Improving the Accuracy and Hardware Efficiency of Neural Networks Using Approximate Multipliers

Mohammad Saeed Ansari, *Student Member, IEEE*, Vojtech Mrazek, *Member, IEEE*, Bruce F. Cockburn, *Member, IEEE*, Lukas Sekanina, *Senior Member, IEEE*, Zdenek Vasicek, and Jie Han, *Senior Member, IEEE*

***Abstract*— Improving the accuracy of a neural network (NN) usually requires using larger hardware that consumes more energy. However, the error tolerance of NNs and their applications allow approximate computing techniques to be applied to reduce implementation costs. Given that multiplication is the most resource-intensive and power-hungry operation in NNs, more economical approximate multipliers (AMs) can significantly reduce hardware costs. In this article, we show that using AMs can also improve the NN accuracy by introducing noise. We consider two categories of AMs: 1) deliberately designed and 2) Cartesian genetic programing (CGP)-based AMs. The exact multipliers in two representative NNs, a multilayer perceptron (MLP) and a convolutional NN (CNN), are replaced with approximate designs to evaluate their effect on the classification accuracy of the Mixed National Institute of Standards and Technology (MNIST) and Street View House Numbers (SVHN) data sets, respectively. Interestingly, up to 0.63% improvement in the classification accuracy is achieved with reductions of 71.45% and 61.55% in the energy consumption and area, respectively. Finally, the features in an AM are identified that tend to make one design outperform others with respect to NN accuracy. Those features are then used to train a predictor that indicates how well an AM is likely to work in an NN.**

***Index Terms*— Approximate multipliers (AMs), Cartesian genetic programing (CGP), convolutional NN (CNN), multi-layer perceptron (MLP), neural networks (NNs).**

I. Introduction

THE increasing energy consumption of computer systems still remains a serious challenge in spite of advances in energy-efficient design techniques. Today's computing systems are increasingly used to process huge amounts of data, and they are also expected to present computationally demanding natural human interfaces. For example, pattern recognition, data mining, and neural network (NN)-based classifiers are especially required for computational resources. Approximate computing is an emerging design paradigm that can reduce the system cost without reducing the system effectiveness. It leverages the inherent error tolerance of many applications, such as machine learning, multimedia processing, pattern recognition, and computer vision, to allow some accuracy to be traded off to save hardware cost [1]. NNs are now recognized as providing the most effective solutions to many challenging pattern recognition and machine learning tasks such as image classification [2]. Due to their intrinsic error tolerance characteristics and high computation and implementation costs, there is increasing interest in using approximation in NNs. Approximation in the memories, where the synaptic weights are stored [3], approximation in the computation, such as using approximate multipliers (AMs) [4], [5] and approximation in neurons [6], [7], are all strategies that have already been reported in the literature.

Given that multipliers are the main bottleneck of NNs [8]–[10], this article focuses on the use of AMs in NNs. The work in [11] showed that using approximate adders (with reasonable area and power savings) has an unacceptable negative impact on the performance of NNs, so only exact adders are used in this article.

Several AMs have been proposed in the literature that decrease the hardware cost, while maintaining acceptably high accuracy. We divide the AMs into two main categories: 1) deliberately designed multipliers, which include designs that are obtained by making some changes in the truth table of the exact designs [12] and 2) Cartesian genetic programing (CGP)-based multipliers, which are designs that are generated automatically using the CGP heuristic algorithm [13]. Note that there are other classes of AMs that are based on analog mixed-signal processing [14], [15]. However, they are not considered in this article since our focus is on digital design that is more flexible in implementation than analog-/mixed-signal-based designs.

There is a tradeoff between the accuracy and the hardware cost, and there is no single best design for all applications. Thus, selecting the appropriate AM for any specific application is a complex question that typically requires careful consideration of multiple alternative designs. In this article, the objective is to find the AMs that improve the performance of an NN, i.e., by reducing the hardware cost while preserving

Manuscript received June 2, 2019; revised August 4, 2019; accepted September 3, 2019. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Project RES0018685 and Project RES0025211; and in part by the INTER-COST under project LTC18053. *(Corresponding author: Mohammad Saeed Ansari.)*

M. S. Ansari, B. F. Cockburn, and J. Han are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada (e-mail: ansari2@ualberta.ca; cockburn@ualberta.ca; jhan8@ualberta.ca).

V. Mrazek, L. Sekanina, and Z. Vasicek are with the IT4Innovations Centre of Excellence, Faculty of Information Technology, Brno University of Technology, 612 66 Brno, Czech Republic (e-mail: mrazek@fit.vutbr.cz; sekanina@fit.vutbr.cz; vasicek@fit.vutbr.cz).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2019.2940943

an acceptable output accuracy. To the best of our knowledge, this article is the first that attempts to find the critical features of an AM that make it superior to others for use in an NN.

Our benchmark multipliers, including 500 CGP-based AMs and 100 variants of deliberately designed multipliers, are evaluated for two standard NNs: a multi-layer perceptron (MLP) that classifies the MNIST data set [16] and a convolutional NN (CNN), LeNet-5 [17], that classifies the SVHN data set [18]. After each network is trained while using double-precision floating-point exact multipliers, the accurate multipliers are replaced with one approximate design (selected from the set of benchmark multipliers), and then five steps of retraining are performed. This process is repeated for each of the benchmark multipliers, resulting in 600 variants for each of the two considered NNs. The retraining is done for each AM only once. Then, the inference is performed to evaluate the accuracy. Since the simulations always start from the same point, i.e., we run the retraining steps on the pre-trained network (with exact multipliers), there is no randomness, and therefore the results will be consistent if the simulation is repeated.

The rest of this article is organized as follows. Section II specifies the considered networks and different types of AMs. Section III evaluates the considered multipliers from two perspectives: 1) application-independent metrics and 2) application-dependent metrics, and discusses the implications of the results. Section IV is devoted to feature selection and describes how the most critical features in an AM can be identified. Section V discusses the error and hardware characteristics of the AMs and recommends the five best AMs. For further performance analysis, these five multipliers are then used to implement an artificial neuron. Finally, Section VI summarizes and concludes this article.

II. Preliminaries

This section provides background information on the two benchmark NNs and describes the considered AMs.

A. Employed Neural Networks and Data Sets

MNIST (Mixed National Institute of Standards and Technology) is a data set of handwritten numbers that consists of a training set of 60 000 and a test set of 10 000 28×28 images and their labels [16]. We used an MLP network with 784 input neurons (one for each pixel of the monochrome image), 300 neurons in the hidden layer, and ten output neurons, whose outputs are interpreted as the probability of each of the classification into ten target classes (digits 0 to 9) [16]. This MLP uses the sigmoid activation function (AF). An AF introduces nonlinearity into the neuron's output and maps the resulting values onto either the interval $[-1, 1]$ or $[0, 1]$ [19]. Using the sigmoid AF, the neuron j in layer l, where $0 < l \leq l_{\max}$, computes an AF of the weighted sum of its inputs, $x_{j,l}$, as given by

$$x_{j,l} = \frac{1}{1 + e^{-\text{sum}_{j,l}}}$$
$$\text{sum}_{j,l} = \sum_{i=1}^{N} x_{i,l-1} \times w_{ij,l-1} \tag{1}$$

where N denotes the number of neurons in layer $l-1$ and $w_{ij,l-1}$ denotes the connection weight between the neuron i in layer $l-1$ and the neuron j in layer l [2].

SVHN is a data set of house digit images taken from Google Street View images [18]. The data set contains 73 257 images for training and 26 032 images for testing. Each digit is represented as a pair of a 32×32 RGB image and its label. We used LeNet-5 [17] to classify this data set. This CNN consists of two sets of convolutional and average pooling layers, followed by a third convolutional layer, and then a fully-connected layer. It also uses ReLU AF, which simply implements $\max(0, x)$. The convolutional and fully connected layers account for 98% of all the multiplications [13], therefore approximation is applied only to these layers. In order to reduce the complexity, we converted the original 32×32 RGB images to 32×32 grayscale images using the standard "luma" mapping [13]

$$Y = 0.299 \times R + 0.587 \times G + 0.114 \times B \tag{2}$$

where R, G, and B denote the intensities of red, green, and blue additive primaries, respectively.

To train an NN, the synaptic weights are initialized to random values. Then, the network is trained by using the standard backpropagation-based supervised learning method. During the training process, the weights are adjusted to reduce the error. Instead of starting the training with random initial weights, one can use the weights of a previously trained network. Initializing the weights in this way is referred to as using a pre-trained network [2]. Note that a pretrained network can be retrained and used to perform a different task on a different data set. Usually, only a few steps of retraining are required to fine-tune the pre-trained network.

B. Approximate Multipliers

Through comprehensive simulations, we confirmed that 8-bit multipliers are just wide enough to provide reasonable accuracies in NNs [10], [20]. Therefore, only 8-bit versions of the approximate multipliers were evaluated in this article.

1) Deliberately Designed Approximate Multipliers: Deliberately designed AMs are obtained by making carefully chosen simplifying changes in the truth table of the exact multiplier. In general, there are three ways of generating AMs [12], [21]: 1) approximation in generating the partial products, such as the under-designed multiplier (UDM) [22]; 2) approximation in the partial product tree, such as the broken-array multiplier (BAM) [23] and the error-tolerant multiplier (ETM) [24]; and 3) approximation in the accumulation of partial products, such as the inaccurate multiplier (ICM) [25], the approximate compressor-based multiplier (ACM) [26], the AM [27], and the truncated AM (TAM) [28]. The other type of deliberately designed AM that is considered in this article is the recently proposed alphabet set multiplier (ASM) [10].

Here, we briefly review the design of the deliberately designed AMs.

The UDM [22] is designed based on an approximate 2×2 multiplier. This approximate 2×2 multiplier produces 111_2, instead of 1001_2 to save one output bit when both of the inputs are 11_2.

The BAM [23] omits the carry-save adders for the least significant bits (LSBs) in an array multiplier in both the horizontal and vertical directions. In other words, it truncates the LSBs of the inputs to permit a smaller multiplier to be used for the remaining bits.

The ETM [24] divides the inputs into separate MSB and LSB parts that do not necessarily have equal widths. Every bit position in the LSB part is checked from left to right and if at least one of the two operands is 1, checking is stopped and all of the remaining bits from that position onward are set to 1. On the other hand, normal multiplication is performed for the MSB part.

The ICM [25] uses an approximate (4:2) counter to build AMs. The approximate 4-bit multiplier is then used to construct larger multipliers.

The ACM [26] is designed by using approximate 4:2 compressors. The two proposed approximate 4:2 compressors (AC1 and AC2) are used in a Dadda multiplier with four different schemes.

The AM [27] uses a novel approximate adder that generates a sum bit and an error bit. The error of the multiplier is then alleviated by using the error bits. The truncated version of the AM multiplier is called the TAM [28].

The ASM [10] decomposes the multiplicand into short bit sequences (alphabets) that are multiplied by the multiplier. Instead of multiplying the multiplier with the multiplicand, some lower-order multiples of the multiplier are first calculated (by shift and add operations) and then some of those multiples are added in the output stage of the ASM [10]. It should be noted that the ASM design was optimized for use in NNs, and so it is not directly comparable to the other AMs considered in this article when used in other applications.

Based on these main designs, variants were obtained by changing the configurable parameter in each design, forming a set of 100 deliberately designed approximate multipliers. For example, removing different carry-save adders from the BAM multiplier results in different designs; also, the widths of the MSB and LSB parts in the ETM multiplier can be varied to yield different multipliers.

2) CGP-Based Approximate Multipliers: Unlike the deliberately designed AMs, the CGP-based designs are generated automatically using CGP [13]. Although several heuristic approaches have been proposed in the literature for approximating a digital circuit, we used CGP, since it is intrinsically multi-objective and has been successfully used to generate other high-quality approximate circuits [29].

A candidate circuit in CGP is modeled as a 2-D array of programable nodes. The nodes in this problem are the 2-input Boolean functions, i.e., AND, OR, XOR, and others. The initial population P of CGP circuits includes several designs of exact multipliers and a few circuits that are generated by performing mutations on accurate designs. Single mutations (by randomly modifying the gate function, gate input connection, and/or primary output connections) are used to generate more candidate solutions. More details are provided in [13] and [29].

TABLE I
CONSIDERED FEATURES OF THE ERROR FUNCTION

Feature	**Description**
ER	Error rate
Var-ED	Variance of the ED values
Mean-ED	Mean value of the ED values
RMS-ED	Root mean square of the ED values
Var-RED	Variance of the RED values
Mean-RED	Mean value of the RED values
RMS-RED	Root mean square of the RED values
Var-AED	Variance of the AED values
Mean-AED	Mean value of the AED values

III. EVALUATION OF APPROXIMATE MULTIPLIERS IN NEURAL NETWORKS

This section considers both application-dependent and application-independent metrics to evaluate the effects of AMs in NNs.

A. Application-Independent Metrics

Application-independent metrics measure the design features that do not change from one application to another. Given that AMs are digital circuits, these metrics can be either error or hardware metrics. Error function metrics are required for the feature selection analysis.

The main four error metrics are the error rate (ER), the error distance (ED), the absolute ED (AED), and the relative ED (RED). We evaluated all 600 multiplier designs using the nine features extracted from these four main metrics, as given in Table I. All of the considered multipliers were implemented in MATLAB and simulated over their entire input space, i.e., for all $256 \times 256 = 65536$ combinations.

The definitions for most of these features are given in

$$\begin{aligned}
\mathrm{ED} &= E - A\\
\mathrm{RED} &= 1 - \frac{A}{E}\\
\mathrm{AED} &= |E - A|\\
\mathrm{RMS}_{\mathrm{ED}} &= \sqrt{\left(\frac{1}{N} \times \sum_{i=1}^{N}(A_i - E_i)^2\right)}\\
\mathrm{Var}_{\mathrm{ED}} &= \frac{1}{N} \times \sum_{i=1}^{N}\left(\mathrm{ED}_i - \frac{1}{N} \times \sum_{i=1}^{N}\mathrm{ED}_i\right)^2.
\end{aligned} \tag{3}$$

Those that are not given in (3) are evident from the description. Note that E and A in (3) refer to the exact and approximate multiplication results, respectively. Also, note that the mean-/ variance-related features in Table I are measured over the entire output domain of multipliers ($N = 65536$), i.e., $256 \times 256 = 65536$ cases for the employed eight-bit multipliers.

Note that the variance and the root mean square (RMS) are distinct metrics, as specified in (3). Specifically, the variance measures the spread of the data around the mean, while the RMS measures the spread of the data around the best fit. In the case of error metrics, the best possible fit is zero.

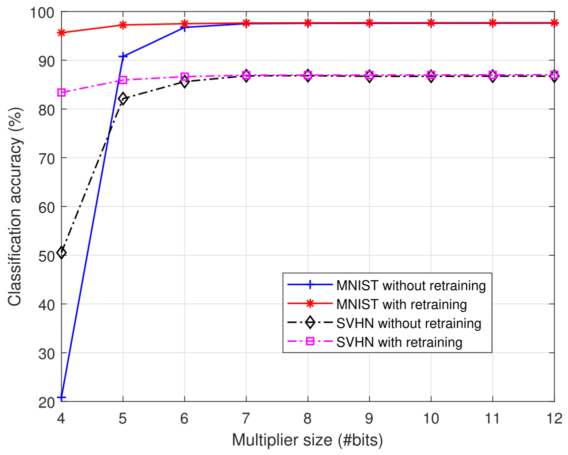


Fig. 1. Effects of multiplier size on classification accuracy.

We found that the majority of the 100 deliberately designed AMs either always overestimate or always underestimate the accurate product of the multiplication. This can be expected to cause problems when these multipliers are used in repetitive or iterative operations, such as matrix multiplications. In those cases, the errors do not cancel out and are instead accumulated. On the other hand, most of the CGP-generated AMs sometimes overestimate and sometimes also underestimate the product. This leads to some error cancellation and tends to make these multipliers better suited for use in NNs.

All of the multipliers were implemented in VHSIC Hardware Description Language (VHDL) and/or Verilog and synthesized using the Synopsys Design Compiler (DC) for the STMicroelectronics CMOS 28-nm process to obtain the most important hardware metrics: the power dissipation, the circuit area, and the critical path delay. These hardware metrics are useful for identifying the most hardware-efficient multiplier among those with similar error characteristics.

We also generated 500 AMs using the CGP algorithm. The Verilog, C, and MATLAB codes for all the designs and their error and hardware characteristics can be found in [30].

B. Application-Dependent Metrics

The classification accuracies of the MLP and LeNet-5 networks were evaluated over the MNIST and SVHN data sets, respectively. All 600 of the AM designs (100 deliberately designed and 500 CGP-based AMs) were employed in both NNs, and their classification accuracy was calculated.

The effect of multiplier size on the classification accuracy is shown in Fig. 1, where different-sized exact multipliers, ranging in width from 4 to 12 bits (including the sign bit), are shown. Note that the multiplication is performed on integer numbers. The original values in the range $[-1, 1]$ are mapped and rounded to the closest integers, with 1 being mapped to the maximum representable value, as determined by the size of the multiplier.

The results show that without performing the retraining steps, the 6-bit multiplier is the smallest design that is able to provide acceptable results. On the other hand, when retraining steps are considered (we performed five retraining steps), 4-bit designs can be used with only 2% degradation in classification accuracy compared to 8-bit designs. Note that the 8-bit designs were found to be only 0.04% less accurate than the 12-bit designs.

Interestingly, we observed that almost all of the AMs result in similar classification accuracies for the MNIST data set, regardless of the circuit design. This was expected, since MNIST is a relatively easy data set to classify. This bodes well for the use of cheaper, AM designs. The SVHN data set, however, shows a drop in classification accuracy more clearly than the MNIST data set when reduced-width multipliers are used. This might be due to the fact that SVHN data are harder to classify than the MNIST data.

C. Overfitting

An interesting finding from this article is the observation that a few AMs have slightly improved the classification accuracy over the exact multipliers. This is a potentially significant result, since it means we can use less hardware and yet get better results. We believe that overfitting in NNs may be the main reason for this interesting result.

Overfitting happens when the network is trained so much that it produces overly complex and unrealistic class boundaries when deciding whether to classify a data point into one class or another [31]. An overfitted network performs well on the training data, since it effectively memorizes the training examples, but it performs poorly on test data because it has not learned to generalize to a larger population of data values. Several solutions have been proposed in the literature to avoid overfitting such as dropout [31], weight decay [32], early stopping [33], and learning with noise [34]–[39].

Dropout techniques help to avoid overfitting by omitting neurons from an NN. More specifically, for each training case, a few neurons are selected and removed from the network, along with all their input and output connections [31]. Weight decay is another strategy to handle overfitting in which a weight-decay term is added to the objective function. This term reduces the magnitude of the trained weights and makes the network's output function smoother, and consequently improves the generalization (i.e., a well-generalized NN can more accurately classify unseen data from the same population as the learning data) and reduces the overfitting [32]. Early stopping approaches stop the training process as soon as a pre-defined threshold value for classification accuracy has been achieved [33].

Last but not least, the addition of noise to the synaptic weights of NNs has been found to be a low-overhead technique for improving the performance of an NN [35]. Murray and Edwards [37] report up to an 8% improvement in the classification accuracy by injecting stochastic noise into synaptic weights during the training phase. The noise injected into the synaptic weights in NNs can be modeled as either additive or multiplicative noise [38], [39], as defined in

$$\begin{aligned}\text{Additive noise}: W_{ij}^{*} &= W_{ij} + \delta_{ij}\\ \text{Multiplicative noise}: W_{ij}^{*} &= W_{ij}\delta_{ij}\end{aligned} \tag{4}$$

and both have been found to be beneficial.

In (4), δ_{ij} denotes the injected noise and W_{ij} denotes the noisy synaptic weight between the ith neuron in layer L and the jth neuron in layer $L+1$. The input of neuron j in layer $L+1$, denoted n_j, is calculated as

$$n_j = \sum_{i=1}^{N_L} x_i \times w_{ij} \tag{5}$$

where N_L is the number of neurons in layer L and x_i and w_{ij} denote a neuron's output and its connection weight to neuron j, respectively. If the exact multiplication in (5) is replaced with an approximate one, the approximate product for multiplicand a and multiplier b is given by

$$M(a,b) = a \times b + \Delta(a,b) \tag{6}$$

where the dither (error function) $\Delta(a,b)$ is the function that expresses the difference between the output of an exact multiplier and an AM. By combining (5) and (6), we obtain

$$\begin{aligned} n_j &= \sum_{i=1}^{N_L} x_i \times w_{ij} = \sum_{i=1}^{N_L} M(x_i, w_{ij}) \\ &\approx \xrightarrow{\text{approximate multipliers}} \sum_{i=1}^{N_L} M'(x_i, w_{ij}) \\ &= \sum_{i=1}^{N_L} \Big((x_i \times w_{ij}) + \Delta(x_i, w_{ij}) \Big) \\ &= \sum_{i=1}^{N_L} \left(x_i \times \left(w_{ij} + \frac{\Delta(x_i, w_{ij})}{x_i} \right) \right) = \sum_{i=1}^{N_L} x_i \times w_{ij}^{*}. \end{aligned} \tag{7}$$

Note that the noise term $\Delta(x_i, w_{ij})$ in (7) depends on the multiplier x_i, and is a different function for each individual design. Hence, we cannot compare the result in (7) to the definitions given in (4), since $\Delta(x_i, w_{ij})$ is an unknown function that changes for different multipliers. However, we hypothesize that the same argument that adding noise to the synaptic weights, as we did in (7), can sometimes help to avoid overfitting in NNs.

To provide experimental support for this hypothesis, we built an analytical AM, which is defined as

$$M'(a,b) = a \times b + \epsilon \tag{8}$$

where ϵ denotes the injected noise. We added Gaussian noise, since it is the most common choice in the literature [34]–[36]. We used this noise-corrupted exact multiplier in an MLP (784-300-10) and tested it over the MNIST data set. Fig. 2 shows how the accuracy is affected by increasing the noise levels. Note that the noise's mean and standard deviation in the noise-corrupted multiplier are the exact multiplication product (EMP) and a percentage of the EMP, respectively. This percentage is given by the term noise level in Fig. 2.

Since the added Gaussian noise is stochastic, we ran the simulations ten times and report the average results. The results in Fig. 2 confirmed the results in [34] and [39]: adding small amounts of noise can indeed improve the classification accuracy. However, as shown in Fig. 2, adding too much noise will degrade the classification accuracy. Note that the classification accuracies in Fig. 2 are normalized to the classification accuracy obtained by using exact multipliers.

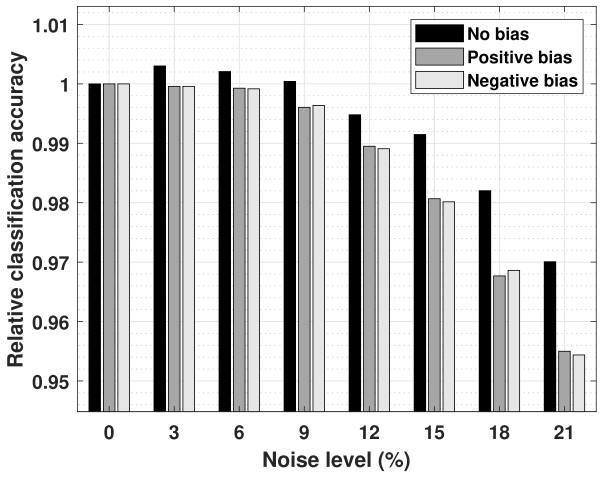


Fig. 2. MNIST classification accuracy, training, and testing with additive Gaussian noise.

Additionally, we injected Gaussian noise with positive and negative offsets in our accuracy analysis in Fig. 2 to show the negative effect of biased noise on the classification accuracy. For the biased noise, the errors are more likely to accumulate, and therefore the accuracy drops. The mean is changed to $1.1 \times$ EMP and $0.9 \times$ EMP to model the positive and negative offsets, respectively.

IV. Critical Features of Multipliers for Neural Networks

In Section III, we showed that adding noise to the multipliers can improve the accuracy of an NN. We also modeled the difference between an exact multiplier and an approximate one using the error function $\Delta(x_i, w_{ij})$ of the AM; see (7). In this section, we consider different multipliers to investigate what properties of the error function might make one design superior to others when employed in an NN.

As previously mentioned, the error function depends on the multiplier and is a different function for each individual design. An exact analysis of the error functions for different multipliers is impractical, and so instead we sought the relevant features of the error functions. Nine seemingly relevant features of the error function were identified, and are listed in Table I. In order to determine the most discriminative features of the error functions, i.e., the features that contribute the most to the performance of an AM in an NN, the nine features in Table I were applied to several statistical feature selection tools (as described next).

To be able to run feature selection algorithms, the multipliers were classified into two categories based on their performance in NNs. We defined a threshold accuracy, A_{th}, and classified the multipliers that produce higher accuracies than A_{th} into class 1, while the others into class 0. Since in the NN accuracy analysis some AMs produce slightly higher classification accuracies than exact multipliers when employed in NNs, it was convenient to choose $A_{\text{th}} = \text{ACC}_{\text{Exact}}$, which is the NN classification accuracy that is obtained when exact multipliers are employed in the network's structure. Note that the average noise level for class 1 AMs is 2.61%, which is close to the obtained noise level range in Fig. 2.

A. Feature Selection

Feature selection is a statistical way of removing less relevant features that are not as important to achieving accurate classification performance. There are many potential benefits to feature selection including facilitating data understanding and space dimensionality reduction [40], [41]. In this article, feature selection algorithms are used to select a subset of multipliers' error function features that are most useful for building a good predictor. This predictor anticipates the behavior of an AM in an NN.

Scikit-learn is a free machine learning tool that is widely used for feature selection [42]. It accepts an input data array and their corresponding labels to build an estimator that implements a fitting method. We used the three classifiers, recursive feature elimination (RFE) [43], mutual information (MI) [44], and Extra-Tree [45].

The RFE classifier iteratively prunes the least important features from the current set of features until the desired number of features is reached. The ith output of the RFE corresponds to the ranking position of the ith feature, such that the selected (i.e., the estimated best) features are assigned a rank of 1. Note that in RFE, the nested feature subsets contain complementary features and are not necessarily individually the most relevant features [43]. MI is another useful feature selection technique that relies on nonparametric methods based on entropy estimation from the K-nearest neighbor distances, as described in [44]. Each feature is assigned a score, where higher scores indicate more important features. Finally, the tree-based estimators can also be used to compute feature importance to discard less relevant features. Extra-Tree, an extremely randomized tree classifier, is a practical classifier that is widely used for feature selection [45]. Similar to MI, the ith output of this classifier identifies the importance of the ith feature, such that the higher the output score, the more important the feature is.

The results of each of the three aforementioned feature selection algorithms are provided in Table II. The results in Table II show that Var-ED is the most important feature according to all three classifiers. RMS-ED is another important metric, i.e., the most important metric according to RFE, the second-most critical feature in MI, and the third-most significant metric in Extra-Tree classifier. Our simulation results show that the average value of the Var-ED and RMS-ED features for class 0 multipliers are 20.21× and 6.42× greater than those of the class 1 AMs, respectively.

Other important features that have a good ranking in the three classifiers are MEAN-AED and VAR-AED. We also observed that the multipliers that produced better accuracies in an NN than the exact multiplier (class 1 multipliers) all have double-sided error functions. Thus, they overestimate the actual multiplication product for some input combinations and underestimate it for others. Having double-sided EDs seems to be a necessary, but not a sufficient condition for better accuracy.

TABLE II
RANKING OF ERROR FUNCTION FEATURES

Features	**Feature ranking**		
	RFE	**MI (score/ranking)**	**Extra-Tree (score/ranking)**
ER	5	0.2844 / 7	0.0518 / 9
Mean-ED	1	0.2124 / 8	0.0534 / 8
Var-ED	1	0.3884 / 1	0.2074 / 1
RMS-ED	1	0.3861 / 2	0.1383 / 3
Mean-RED	7	0.1623 / 9	0.0562 / 7
Var-RED	6	0.3309 / 5	0.0993 / 6
RMS-RED	4	0.3253 / 6	0.1235 / 5
Mean-AED	2	0.3655 / 3	0.1452 / 2
Var-AED	3	0.3424 / 4	0.1244 / 4

Given that class 1 AMs tend to have smaller Var-ED and RMS-ED values and the observation that double-sided errors are necessary for a good AM, the difference in the error magnitude should be small to meet the RMS-ED requirement i.e., having small RMS-ED values. Moreover, since the error should be double-sided to have a small variance, these errors should be distributed around zero.

B. Training the Classifier

Now, having found the most important features of the error function of an AM, we can use them to predict how well a given AM would work in an NN. In this section, we explain how to build a classifier that has the error features of an AM as inputs and predicts if it belongs to class 1 or class 0.

1) NN-Based Classifier: The error features of 500 randomly selected multipliers were used to train the NN-based classifier and those of the 100 remaining multipliers were used as the test samples to obtain the classification accuracy of the trained model. We designed a three-layer MLP with 20 neurons in the hidden layer and two neurons in the output layer (since we have two classes of multipliers). The number of neurons in the input layer equals the number of features that are considered for classification. The number of considered multiplier error features that were used as inputs to the NN-based classifier was varied from 1 up to 9 (for nine features, in total, see Table I). The resulting classification accuracies, plotted in Fig. 3, reflect how well the classifier classifies AMs into class 1 or class 0.

Note that when fewer than nine features are selected, the combination of features giving the highest accuracy is reported in Fig. 3. The combination of features is selected according to the results in Table II and is given in Table III.

To choose two features, for example, the candidate features are selected from the top-ranked ones in Table II: 1) Var-ED and Mean-AED (by Extra-Tree); 2) Var-ED and RMS-ED (by MI); and 3) Mean-ED, Var-ED, and RMS-ED (by RFE). For these four features (i.e., Mean-ED, Var-ED, RMS-ED, and Mean-AED), we consider all six possible combinations and report the results for the combination that gives the highest accuracy. Using the same process as in this example, the feature combinations for which the accuracy is maximized were found, and are provided in Table III.

TABLE III
FEATURE COMBINATIONS THAT GIVE THE HIGHEST MULTIPLIER CLASSIFICATION ACCURACY

Number of features	Selected combination
1	Var-ED
2	Var-ED, RMS-ED
3	Var-ED, RMS-ED, Mean-AED
4	Var-ED, RMS-ED, Mean-AED, Var-AED
5	Var-ED, RMS-ED, Mean-AED, Var-AED, RMS-RED
6	Var-ED, RMS-ED, Mean-AED, Var-AED, RMS-RED, Mean-ED
7	Var-ED, RMS-ED, Mean-AED, Var-AED, RMS-RED, Mean-ED, Var-RED
8	Var-ED, RMS-ED, Mean-AED, Var-AED, RMS-RED, Mean-ED, Var-RED, ER

As shown in Fig. 3, the highest classification accuracy is achieved when two features are used as inputs to the NN-based classifier, namely Var-ED and RMS-ED. Also, Fig. 3 shows that using more than two features does not necessarily result in a higher accuracy.

2) MATLAB Classification Learner Application: The MATLAB software environment provides a wide variety of specialized applications [46]. In particular, the classifier learner application, available in the apps gallery, allows us to train a model (classifier) that predicts if a multiplier falls into class 0 or class 1 when applied to an NN. This application provides the option of choosing a model type, i.e., decision trees, K-nearest neighbors, support vector machines (SVMs), and logistic classifiers among others. We considered all of these model types (with their default settings) to find the model that most accurately fits the classification problem. Similarly, 500 randomly selected multipliers were used to train the model and the 100 remaining multipliers were used as test samples to obtain the classification accuracy of the trained model.

Fig. 3 also shows the effect of the number of selected features on the accuracy of each of the three considered classifiers. Note that the SVM- and KNN-based classifiers achieve higher accuracies than the decision tree-based classifier. All three classifiers achieve better accuracies than the NN-based classifier.

Similar to the NN-based classifier, the classifier's accuracy for the combination of features that gives the highest accuracy is shown in Fig. 3 when fewer than nine features are selected. The highest classification accuracy for the SVM- and KNN-based classifiers is achieved when only two features are used as inputs to the classifier: i.e., Var-ED and RMS-ED. However, the decision tree-based classifier has the highest accuracy when only one feature, Var-ED, is considered.

C. Verifying the Classifiers

The trained SVM classifier was verified in Section III-B by using 100 AMs, where an accuracy of almost 86% was

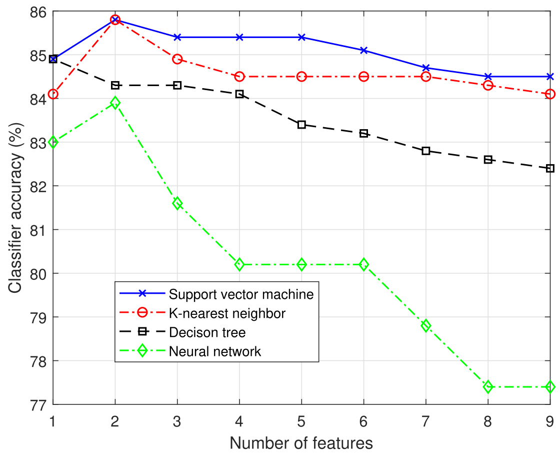


Fig. 3. Effect of the number of selected features on AM classifier accuracy.

achieved. In this section, the SVM classifier is used to predict the performance of 14 representative AMs in a different benchmark NN. The SVM classifier is selected since it shows the best performance compared to other classifiers, see Fig. 3.

Ideally, we would want to verify the classifier using all 600 AMs. However, the large number of multipliers in a deep NN benchmark and the large number of images in the data set would make the exhaustive experiment prohibitively time consuming. Therefore, in addition to the 100 previously considered multipliers, five multipliers were randomly selected from each class of AMs, plus the two multipliers that provided the best accuracy when used in an NN to classify the SVHN and MNIST data sets, and the two multipliers that had the worst accuracy for those same data sets. The SVM classifier was used to predict the behavior of each of these multipliers in a given NN benchmark. Then, these multipliers were used in the NN to verify the classifier's accuracy.

AlexNet is considered as the benchmark NN and is trained to classify the ImageNet data set [47]. AlexNet is a CNN with nine layers: an input layer, five convolution layers, and three fully connected layers [48]. Note that training a deep CNN over a big data set, such as ImageNet, would be very time consuming. Hence, we used the MATLAB pre-trained model and performed ten retraining steps (using the AMs) as an alternative to train the network from scratch.

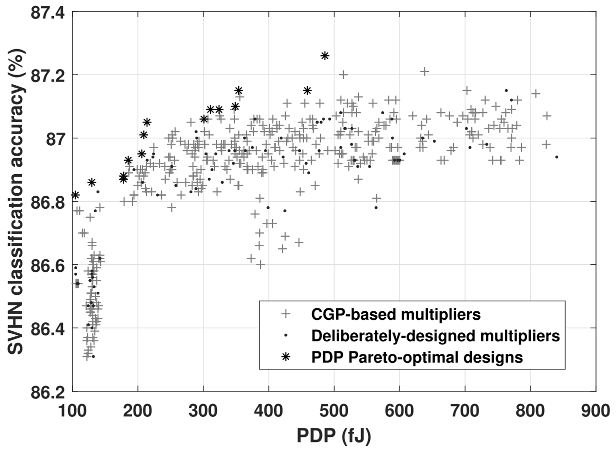
Table IV shows how the SVM classifier anticipates the performance of each of the 14 multipliers (i.e., the five randomly selected multipliers from each class of AMs and the four multipliers that provided the best and the worst accuracies when used in an NN to classify the SVHN and MNIST data sets) in AlexNet.

As shown in Fig. 3, none of the classifiers is 100% accurate. For instance, AlexNet implemented with the AM $M1$ has a worse accuracy than A_{th} (i.e., the accuracy of AlexNet implemented with exact multipliers) even though the multiplier is classified into class 1 (see Table IV). However, this misclassified multiplier produces an accuracy close to A_{th} and the difference (0.41%) is small.

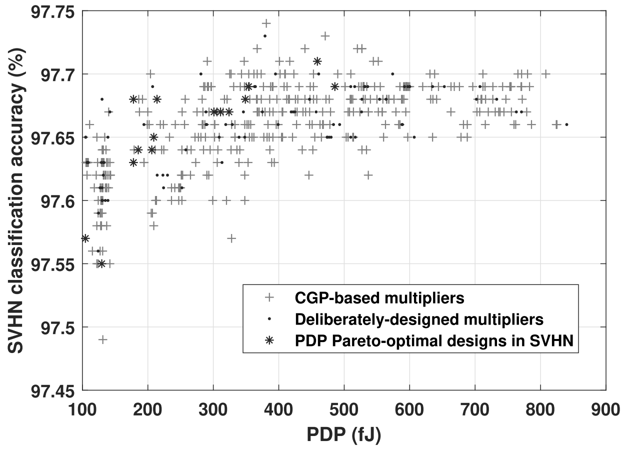
While some multipliers might perform well for one data set, they might not work well for other data sets. In other words,

TABLE IV
CLASSIFICATION ACCURACY OF ALEXNET ON THE IMAGENET LSVRC-2010 DATA SET

Class of multipliers	Multiplier	SVM classifier
class 1	M0: Randomly selected	✓
	M1: Randomly selected	✗
	M2: Randomly selected	✓
	M3: Randomly selected	✓
	M4: Randomly selected	✓
	Best for SVHN	✓
	Best for MNIST	✓
class 0	M5: Randomly selected	✓
	M6: Randomly selected	✓
	M7: Randomly selected	✓
	M8: Randomly selected	✓
	M9: Randomly selected	✓
	Worst for SVHN	✓
	Worst for MNIST	✓



(a)



(b)

Fig. 4. NN accuracy using the same AMs for different data sets. (a) Pareto-optimal design in PDP for the SVHN. (b) Behavior of SVHN Pareto-optimal multipliers for the MNIST.

the performance of a multiplier is application dependent. To illustrate this claim, we have plotted the Pareto-optimal designs in power-delay product (PDP) for the SVHN data set using all 600 AMs in Fig. 4(a).

Fig. 4(b) shows the performance of the Pareto-optimal multipliers in PDP for the SVHN data set for the MNIST data set. Note that a multiplier is considered to be PDP-Pareto optimal if there does not exist any other multiplier which improves the classification accuracy with the same PDP. It is clear from Fig. 4 that the Pareto-optimal designs for the two data sets are different.

V. Error and Hardware Analyses of Approximate Multipliers

This section analyzes the error and hardware characteristics of AMs. Based on this analysis, a few designs that have a superior performance in both considered data sets are identified and recommended.

A. Error Analysis

Fig. 5 compares class 0 and class 1 multipliers with respect to four important error features: Var-ED, RMS-ED, Mean-AED, and Var-AED. This plot shows how the class 1 and class 0 multipliers measure differently for the considered features. As shown in Fig. 5, class 1 multipliers generally have smaller Mean-AED, Var-ED, Var-AED, and RMS-ED values, when compared to class 0 multipliers. It also shows, in the zoomed-in insets, that some class 0 multipliers having smaller Var-AED, RMS-ED, Mean-AED, and/or Var-ED values than some class 1 multipliers is the reason why some multipliers are misclassified by the classifiers.

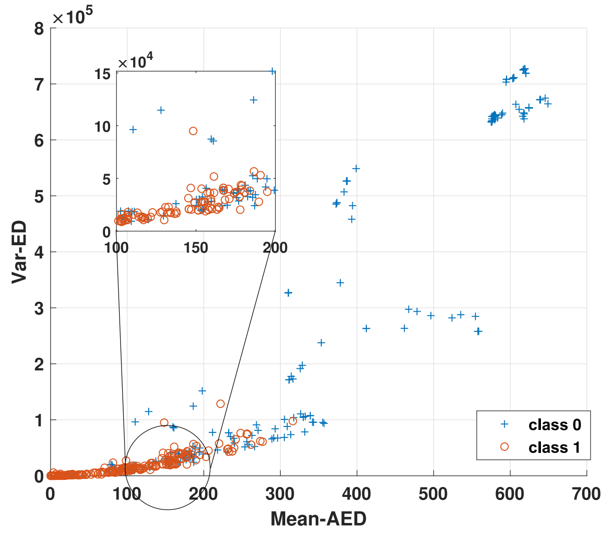
B. Hardware Analysis

To further understand the quality of AMs, we performed a hardware analysis. The main hardware metrics of a multiplier, i.e., power consumption, area, and critical path delay, and PDP, are considered in this analysis. Note that all of the considered multipliers in this article are pure combinational circuits for which the throughput is inversely proportional to the critical path delay.

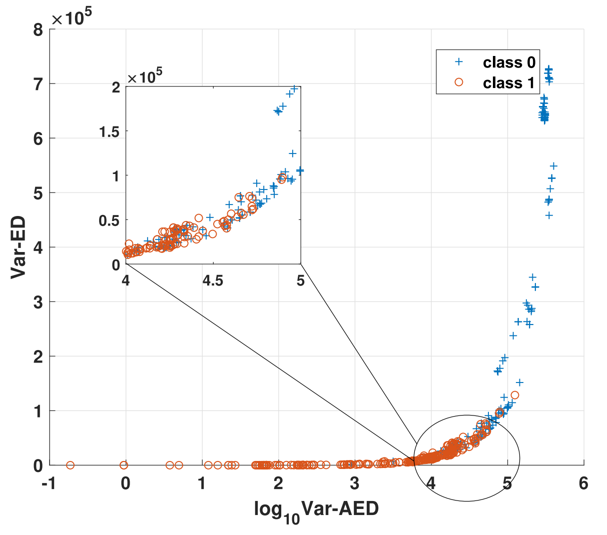
Fig. 6 shows two scatter plots that best distinguish the two classes of AMs, which are area versus delay (see Fig. 6(a)) and power consumption versus delay (see Fig. 6(b)). Note that only the results for the SVHN data set are shown as the results for the MNIST are almost the same.

As the results in Fig. 6 show, unlike for the error metrics, there is no clear general trend in the hardware metrics. However, the designs with small delay and power consumption are preferred for NN applications, as discussed next.

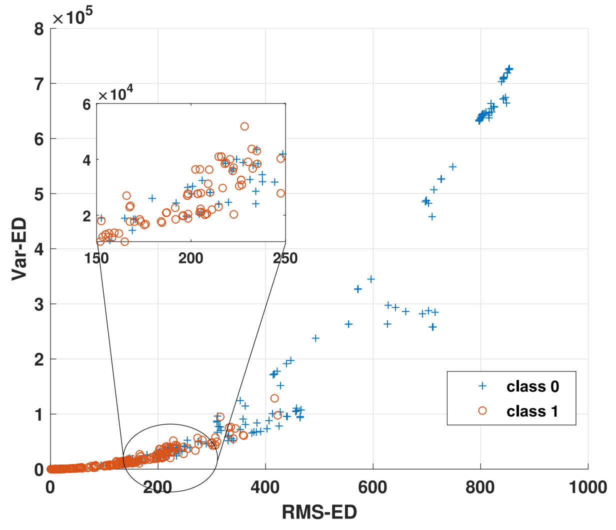
As AMs are obtained by simplifying the design of an exact multiplier, more aggressive approximations can be used to further reduce the hardware cost and energy consumption. As previously discussed, some multipliers have almost similar accuracies, while as shown in Fig. 4, they have different hardware measures. The main reasons are as follows: 1) the hardware cost of a digital circuit totally depends on how it is implemented in hardware; e.g., array and Wallace multipliers are both exact designs, and therefore they have the same classification accuracy. However, they have different hardware costs and 2) the classification accuracy of NNs is application dependent and it depends on the network type, the data set, the learning algorithm, and the number of training iterations.



(a)

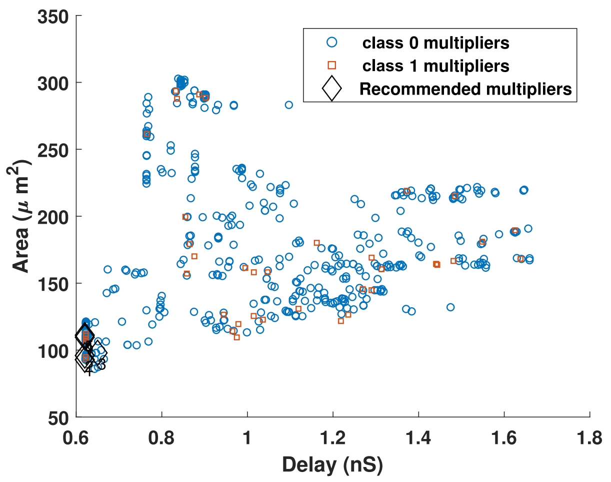


(b)

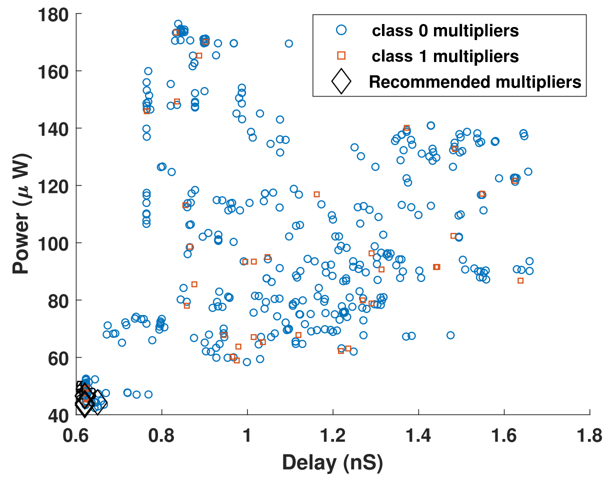


(c)

Fig. 5. Classification of class 0 and class 1 multipliers based on the most important features. (a) Var-ED versus mean-AED. (b)Var-ED versus log_{10}(Var-AED). (c) Var-ED versus RMS-ED.



(a)



(b)

Fig. 6. Hardware comparison between class 0 and class 1 AMs. (a) Area versus delay for class 1 and class 0 AMs. (b) Power versus delay for class 1 and class 0 AMs.

C. Recommended Approximate Multipliers

This section identifies a few AMs that exhibit superior performance for both considered data sets. We chose the five best AMs that produce better accuracies than exact multipliers when used in the two considered NNs: the MLP for the MNIST data set and LeNet-5 for the SVHN data set. Note that these five designs were selected and sorted based on their low PDP values.

Table V lists and Fig. 6 shows these multipliers. Their Verilog, C, and MATLAB descriptions can be found online from [30]. Table V also reports the main hardware characteristics of these designs, i.e., the area, power consumption, delay, and PDP. The results in Table V indicate that all five chosen AMs (which are all CGP-based AMs) consume less power (at least 73%) than the exact multiplier, while providing slightly higher accuracies (up to 0.18% or more) when they are used in NNs. Comparing the average area and the PDP shows significant savings in hardware cost (i.e., 65.20% and 81.74% less area and PDP, respectively) by replacing the exact multipliers with the approximate ones.

TABLE V
HARDWARE CHARACTERISTICS OF THE FIVE BEST AMs

Multiplier	**Area** (μm^2)	**Power** (μW)	**Delay** (nS)	**PDP** (fJ)
Exact	290.98	176.90	0.92	162.74
mul8-350	92.86	43.37	0.62	26.97
mul8-439	95.96	44.03	0.62	27.40
mul8-120	97.92	44.30	0.65	28.62
mul8-183	109.67	46.45	0.62	28.92
mul8-134	111.14	46.69	0.62	29.07

TABLE VI
ERROR CHARACTERISTICS OF THE FIVE BEST AMs

Multiplier	**ER**	**VAR-ED**	**RMS-ED**	**Accuracy (%)**	
				MNIST	**SVHN**
Exact	0	0	0	97.69	86.93
mul8-350	99.0	1.246e+4	123.0	97.70	**87.00**
mul8-439	97.8	4.500e+4	275.5	97.71	86.96
mul8-120	98.5	3.954e+4	217.3	97.70	**87.00**
mul8-183	97.2	1.334e+4	135.6	97.70	86.98
mul8-134	93.9	0.768e+4	111.1	**97.72**	86.95

TABLE VII
HARDWARE CHARACTERISTICS OF AN ARTIFICIAL NEURON IMPLEMENTED USING RECOMMENDED AMs

Multiplier used in the neuron	**Energy**(fJ)	**Area**(μm^2)
Exact	944.08	956.02
mul8-350	269.48	367.52
mul8-439	438.29	475.72
mul8-120	553.68	599.10
mul8-183	631.76	561.40
mul8-134	801.73	583.92

The accuracies of the five recommended multipliers when employed in the two NN workloads are reported in Table VI. Although not an important error feature, the ER is shown in Table VI, together with VAR-ED and RMS-ED, which are two critical error features for the performance of an AM in NNs. The results show that the five recommended multipliers all have small VAR-ED and RMS-ED values.

Hardware descriptions (in Verilog) of all of the CGP-based AMs can be found online in [30]. By using the Verilog code, one can easily obtain the truth table and/or the logic circuit for each design.

VAR-ED and RMS-ED, as the two most critical error features for the performance of an AM in NNs, are also given in Table VI. The results show that the five recommended multipliers all have small VAR-ED and RMS-ED values, which is consistent with the results in Fig. 5.

An artificial neuron was also implemented using the five recommended AMs to replace the exact ones. The implemented neuron has three inputs and an adder tree composed of two adders to accumulate the three multiplication products. This is a widely used technique for the performance analysis of multipliers in NNs [10].

The hardware characteristics of the implemented neuron are given in Table VII. The results show that the neurons constructed using the recommended multipliers can be up to 71.45% more energy-efficient than the neuron that uses the exact multiplier while being 61.55% smaller than it.

VI. CONCLUSION

This article described the evaluation of a large pool of AMs, which contained 100 deliberately designed and 500 CGP-based multipliers, for application in NNs. The exact multipliers in two benchmark networks, i.e., one MLP and one CNN (LeNet-5), were replaced after training with AMs to see how the classification accuracy is affected. The MLP and the CNN were employed to classify the MNIST and SVHN data sets, respectively. The classification accuracy was obtained experimentally for both data sets for all 600 AMs.

The features in an AM that tend to make it superior to others with respect to NN accuracy were identified and then used to build a predictor that forecasts how well an multiplier is likely to work in an NN. This predictor was verified by classifying 114 AMs based on their performance in LeNet-5 and AlexNet CNN for the SVHN and ImageNet data sets, respectively.

The major findings of this article are as follows.

1) Unlike most of the CGP-generated AMs, the majority of the 100 deliberately designed AMs either always overestimate or always underestimate the actual value of the multiplication. Hence, the errors in CGP-generated multipliers are more likely to cancel out, and therefore these multipliers are better suited for use in NNs.
2) It is not only possible, but can also be practical and more economical, to use AMs in the structure of NNs instead of exact multipliers.
3) NNs that use appropriate AMs can provide higher accuracies compared to NNs that use the same number of exact multipliers. This is a significant result since it shows that a better NN performance can be obtained with significantly lower hardware cost while using approximation.
4) It appears that using AMs adds small inaccuracies (i.e., approximation noise) to the synaptic weights and this noise helps to mitigate the overfitting problem, and thus improves the NN accuracy.
5) The most important features that make a design superior to others are the variance of the ED (Var-ED) and the RMS of the ED (RMS-ED).

Although the statistically most relevant and critical features of AMs are identified in this article, a statistically accurate predictor based on those features cannot guarantee that the best approximate design will be identified: ensuring the best choice of AM requires application-dependent experimentation.

REFERENCES

[1] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” in *Proc. 18th IEEE Eur. Test Symp. (ETS)*, May 2013, pp. 1–6.
[2] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.
[3] G. Srinivasan, P. Wijesinghe, S. S. Sarwar, A. Jaiswal, and K. Roy, “Significance driven hybrid 8T-6T SRAM for energy-efficient synaptic storage in artificial neural networks,” in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 151–156.
[4] T. Na and S. Mukhopadhyay, “Speeding up convolutional neural network training with dynamic precision scaling and flexible multiplier-accumulator,” in *Proc. Int. Symp. Low Power Electron. Design*, 2016, pp. 58–63.

[5] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," 2014, *arXiv:1412.7024*. [Online]. Available: https://arxiv.org/abs/1412.7024

[6] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "AxNN: Energy-efficient neuromorphic systems using approximate computing," in *Proc. Int. Symp. Low Power Electron. Design*, 2014, pp. 27–32.

[7] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "ApproxANN: An approximate computing framework for artificial neural network," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, 2015, pp. 701–706.

[8] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, "Fast neural networks without multipliers," *IEEE Trans. Neural Netw.*, vol. 4, no. 1, pp. 53–62, Jan. 1993.

[9] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," 2015, *arXiv:1510.03009*. [Online]. Available: https://arxiv.org/abs/1510.03009

[10] S. S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, and K. Roy, "Energy-efficient neural computing with approximate multipliers," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 14, no. 2, 2018, Art. no. 16.

[11] H. R. Mahdiani, M. H. S. Javadi, and S. M. Fakhraie, "Efficient utilization of imprecise computational blocks for hardware implementation of imprecision tolerant applications," *Microelectron. J.*, vol. 61, pp. 57–66, Mar. 2017.

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

[13] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *Proc. 35th Int. Conf. Comput.-Aided Design*, 2016, pp. 1–7.

[14] E. H. Lee and S. S. Wong, "Analysis and design of a passive switched-capacitor matrix multiplier for approximate computing," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 261–271, Jan. 2017.

[15] S. Gopal *et al.*, "A spatial multi-bit sub-1-V time-domain matrix multiplier interface for approximate computing in 65-nm CMOS," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 3, pp. 506–518, Sep. 2018.

[16] Y. LeCun, C. Cortes, and C. Burges. (2010). MNIST handwritten digit database. AT&T Labs. [Online]. Available: http://yann.lecun.com/exdb/mnist

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[18] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011, p. 5.

[19] R. J. Schalkoff, *Artificial Neural Networks*, vol. 1. New York, NY, USA: McGraw-Hill, 1997.

[20] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2017, pp. 1–12.

[21] M. S. Ansari, H. Jiang, B. F. Cockburn, and J. Han, "Low-power approximate multipliers using encoded partial products and approximate compressors," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 3, pp. 404–416, Sep. 2018.

[22] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *Proc. 24th Int. Conf. VLSI Design*, 2011, pp. 346–351.

[23] 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 Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 4, pp. 850–862, Apr. 2010.

[24] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *Proc. Int. Conf. Electron Devices Solid-State Circuits*, 2010, pp. 1–4.

[25] C.-H. Lin and I.-C. Lin, "High accuracy approximate multiplier with error correction," in *Proc. 31st Int. Conf. Comput. Design*, Oct. 2013, pp. 33–38.

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

[27] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, 2014, pp. 1–4.

[28] H. Jiang, J. Han, F. Qiao, and F. Lombardi, "Approximate radix-8 booth multipliers for low-power and high-performance operation," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2638–2644, Aug. 2016.

[29] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Trans. Evol. Comput.*, vol. 19, no. 3, pp. 432–444, Jun. 2015.

[30] (2016). *EvoApprox8b—Approximate Adders and Multipliers Library*. [Online]. Available: http://www.fit.vutbr.cz/research/groups/ehw/approxlib/

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[32] C. S. Leung, H.-J. Wang, and J. Sum, "On the selection of weight decay parameter for faulty networks," *IEEE Trans. Neural Netw.*, vol. 21, no. 8, pp. 1232–1244, Aug. 2010.

[33] Y. Shao, G. N. Taff, and S. J. Walsh, "Comparison of early stopping criteria for neural-network-based subpixel classification," *IEEE Geosci. Remote Sens. Lett.*, vol. 8, no. 1, pp. 113–117, Jan. 2011.

[34] Y. Luo and F. Yang. (2014). *Deep Learning With Noise*. [Online]. Available: hp://www.andrew.cmu.edu/user/fanyang1/deep-learning-with-noise.pdf

[35] N. Nagabushan, N. Satish, and S. Raghuram, "Effect of injected noise in deep neural networks," in *Proc. Int. Conf. Comput. Intell. Comput. Res.*, 2016, pp. 1–5.

[36] T. He, Y. Zhang, J. Droppo, and K. Yu, "On training bi-directional neural network language model with noise contrastive estimation," in *Proc. 10th Int. Symp. Chin. Spoken Lang. Process.*, 2016, pp. 1–5.

[37] A. F. Murray and P. J. Edwards, "Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training," *IEEE Trans. Neural Netw.*, vol. 5, no. 5, pp. 792–802, Sep. 1994.

[38] J. Sum, C.-S. Leung, and K. Ho, "Convergence analyses on on-line weight noise injection-based training algorithms for MLPs," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 11, pp. 1827–1840, Nov. 2012.

[39] K. Ho, C.-S. Leung, and J. Sum, "Objective functions of online weight noise injection training algorithms for MLPs," *IEEE Trans. Neural Netw.*, vol. 22, no. 2, pp. 317–323, Feb. 2011.

[40] I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror, "Result analysis of the NIPS 2003 feature selection challenge," in *Proc. Adv. Neural Inf. Process. Syst.*, 2005, pp. 545–552.

[41] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, Jan. 2003.

[42] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[43] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Mach. Learn.*, vol. 46, nos. 1–3, pp. 389–422, 2002.

[44] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 69, no. 6, 2004, Art. no. 066138.

[45] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, no. 1, pp. 3–42, 2006.

[46] MathWorks. *MATLAB Classification Learner App*. Accessed: Oct. 1, 2019. [Online]. Available: https://www.mathworks.com/help/stats/classificationlearner-app.html

[47] (2015). *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*. [Online]. Available: http://www.image-net.org/challenges/LSVRC/

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

Mohammad Saeed Ansari (S'16) received the B.Sc. and M.Sc. degrees in electrical and electronic engineering from Iran University of Science and Technology, Tehran, Iran, in 2013 and 2015, respectively. He is currently working toward the Ph.D. degree in electrical and computer engineering at the University of Alberta, Edmonton, AB, Canada.

His current research interests include approximate computing, design of computing hardware for emerging machine learning applications, multilayer perceptrons (MLPs), convolutional NNs (CNNs) in particular, and reliability and fault tolerance.

Vojtech Mrazek (M'18) received the Ing. and Ph.D. degrees in information technology from the Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic, in 2014 and 2018, respectively.

He is currently a Researcher with the Evolvable Hardware Group, Faculty of Information Technology, Brno University of Technology. He is also a Visiting Postdoctoral Researcher with the Department of Informatics, Institute of Computer Engineering, Technische Universität Wien (TU Wien), Vienna, Austria. He has authored or coauthored over 30 conference/journal papers focused on approximate computing and evolvable hardware. His current research interests include approximate computing, genetic programming, and machine learning.

Dr. Mrazek received several awards for his research in approximate computing, including the Joseph Fourier Award for research in computer science and engineering in 2018.

Bruce F. Cockburn (S'86–M'90) received the B.Sc. degree in engineering physics from Queen's University, Kingston, ON, Canada, in 1981, and the M.Math. and Ph.D. degrees in computer science from the University of Waterloo, Waterloo, ON, Canada, in 1985 and 1990, respectively.

From 1981 to 1983, he was a Test Engineer and a Software Designer with Mitel Corporation, Kanata, ON, Canada. He was a Sabbatical Visitor with Agilent Technologies, Inc., Santa Clara, CA, USA, and The University of British Columbia, Vancouver, BC, Canada, in 2001 and from 2014 to 2015, respectively. He is currently a Professor with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. His current research interests include the testing and verification of integrated circuits, FPGA-based hardware accelerators, parallel computing, stochastic and approximate computing, and bioinformatics.

Lukas Sekanina (M'02–SM'12) received the Ing. and Ph.D. degrees from Brno University of Technology, Brno, Czech Republic, in 1999 and 2002, respectively.

He was a Visiting Professor with Pennsylvania State University, Erie, PA, USA, in 2001, and the Centro de Eléctronica Industrial (CEI), Universidad Politécnia de Madrid (UPM), Madrid, Spain, in 2012, and a Visiting Researcher with the Department of Informatics, University of Oslo, Oslo, Norway, in 2001. He is currently a Full Professor and the Head of the Department of Computer Systems, Faculty of Information Technology, Brno University of Technology.

Dr. Sekanina received the Fulbright Scholarship to work with the NASA Jet Propulsion Laboratory, Caltech, in 2004. He has served as an Associate Editor for the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION from 2011 to 2014, the *Genetic Programming and Evolvable Machines Journal*, and the *International Journal of Innovative Computing and Applications*.

Zdenek Vasicek received the Ing. and Ph.D. degrees in electrical engineering and computer science from the Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic, in 2006 and 2012, respectively.

He is currently an Associate Professor with the Faculty of Information Technology, Brno University of Technology. His current research interests include evolutionary design and optimization of complex digital circuits and systems.

Dr. Vasicek received the Silver and Gold medals at HUMIES, in 2011 and 2015, respectively.

Jie Han (S'02–M'05–SM'16) received the B.Sc. degree in electronic engineering from Tsinghua University, Beijing, China, in 1999, and the Ph.D. degree from Delft University of Technology, Delft, The Netherlands, in 2004.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. His current research interests include approximate computing, stochastic computing, reliability and fault tolerance, nanoelectronic circuits and systems, and novel computational models for nanoscale and biological applications.

Dr. Han was a recipient of the Best Paper Award at the International Symposium on Nanoscale Architectures (NanoArch 2015) and Best Paper Nominations at the 25th Great Lakes Symposium on VLSI (GLSVLSI 2015), NanoArch 2016, and the 19th International Symposium on Quality Electronic Design (ISQED 2018). He served as the General Chair for GLSVLSI 2017 and the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT 2013). He served as the Technical Program Committee Chair for GLSVLSI 2016 and DFT 2012. He is currently an Associate Editor of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING (TETC), the IEEE TRANSACTIONS ON NANOTECHNOLOGY, and *Microelectronics Reliability* (Elsevier Journal).