lipasti, A. Nere, S. Qiu, M. Sebag, and O. Temam. BenchNN: On the Broad Potential Application Scope of Hardware Neural Network Accelerators. In *International Symposium on Workload Characterization*, 2012.

[6] D Shin et al. Approximate logic synthesis for error tolerant applications. *in the proc. of DATE*, pages 957 – 960, 2010.

[7] P. Dubey. Recognition, Mining and Synthesis Moves Computers to the Era of Tera. *Technology@Intel Magazine*, 9(2):1–10, 2005.

[8] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture support for disciplined approximate programming. In T. Harris and M. L. Scott, editors, *ASPLOS*, pages 301–312. ACM, 2012.

[9] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Neural Acceleration for General-Purpose Approximate Programs. In *International Symposium on Microarchitecture*, 2012.

[10] G Venkatesh et al. Conservation Cores: Reducing the Energy of Mature Computations. *in proc. of the Architectural Support for Programming Languages and Operating Systems*, pages 205–218, March 2010.

[11] J. George, B. Marr, B. E. S. Akgul, and K. V. Palem. Probabilistic arithmetic and energy efficient embedded signal processing. In *proc. of IEEE/ACM CASES*, pages 158 – 168, 2006.

[12] S. Haykin. *Neural Networks*. Prentice Hall Intl, London, UK, 2nd edition, 1999.

[13] K Palem et al. What to do about the end of moore's law, probably! *in the 49th Annual Design Automation Conference*, pages 924–929, 2012.

[14] E. J. King and E. E. Swartzlander Jr. Data-dependent truncation scheme for parallel multipliers. In *Signals, Systems & Computers, 1997. Conference Record of the Thirty-First Asilomar Conference on*, volume 2, pages 1178–1182. IEEE, 1997.

[15] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *International Conference on Machine Learning*, pages 473–480, New York, New York, USA, 2007. ACM Press.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 11(86):2278–2324, 1998.

[17] Lingamneni et al. Energy parsimonious circuit design through probabilistic pruning. *in proc. of DATE*, pages 764–769, Mar 2011.

[18] Lingamneni et al. Parsimonious circuit design for error-tolerant applications through probabilistic logic minimization. *in the proc. of the PATMOS*, pages 204–213, 2011.

[19] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas. Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(4):850–862, Apr. 2010.

[20] M. Muller. Dark Silicon and the Internet. In *EE Times ”Designing with ARM” virtual conference*, 2010.

[21] P Merolla et al. A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm. In *IEEE Custom Integrated Circuits Conference*, pages 1–4. IEEE, Sept. 2011.

[22] K. V. Palem. Energy aware algorithm design via probabilistic computing: From algorithms and models to Moore's law and novel (semiconductor) devices. In *proc. of CASES*, pages 113 – 116, 2003.

[23] M. Stojilovic, D. Novo, L. Saranovac, P. Brisk, and P. Ienne. Selective flexibility: Breaking the rigidity of datapath merging. *in proc. of DATE*, pages 1543 –1548, march 2012.

[24] O. Temam. A Defect-Tolerant Accelerator for Emerging High-Performance Applications. In *International Symposium on Computer Architecture*, Portland, Oregon, 2012.

[25] Z. M. Kedem et al. Optimizing energy to minimize errors in dataflow graphs using approximate adders. In *in proc. of CASES*, pages 177–186, 2010.