An Improved Logarithmic Multiplier for Energy-Efficient Neural Computing

Mohammad Saeed Ansari, Bruce F. Cockburn, Jie Han

Abstract—Multiplication is the most resource-hungry operation in neural networks (NNs). Logarithmic multipliers (LMs) simplify multiplication to shift and addition operations and thus reduce the energy consumption. Since implementing the logarithm in a compact circuit often introduces approximation, some accuracy loss is inevitable in LMs. However, this inaccuracy accords with the inherent error tolerance of NNs and their associated applications. This article proposes an improved logarithmic multiplier (ILM) that, unlike existing designs, rounds both inputs to their nearest powers of two by using a proposed nearest-one detector (NOD) circuit. Considering that the output of the NOD uses a one-hot representation, some entries in the truth table of a conventional adder cannot occur. Hence, a compact adder is designed for the reduced truth table. The 8×8 ILM achieves up to 17.48% saving in power consumption compared to a recent LM in the literature while being almost 8% more accurate. Moreover, the evaluation of the ILM for two benchmark NN workloads shows up to 21.85% reduction in energy consumption compared to the NNs implemented with other LMs. Interestingly, using the ILM increases the classification accuracy of the considered NNs by up to 1.4% compared to a NN implementation that uses exact multipliers.

Index Terms—neural network, logarithmic multiplier, adder, energy efficiency, error-tolerant.

1 Introduction

Neural networks (NNs) are a class of computing architectures that are inspired by how a biological brain processes information. An artificial NN is composed of numerous highly interconnected processing elements, called neurons [1]. More complex deep learning networks (DLNs) are recent variants of NNs that have been shown to have especially strong performance. With their remarkable ability to create classification models from a large set of complex data items [2], NNs and especially DLNs, are being used successfully for a wide variety of real-world applications.

Due to the massive computation workloads of NNs, developing improved hardware implementations of DLNs is an important topic. The conventional architecture suffers from relatively high energy consumption and area overhead [3]. Several solutions that improve the hardware efficiency of the DLNs have been developed for different abstraction levels, ranging from architecture-level [4], [5] to transistor-level techniques [6], [7].

Fortunately, NNs and their associated applications are inherently error-tolerant. Typically, there is a range of acceptable, good-enough results rather than a unique required result. This flexibility allows designers to exploit approximation to increase computational efficiency. Several recent studies have exploited the use of approximation in NNs at the hardware level. One particular technique is to use reduced precision (e.g., use 8-bit fixed-point numbers rather than 32-bit floating-point numbers) [8], [9], [10]. Since multiplying the inputs with their corresponding weights is the most resource-hungry operation in NNs [11], reducing the bit precision could allow a custom processor to complete a larger number of neuron evaluations per second. Reducing the precision also reduces the area of the processor and the memory size.

Given that multipliers are the main bottleneck of NNs, several approximate multipliers have been proposed for the implementation of NNs [12]. Cartesian Genetic Programming (CGP) is used in [13] to generate approximate multipliers for NNs. This method requires a time-consuming search to evolve and select approximate multipliers that have favorable trade-offs between accuracy and hardware cost. The so-called alphabet-set multipliers are proposed in [11] with reduced alphabets that are used to approximate the sum of partial products. However, these multipliers do not support all the input combinations and they impose restrictions on the weights. Both methods in [13] and [11] incur some accuracy loss, and retraining is necessary to approach the accuracy obtained by using exact multipliers.

Following a different approach, we propose a hardware-efficient logarithmic multiplier (LM) for use in NNs. LMs convert multiplication into only shift and addition operations, thus allowing it to be done faster and with smaller area and power consumption [14], [15], [16]. LMs are inherently approximate designs due to: (1) a restricted bit-width precision and (2) permitted inaccuracy in computing the function $log_2(x)$ [14]. Using either piece-wise linear approximations over a finely subdivided input domain and/or iterative techniques can compensate for the accuracy loss in computing $log_2(x)$ [17].

Existing approaches for the hardware implementation of logarithmic conversion can be classified into three main categories [18]: (1) the digit-recurrence method, (2) look-up table (LUT)-based methods, and (3) piece-wise polynomial approximations. The digit-recurrence method is slow and can have convergence problems while LUT-based methods suffer from relatively large hardware and memory usage [18]. On the other hand, piece-wise polynomial approxima-

This work was supported financially by the Natural Sciences and Engineering Research Council of Canada (NSERC) (Project Nos. RES0018685 and RES0025211).

The authors are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, T6G 1H9, Canada (e-mail: ansari2, cockburn, jhan8@ualberta.ca).

tion is often the most efficient solution [16], [18], [19], [20].

An early piece-wise polynomial approximation was proposed by Mitchell [21]. It rounds a number N to the highest power of two, 2^k, such that $2^k \leq N$. Unfortunately, Mitchell's method can have relatively large approximation errors [18]. Several Mitchell-based multipliers have been proposed to improve the accuracy. They usually divide the power-of-two intervals into more than one region and then apply piece-wise linear approximation within each region. The published designs differ in the number of regions and in the piece-wise linear approximation functions used in each region, for example [16].

The approximation error in the reported Mitchell-based multipliers is always negative (i.e., the magnitude of the approximate product is smaller than the exact product) [17]. This systematic error causes problems in repetitive or iterative operations, such as matrix multiplications, since the errors do not cancel and are accumulated. Unlike the previous designs, the proposed improved LM (ILM) has a double-sided error distribution and benefits from a novel exact adder structure for further hardware and energy reductions. This is due to the design of a new nearest-one detector (NOD) that rounds each input operand up or down to its nearest power of two. The output of the NOD uses a one-hot representation, and this property is leveraged to simplify the structure of a conventional full adder (FA). Specifically, not all of the entries in the truth table of a conventional adder can occur when adding the base-2 logarithms of the input operands, and so the truth table can be reduced, which leads to a compact and yet still accurate adder.

The proposed ILM and its variants (i.e., ILMs with a few approximation bits) are evaluated for two well-known benchmark NNs: a multi-layer perceptron (MLP) that classifies the Mixed National Institute of Standards and Technology (MNIST) dataset [22] and a convolutional neural network (CNN), Alexnet [23], that classifies the Canadian Institute For Advanced Research (CIFAR)-10 dataset [24]. The accurate multipliers in both networks are replaced with LMs in each design during inference in order to comparatively evaluate different LMs. The training is done by using exact multipliers since a NN is usually trained once (or rarely) but is then evaluated many times thereafter.

Some of our preliminary work was published in [25]. In this article, the NOD design in [25] is simplified for use in NNs. In particular, we observed that large weights are unlikely to appear in trained NNs and that removing them would not significantly influence the performance of NNs (this is elaborated upon in Section 2). Hence, the NOD circuit in [25] is simplified further so that it produces an n-bit output (instead of $n+1$ bits in [25]) for an n-bit input. Thus the particular properties of NNs are exploited for the proposed ILM to achieve a higher energy efficiency. Moreover, as mentioned above, a novel exact adder is devised that modifies the truth table of a FA by removing the entries which cannot occur. Consequently, a simpler structure than that of the conventional adder can be used.

The remainder of this article is organized as follows: Section 2 is a brief introduction to NNs. Section 3 reviews published LMs. The proposed approximation approach and the hardware implementation of the ILM are discussed in Section 4. Section 5 evaluates the error and hardware performance of the proposed and state-of-the-art LMs. Two benchmark NNs are considered in Section 6 to provide an objective evaluation of the accuracy of the proposed ILM design. Moreover, the hardware characteristics of the considered NNs using LMs are discussed in Section 6. Finally, Section 7 concludes the article.

2 Basics of Neural Networks

Neural networks process information in an entirely different way than a conventional (von Neumann) computer [26]. During training, weights are adjusted in the neurons of a NN to allow the NN to perform certain computations (e.g., pattern recognition and classification) [27]. The neurons in a NN are arranged in several layers including an input layer, a variable number of hidden layer(s) (of the same or different types) followed by an output layer.

As experience is being gained in machine learning tasks, diverse types of hidden NN layers are being proposed. The authors in [28] employed convolutional layers that function as local filters over the data from the previous layers. Other common types of hidden layers are the average and max pooling layers that are used for weighted sub-sampling [29]. More recently, several application-specific layers have been proposed for image classification [23], segmentation [30] and speech processing [31].

2.1 Artificial neuron

Neurons are the main processing units of NNs that compute a weighted sum of their inputs and send the result through an activation function (AF). The AF introduces non-linearity into a neuron's behavior and maps the resulting output values into either the interval (-1, 1) or (0, 1) [13]. The AF can be either a hard-limiting (e.g., a step function) or a soft-limiting function (e.g., a sigmoid function) [11].

Fig. 1 shows the structure of an artificial neuron. A neuron has $n \geq 2$ inputs (depending on the network structure) and one output. Each input x_i is multiplied by its corresponding synaptic weight $w_i, i = 0, 1, ..., n$. An adder tree is then used to sum up the products. The resulting sum is then input to the AF. An external bias b is often added to increase or lower the input value of the AF [27].

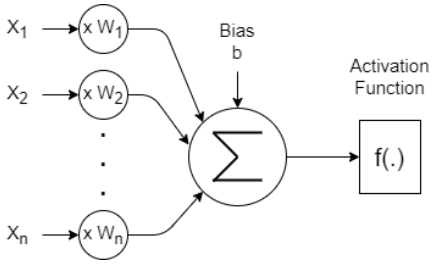


Fig. 1: Model of an artificial neuron.

2.2 Feed-forward neural networks

The two major operating modes for NNs are training and inference. The training process is usually performed infrequently and off-line and, therefore, its energy consumption

is less of a concern [11]. The inference process, on the other hand, is done frequently. Although it is less computation-intensive than the training process, inference still requires significant computation for large networks. Fig. 2 shows a simple feed-forward NN with n, k, and m neurons in the input, hidden, and output layers, respectively.

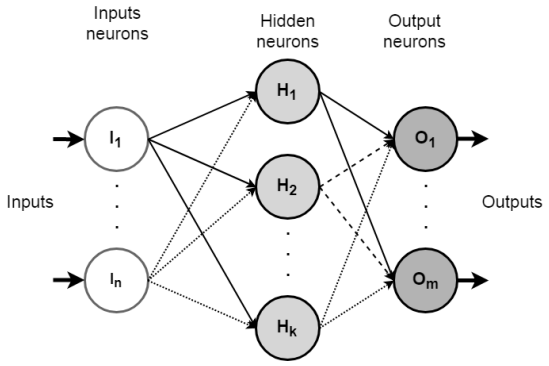


Fig. 2: Structure of a feed-forward NN.

2.3 Synaptic weights in neural networks

The n multiplications of $x_i \times w_i$ in Fig. 1, where $1 \leq i \leq n$, are the main bottleneck in the performance of NNs. Hence, it is helpful to have insight into the synaptic weights when designing multipliers for NNs. Here are the three main observations that have been found to be helpful:

- Past research has investigated the effects of reduced precision on the performance of NNs and has found that the precision can be safely reduced without significant negative impact on the accuracy of NNs [8], [9], [10].
- The authors in [32] showed that the trained weights in NNs are mostly centered near zero. Hence, even after up-scaling, as will be discussed in Section 5, the chances of having large weights, i.e. weights at both the positive and negative ends of the scaled spectrum, are small.
- Large synaptic weights are not desirable in NNs and it is common to use weight decay techniques to reduce them. During weight decay, after each update, the weights are multiplied by a factor slightly less than 1 to prevent them from growing too large [33].

Consequently, large weights are unlikely to appear in trained NNs and limiting them should not significantly influence the performance of NNs. We exploit this insight in the design of the proposed multiplier, more specifically in the design of the NOD circuit, as discussed in Section 4.2.

3 Review of Logarithmic Multipliers

Let $Z = z_{n-1}z_{n-2}...z_1z_0$ be the n-bit binary representation of a positive integer N. Without loss of generality, let z_k, where $k < n$, be the most significant '1' in Z. Hence, N can be represented as:

$$N = 2^k(1 + x), \tag{1}$$

where $0 \leq x < 1$.

Let A and B be the multiplicand and the multiplier, respectively. Following (1), once the base-2 logarithms of input operands $A = 2^{k_1}(1 + x_1)$ and $B = 2^{k_2}(1 + x_2)$ are calculated as:

$$log_2A = k_1 + log_2(1 + x_1), \tag{2}$$

$$log_2B = k_2 + log_2(1 + x_2), \tag{3}$$

their product $A \times B$ is given by:

$$A \times B = 2^{k_1+k_2}(1 + x_1)(1 + x_2). \tag{4}$$

Depending on the computation process, different values for log_2A and log_2B and, consequently, different approximate products can be obtained. For example, the Mitchell algorithm for an LM uses the following approximation:

$$A \times B \approx \begin{cases} 2^{k_1+k_2}(1 + x_1 + x_2), & x_1 + x_2 < 1, \\ 2^{k_1+k_2+1}(x_1 + x_2), & x_1 + x_2 \geq 1. \end{cases} \tag{5}$$

It was found in [34] that the average error for given k_1, k_2, $x_1 \in [0, 1)$, and $x_2 \in [0, 1)$, for the Mitchell algorithm can be expressed as:

$$E_A = -0.08333 \times 2^{k_1+k_2}. \tag{6}$$

Hence, an error correction term c can be added to the Mitchell algorithm to reduce the average error [34]:

$$A \times B \approx \begin{cases} 2^{k_1+k_2}(1 + x_1 + x_2 + c), & x_1 + x_2 < 1, \\ 2^{k_1+k_2+1}(x_1 + x_2 + \frac{c}{2}), & x_1 + x_2 \geq 1. \end{cases} \tag{7}$$

However, this modified technique increases the area and power consumption compared to the Mitchell algorithm [34].

The approximate LM in [17] uses a so-called set-to-one adder (ALM-SOA). The set-one-adder (SOA) with k approximation bits (SOA-k) assigns 1 to the k LSBs and, therefore, the actual product is overestimated. Given the fact that the Mitchell multiplier always underestimates the actual product, using a SOA can compensate for the accuracy loss in the multiplier. This technique is used in [17] to improve the accuracy of the Mitchell multiplier with less hardware cost.

A low-power implementation of the Mitchell multiplier is proposed in [35]. As extended work, a parameter w is introduced in [36] for a customizable LM in which only the most significant w bits of the operands are taken into account. Subsequently, truncation is performed after the approximate logarithms of the operands are calculated (by using the Mitchell algorithm). This differs from truncating the input operands before computing their logarithm. Due to the truncation, this multiplier is more hardware-efficient than the Mitchell multiplier. However, it is less accurate than it in terms of both the mean and worst-case errors.

4 Improved Logarithmic Multipliers

Here we propose a method to approximate log_2N which, unlike the existing approaches, has a double-sided error distribution and can be used as a more accurate baseline design instead of the Mitchell approach. The existing techniques in the literature for improving the accuracy of the Mitchell method are also applicable to the proposed method.

4.1 Proposed approximation approach

In addition to the expression (1), any positive integer N can be also represented as:

$$N = 2^{k+1}(1-y), \tag{8}$$

where $0 < y \leq 1$.

The conventional logarithmic approximation (1) uses the highest power of two smaller than the given number N. Instead, we propose the approximation given in Algorithm 1. Note that $2^k \leq N < 2^{k+1}$. As shown in Algorithm 1, when $N - 2^k < 2^{(k+1)} - N$ we underestimate the value of $log_2 N$ as k; otherwise, we overestimate it as $k+1$.

Algorithm 1 Proposed approximation for $log_2 N$

1: $N = 2^k(1+x) = 2^{k+1}(1-y)$
2: **if** $N - 2^k < 2^{(k+1)} - N$ **then** ▷ use underestimate
3: $x = N/2^k - 1$
4: $log_2 N \approx k + x$
5: **else** ▷ use overestimate
6: $y = 1 - N/2^{k+1}$
7: $log_2 N \approx k + 1 - y$
8: **end if**

The exact, Mitchell, and the proposed methods for computing $log_2 N$ are plotted in Fig. 3. The values of k corresponding to the N values, obtained from Algorithm 1, are also shown in Fig. 3. Only $k \geq 3$ is shown in order to keep the figure clear and easy to read. One power-of-two interval, i.e. $k = 6$, is also enlarged and depicted as an inset in Fig. 3 to better show the behavior of the two approximate methods compared to the exact $log_2 N$ function.

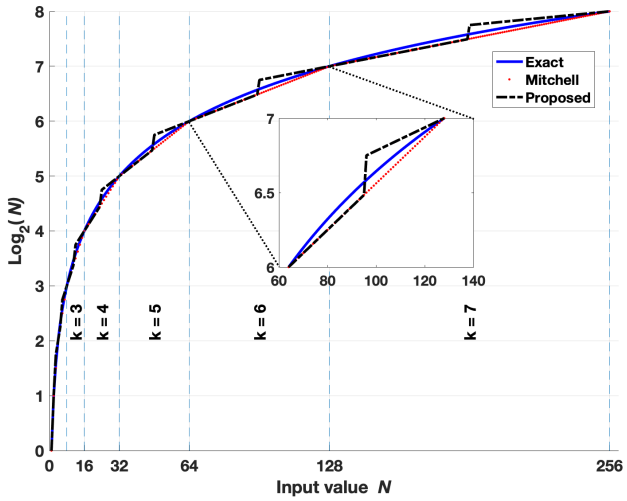


Fig. 3: Approximation of $log_2 N$.

Note that the proposed approximation results in a more than 6× smaller average error (over the range [1, 255]) than the Mitchell method (0.0088 vs. 0.0568), which is due to the double-sided error distribution of the proposed approach. With respect to the root mean square error (RMSE), the Mitchell approach is slightly more accurate than the proposed method (0.0627 vs. 0.0794). It is also evident in Fig. 3 that the magnitude of the error can be larger than the Mitchell method for the proposed method. However, the error analysis of the resulting multipliers with respect to two other error metrics, i.e. the common mean relative error distance (MRED) and the normalized mean error distance (NMED, by the maximum output of the accurate design) ([37], [38], [39]) shows that the proposed designs are actually more accurate. This occurs because using the proposed method for approximating $log_2 N$, the errors are likely to cancel out each other and, therefore, the proposed multiplier's accuracy increases.

To further illustrate the error behavior of the proposed method, we compare the mean error distance (MED) and the mean square error (MSE) values of the proposed and the Mitchell approaches. In the Mitchell approach $log_2(1+x) \approx x$. On the other hand, using (1) and (8) and according to Algorithm 1, $log_2 N$ can be approximated as follows:

$$log_2 N \approx \begin{cases} k + x, & \text{for } N = 2^k(1+x), \\ k + 1 - y, & \text{for } N = 2^{k+1}(1-y). \end{cases} \tag{9}$$

Given the approximated values for $log_2 N$, the MED and MSE for the proposed and the Mitchell approaches can be easily calculated. The resulting MSE_M for the Mitchell method is as follows:

$$MSE_M = \frac{1}{n} \times \sum_{k=0}^{n-1} \left[\frac{1}{2^k} \times \sum_{i=0}^{2^k-1} \left(log_2(1 + \frac{i}{2^k}) - \frac{i}{2^k} \right)^2 \right]. \tag{10}$$

Similarly, the MED_M for the Mitchell method can be easily calculated as:

$$MED_M = \frac{1}{n} \times \sum_{k=0}^{n-1} \left[\frac{1}{2^k} \times \sum_{i=0}^{2^k-1} \left| log_2(1 + \frac{i}{2^k}) - \frac{i}{2^k} \right| \right]. \tag{11}$$

The summation over k, where $k \in \{0, 1, ..., n-1\}$, is provided to cover the entire input range for an n-bit design. Similarly, the MED and MSE for the proposed approach can be calculated. For the proposed method, we need to divide the input domain into two regions. In the first region, the input operand is closer to the largest power of two smaller than or equal to it. In the second region, on the other hand, the input operand is closer to the smallest power of two that is larger than it. This can be done for the MSE_P as follows:

$$\begin{aligned} MSE_P = \frac{1}{n} \times \sum_{k=0}^{n-1} \Bigg[\frac{1}{2^k} \times \Bigg[& \sum_{i=0}^{2^{k-1}-1} \left(log_2(1 + \frac{i}{2^k}) - \frac{i}{2^k} \right)^2 \\ & + \sum_{i=2^{k-1}}^{2^k-1} \left(log_2(\frac{2^k + i}{2^{k+1}}) - \frac{2^k - i}{2^{k+1}} \right)^2 \Bigg] \Bigg]. \end{aligned} \tag{12}$$

We can use a similar approach to calculate the MED of the proposed method, MED_P, as given by:

$$\begin{aligned} MED_P = \frac{1}{n} \times \sum_{k=0}^{n-1} \Bigg[\frac{1}{2^k} \times \Bigg[& \sum_{i=0}^{2^{k-1}-1} \left| log_2(1 + \frac{i}{2^k}) - \frac{i}{2^k} \right| \\ & + \sum_{i=2^{k-1}}^{2^k-1} \left| log_2(\frac{2^k + i}{2^{k+1}}) - \frac{2^k - i}{2^{k+1}} \right| \Bigg] \Bigg]. \end{aligned} \tag{13}$$

The proofs for (10) and (12) are provided in Appendix A.1 and A.2, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/, respectively. (11) and (13), on the other hand, can be easily obtained by replacing the square of the error with the absolute values. Hence, the proofs for these two equations are not provided.

These equations are useful to mathematically evaluate the accuracy of the proposed method against the Mitchell method. As absolute values are considered in these equations, the Mitchell approach for estimating the base-2 logarithm of a positive integer provides smaller values of the MSE and MED, i.e., 0.0039 and 0.0568 for an 8-bit integer ($n = 8$), whereas those values are 0.0063 and 0.0681, respectively, for the proposed method. However, as mentioned earlier, the signed errors in the proposed method help with error cancellation and, consequently, multipliers that use the proposed method are shown to achieve a higher accuracy (in Section 5.1).

4.2 Improved LM (ILM) design

4.2.1 High-level description of the ILM design

The proposed ILM first transforms the multiplicand A and multiplier B to the closest powers of two plus an additional term, as given by:

$$A = 2^{k_1} + q_1, \tag{14}$$

$$B = 2^{k_2} + q_2, \tag{15}$$

and, therefore, the product $A \times B$ can be approximated as:

$$A \times B \approx (2^{k_1+k_2} + q_2 2^{k_1} + q_1 2^{k_2}) + \cancel{q_1 q_2}. \tag{16}$$

As shown in (16), the three most significant terms are all multiples of powers of two that can be efficiently implemented as left-shift operations in hardware. In this design, the least significant term ($q_1 q_2$) is ignored and left as the approximation error. A more detailed description of the ILM is provided in Algorithm 2, where NOD, PE and DEC denote the nearest-one detector, the priority encoder, and the decoder, respectively. Detailed descriptions of these three components are given in the following subsection.

Algorithm 2 Proposed logarithmic multiplication

1: **procedure** M(A, B)
2: A, B: inputs, γ: approximate output
3: $2^{k_1} \leftarrow \text{NOD}(A)$,
4: $k_1 \leftarrow \text{PE}(2^{k_1})$,
5: $q_1 \leftarrow A - 2^{k_1}$, ▷ for steps 3-5 see (14)
6: $2^{k_2} \leftarrow \text{NOD}(B)$,
7: $k_2 \leftarrow \text{PE}(2^{k_2})$,
8: $q_2 \leftarrow B - 2^{k_2}$, ▷ for steps 6-8 see (15)
9: $q_1 2^{k_2} \leftarrow q_1 << k_2$,
10: $q_2 2^{k_1} \leftarrow q_2 << k_1$,
11: $2^{k_1+k_2} \leftarrow \text{DEC}(k_1 + k_2)$,
12: $\gamma \leftarrow 2^{k_1+k_2} + q_2 2^{k_1} + q_1 2^{k_2}$. ▷ see (16)

4.2.2 Hardware implementation

The ILM can be implemented by either: (1) implementing the logic to calculate the nearest powers of two, or (2) using look-up tables (LUTs). We decided not to use LUTs as that would increase the memory usage, which is often a serious bottleneck for neural network applications [7], [11].

The block diagram of the 8-bit ILM is given in Fig. 4(a). The NOD circuits (Figs. 4(b) and 4(c)) are based on a leading-one detector (LOD) circuit. However, unlike the LOD, the NODs find the nearest power of two to a given input. Similar to some existing LODs [40], [41], the proposed NODs evaluate from the MSB to the LSB.

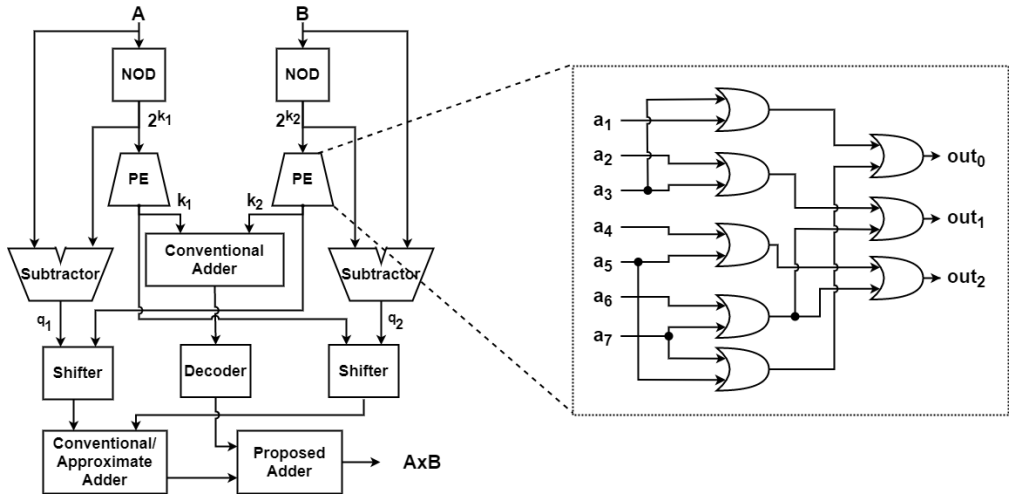
The priority encoder (PE) in Fig. 4(a) determines the number of required shifts based on the NOD's output. The two residue terms q_1 and q_2 are also calculated and shifted according to the k_2 and k_1 values, respectively, and a decoder generates the most significant term, $2^{k_1+k_2}$. Finally, the three resulting terms are summed up to obtain the approximate product. For hardware savings, we used the PE proposed in [35].

Fig. 4(b) depicts the design of the proposed NOD, where I and O are the primary input and output signals, respectively. The design is a simplified version of the NOD in [25]. Normally, nine bits are needed to represent the nearest power of two to an 8-bit input. However, as previously discussed, large synaptic weights are unlikely to appear in trained NNs and removing them would not significantly influence the performance of a NN. Hence, the NOD in [25] is simplified by rounding down the output of the NOD to the largest power of two representable in 8 bits, i.e. 128. In other words, up-rounding is not performed if the nearest-power of two is greater than 128. This will not have a significant detrimental impact on the performance of NNs, as supported by the simulation results in Section 5.

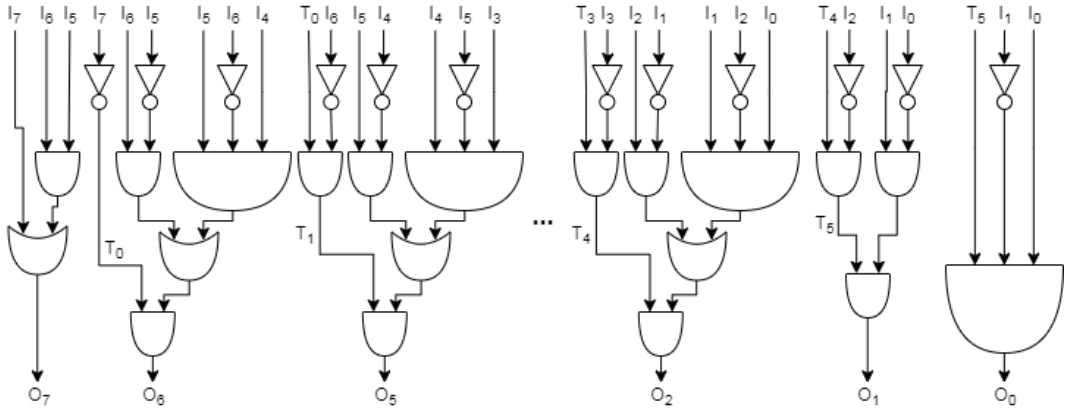
Note that the ILM needs to use the sign-magnitude representation for applications that require signed multiplications. It may not be as hardware-efficient as the 2's complement representation when multiply-accumulate operations are required. However, it is widely-used for logarithmic and non-logarithmic arithmetic circuits that are designed for unsigned numbers, e.g. [11], [12], [13]. Using this method, the sign bits of the two input operands are XOR-ed to obtain the sign of the final product and only the magnitude is computed using the design described here.

In order to further improve the hardware efficiency, we also propose a novel adder. This adder is used in the final stage, i.e. the adder that produces $A \times B$ in Fig. 4(a). There are three inputs to this adder (i.e., $2^{k_1+k_2}$, $q_1 \times 2^{k_2}$, and $q_2 \times 2^{k_1}$, step 12 in Algorithm 2), hence an adder tree composed of two adders is required. The conventional 8-bit adder (composed of conventional FAs) is used to add $q_1 \times 2^{k_2}$ and $q_2 \times 2^{k_1}$ and the proposed adder is used to add the result to the third term, $2^{k_1+k_2}$, see Fig. 4(a).

Note that the proposed adder is not an approximate design, however it has a simplified structure. Since $2^{k_1+k_2}$ is in a one-hot representation, the structure of the 8-bit adder can be modified and simplified accordingly. The truth tables for both the conventional and the proposed FAs are shown in Table 1. Note that the "not applicable" (N/A) entries in Table 1 cannot happen because there is only one '1' in one of the inputs. If A is a one-hot number and the



(a) Block diagram of the ILM and the priority encoder [35].



(b) Proposed nearest-one detector (NOD) Circuit.

Fig. 4: The proposed 8-bit improved logarithmic multiplier (ILM) design.

'1' is at bit position i, then it is not possible to have a carry in from less significant positions. The performance of the proposed adder was compared to the conventional FA and the results are given in Table 2. Both adders were implemented in VHDL and then synthesized using the Synopsys Design Compiler (DC) for ST Micro's CMOS 28-nm process. As shown, the proposed adder is 33.3% smaller and 60.27% more energy-efficient, with respect to the power-delay product (PDP) than the conventional full adder. This can significantly reduce the hardware implementation cost, as discussed in the next section. To further improve the hardware efficiency, the "conventional/approximate adder" in Fig. 4(a) is implemented with an approximate adder. A modified SOA-k adder is used in which, instead of setting all of the k LSBs to '1', these bits are set alternately to '1' and '0'. By doing so, the resulting adder can either overestimate or underestimate the result. Therefore, the double-sided error distribution property in the proposed ILM is preserved. The resulting ILM with the k modified LSBs will be referred to as ILM-k.

TABLE 1: The proposed vs. conventional full adder.

a	b	c_{in}	Conventional FA		Proposed FA	
			sum	c_{out}	sum	c_{out}
0	0	0	0	0	0	0
0	0	1	1	0	1	0
0	1	0	1	0	1	0
0	1	1	0	1	0	1
1	0	0	1	0	1	0
1	0	1	0	1	N/A	N/A
1	1	0	0	1	0	1
1	1	1	1	1	N/A	N/A
Conventional	$sum = \bar{a}\bar{b}c_{in} + \bar{a}b\overline{c_{in}} + a\bar{b}\overline{c_{in}} + abc_{in}$ $c_{out} = ab + ac_{in} + bc_{in}$					
Proposed	$sum = \bar{b}c_{in} + \bar{a}b\overline{c_{in}} + a\bar{b}$ $c_{out} = ab + bc_{in}$					

TABLE 2: Hardware comparison of the conventional and proposed full adders.

Full adder	Power (μW)	Delay (nS)	Area (μm^2)	PDP (fJ)
Conventional	1.32	0.09	3.42	0.1188
Proposed	**0.59**	**0.08**	**2.28**	**0.0472**

5 Performance Evaluation of Logarithmic Multipliers

A performance analysis of the LMs is presented in this section using both (1) accuracy and (2) hardware metrics.

Note that as 8-bit precision has been shown to be sufficient for most NN applications [8], [9], [10], only 8-bit multipliers are considered in this work.

5.1 Accuracy metrics

The error for the multiplication of $A = 2^{k_1}(1 + x_1)$ and $B = 2^{k_2}(1+x_2)$ depends on k_1 and k_2, i.e. the intervals of powers of two into which the input operands fall. Accordingly, the difference between the exact and approximate products is plotted for two designs, the Mitchell and the proposed ILM multipliers, in Fig. 5 for $A, B \in [0, 255]$ to provide a better visualization of the error behavior.

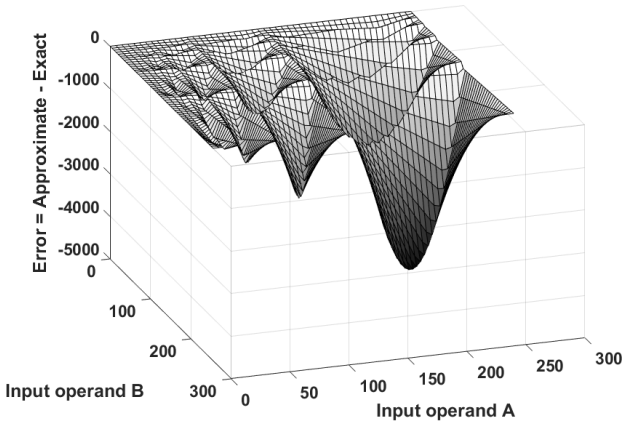
As shown in Fig. 5, the error tends to increase as k_1 and k_2 increase. However, the proposed design has a smaller peak error (3844 vs. 4096, magnitude-wise). As shown in Fig. 3, the proposed log_2N approximation has a double-sided error distribution. Note that the errors may cancel out even for a single multiplication, when the base-2 logarithm of the input operands are added.

Fig. 5 shows the error behavior over the multiplier's entire input domain. However, it has been shown that the trained synaptic weights in NN applications do not have a uniform distribution and they are mostly centered around zero [32] and [42]. On the other hand, the input distribution varies for the NN; it would depend on the application. Therefore, in Table 3 the accuracy metrics are reported for the two most common general distributions, the uniform and standard normal. In fact, a standard normal distribution with zero mean and unit variance is generated and the samples are then mapped to the range of [0, 255]. Note that since the sign-magnitude method is used, we do not need to worry about the sign of the generated samples. 10^6 input pairs were generated for multiplications using exact and logarithmic multipliers; the MRED, average error (AE), and the NMED [43] were then calculated. Note that the error distance is defined as the absolute difference between the exact and the approximate products, P_e and P_a, while the AE is calculated as:

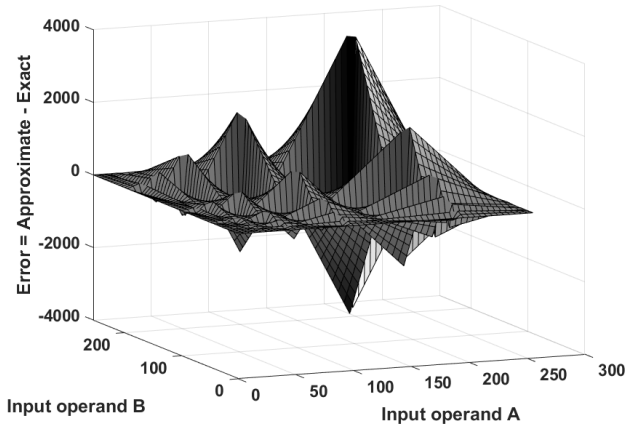
$$AE = \frac{1}{M} \times \sum_{i=1}^{M}(P_a - P_e), \tag{17}$$

where M is the number of random input pairs, 10^6 in the simulations. The parameter k in the ALM-SOA-k in Tables 3 and 4 indicates the number of LSBs to which approximation is applied. Similarly, IML-k indicates that the k LSBs are approximated (see section 4) in the proposed ILM.

The results in Table 3 show that the proposed ILM-0 is the most accurate design in terms of MRED and |AE|, for both input distributions. ILM-5, on the other hand, is the second most accurate design when the inputs are uniformly distributed. However, for the normal input distribution, the Mitchell, ALM-SOA-3, and the ALM-SOA-5 LMs perform better than the ILM-5 with respect to the MRED. Finally, ILM-9 has the worst error behavior as it approximates the nine LSBs, as explained in Section 4.2. The error metrics for the LMs when used in the two considered NN workloads are given in Table 4. Instead of assuming a general input distribution, as in Table 3, we performed all of the multiplications $x_i \times w_i$ in the NNs using LMs and then calculated their error metrics. The results in Table 4 show that ILM-0 and ILM-5 achieve the smallest AE and an average MRED for the two benchmark NNs. Because the error rate (ER) is generally high (more than 98% [17], [25]) for LMs due to the approximation in the base-2 logarithm and because it does not give any insight as to how close the approximated result is to the exact one, the ER values are not reported here.



(a) Mitchell's multiplier, $error_{maximum} = -4096$.



(b) The proposed ILM multiplier, $error_{maximum} = -3844$.

Fig. 5: Error characteristics for LMs (with outputs in [0, 65535]).

TABLE 3: Error metrics of the LMs for general input distributions.

Distribution	Multiplier Type	|AE|	MRED	NMED
Uniform	Mitchell [21]	618.91	0.0382	0.0095
	ALM-SOA-3 [17]	541.63	0.0336	0.0083
	ALM-SOA-5 [17]	550.84	0.0432	0.0085
	ILM-0	**0.25**	**0.0275**	**0.0068**
	ILM-5	28.03	0.0296	0.0068
	ILM-9	288.49	0.1069	0.0086
Normal	Mitchell [21]	93.90	0.0368	0.0014
	ALM-SOA-3 [17]	61.18	0.0292	0.0009
	ALM-SOA-5 [17]	46.00	0.0396	**0.0007**
	ILM-0	**5.24**	**0.0269**	0.0008
	ILM-5	19.26	0.0951	0.0010
	ILM-9	270.60	1.6982	0.0044

TABLE 4: Error metrics of the LMs for the two NN workloads.

NN Type	Multiplier Type	|AE|	MRED	NMED
MLP(784-128-10) MNIST dataset	Mitchell [21]	520.8	0.1374	0.0444
	ALM-SOA-3 [17]	43.1	0.1352	0.0319
	ALM-SOA-5 [17]	491.0	0.4361	0.0733
	ILM-0	**13.1**	**0.1193**	**0.0296**
	ILM-5	55.3	0.2539	0.0299
	ILM-9	602.4	0.4757	0.0933
Alexnet CIFAR-10 dataset	Mitchell [21]	181.20	**0.0279**	0.0133
	ALM-SOA-3 [17]	77.15	0.0297	0.0105
	ALM-SOA-5 [17]	46.13	0.0511	0.0189
	ILM-0	25.13	0.0300	0.0087
	ILM-5	**6.73**	0.0303	**0.0083**
	ILM-9	261.66	0.0869	0.0349

5.2 Hardware metrics

All of the designs were implemented using the VHDL hardware description language and then synthesized using the Synopsys Design Compiler (DC) for ST Micro's CMOS 28-nm process. The supply voltage and the temperature in all simulations were set to $1\,V$ and $25\,^{\circ}C$ respectively. Additionally, a 250 MHz clock frequency is used for estimating the power dissipation. All designs were synthesized with a cell library that includes AND-OR-Inverter (AOI) logic gates and without any optimization. Although the designs in DC can be optimized with respect to different metrics (delay, area, and power consumption) during the synthesis process, we neither performed any optimization nor imposed any timing constraint on any of the designs to ensure a fair comparison. Moreover, we used the default toggle rate for power estimation (i.e., 0.5), input-drive strength, and output capacitive load for all the simulations.

The hardware measurements for four key metrics are given in Table 5. The design in [44] was considered as the reference Mitchell multiplier as [21] does not detail any particular hardware implementation. Only the basic block in [44] was implemented, i.e. no iterations, as iterative algorithms can be applied to any logarithmic design and they would significantly increase the hardware costs [35].

TABLE 5: Hardware metrics of the logarithmic multipliers.

Multiplier	Power (μW)	Delay (nS)	Area (μm^2)	PDP (fJ)
Array	91.01	1.39	293.5	126.5
Wallace	99.30	1.06	235.9	105.2
Mitchell [21]	66.26	1.42	281.2	94.09
ALM-SOA-5 [17]	61.04	**1.39**	255.4	84.84
ILM-0	53.72	1.68	287.4	90.25
ILM-5	**50.37**	1.64	255.3	**82.61**

As shown in Table 5, the smallest design is ILM-5, which is 14.68% smaller than the reference Mitchell design while being almost 20% more accurate (see Tables 3 and 4). With respect to delay, the ALM-SOA and Mitchell multipliers are 17.98% and 15.49% faster than the proposed ILM-5. However, the results in Table 5 show that ILM-5 has the lowest PDP among all the considered designs. In term of power consumption, ILM-5 is the most efficient design, consuming 21.18% less power than its closest competitor, ALM-SOA-5. Note that the array and Wallace multipliers are standard (non-logarithmic) multipliers that were simulated only to confirm the benefits of using LMs.

Although ILM uses a more complex method for calculating the base-2 logarithm of the input operands, it uses some other simplifications that make it more power-efficient than the Mitchell multiplier. For example, ILM uses a compact adder at the final stage that is almost 3x more energy-efficient than the conventional adder used in the Mitchell multiplier. The other simplification lies in the approximation of a few LSBs (in ILM-5) for further hardware saving. Finally, a more power-efficient PE is used in the ILA.

Fig. 6 shows the MRED versus PDP for the considered multipliers. The designs with lower MRED and smaller PDP and area (in the bottom-left corners of the two figures) are preferable. It can be seen that ILM-5 is the most hardware-efficient design, while being the third most accurate one. Note that the MRED in Fig. 6 is the average MRED over the two benchmark NNs, which are given in Table 4.

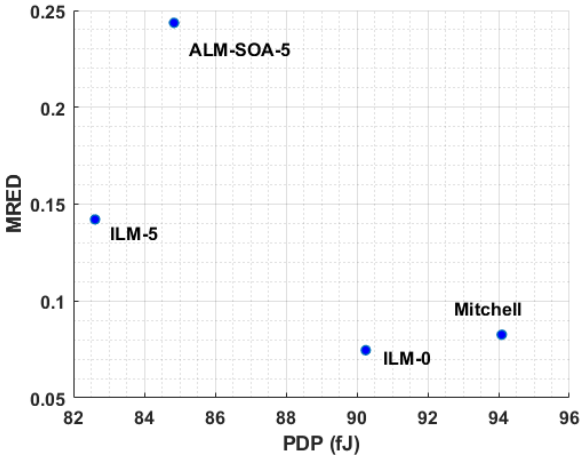


Fig. 6: MRED vs. PDP trade-off: the proposed ILM can achieve a higher accuracy (ILM-0) and a better trade-off in accuracy and hardware efficiency (ILM-5), compared to other LMs.

The hardware complexity of the LMs, including the proposed design, increases almost linearly with the input size N, while that of the conventional multipliers increases almost quadratically with N [17]. Hence, more significant savings can be expected when the proposed multiplier is extended to larger designs (e.g., 64-bit multipliers).

6 Neural Network Applications

The proposed ILM can be used in various types of NNs; however, the most common feed-forward NNs are studied in this work. Two NNs are considered to evaluate the LMs. The first one is an MLP that classifies the MNIST dataset and the other one is a CNN that classifies the CIFAR-10 dataset.

6.1 Neural network workloads

6.1.1 An MLP for classifying the MNIST dataset

MNIST is a dataset 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 [22]. We used an MLP network with 784 input neurons (one for each pixel of the monochrome image), 128 neurons in the hidden layer and 10 output

neurons. The outputs are interpreted to be the probability of classification into the 10 target classes of the digits 0 to 9 [22]. The MLP uses the soft-limiting sigmoid activation function.

The exact multipliers in the considered MLP are replaced with each of the LMs and the top-1 [23] classification accuracy is evaluated. The weights are plotted in Fig. 7 to determine if the trained synaptic weights are normally distributed. Since an 8-bit width is used for inference and the most significant bit is the sign bit, the trained synaptic weights are mapped into the range [-127, 127] in Fig. 7.

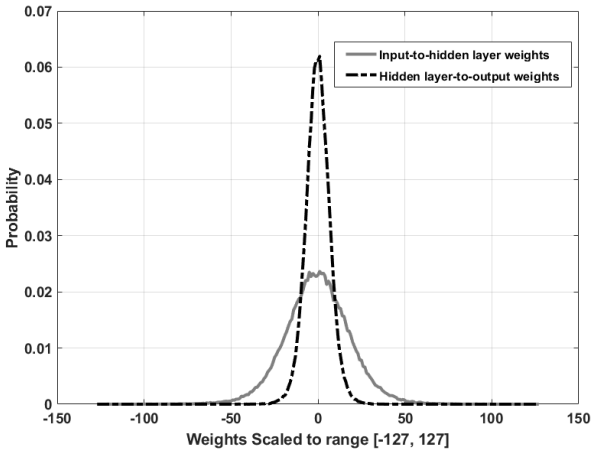


Fig. 7: Probability distribution of the trained weights for the (784-128-10) MLP, mapped into the range [-127, 127].

6.1.2 A CNN for classifying the CIFAR-10 dataset

CIFAR-10 is a dataset containing images for ten classes, namely, airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset consists of a training set of 50,000 and a test set of 10,000 32×32 colour images. It is a common dataset that is widely used in machine learning algorithms, NNs in particular [24].

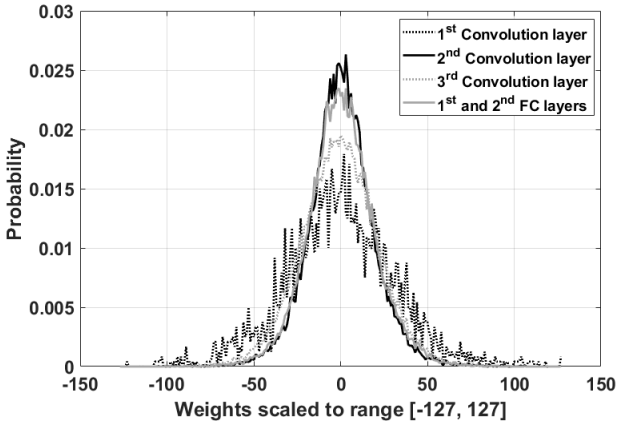


Fig. 8: Probability distribution of the trained weights for Alexnet, mapped into the range [-127, 127].

AlexNet is used to classify the CIFAR-10 dataset [23]. AlexNet is a CNN configuration that uses rectified linear unit (ReLU) activation functions and has three convolutional layers, two fully-connected (FC) layers, max-pooling layers and average pooling layers. Similarly, the exact multipliers (in the convolutional and fully-connected layers) were replaced with each type of the LMs and the resulting top-1 classification accuracy was evaluated. The ReLUs were used as the activation functions in this network. An ReLU has an output of zero if the input is less than zero, otherwise the output is equal to the input [45]. The weights are also plotted in order to investigate the distribution of the trained synaptic weights. The weights are mapped into the range [-127, 127], as shown in Fig. 8.

Figs. 7 and 8 clearly show that, as mentioned before, it is less likely to have larger weights (magnitude-wise) in trained neural networks. This justifies the proposed simplification in the design of the NOD, i.e. rounding down the output of the NOD to the largest power of two representable in 8 bits.

6.2 Accuracy analysis

The top-1 classification accuracy comparison for both the MNIST and the CIFAR-10 datasets using the various LMs are plotted in Fig. 9. Unlike previous studies, such as [11], [13], retraining has not been performed. The proposed designs ILM-5 and ILM-9 show only 0.08% and 0.12% accuracy degradation, respectively, compared to the NN with exact multipliers for the MNIST dataset (with accuracy 98.13%). The ALM-SOA-3, on the other hand, has the same accuracy as the NN with exact multipliers.

As also shown in Fig. 9, all three variants of the proposed ILM increase the classification accuracy for the CIFAR-10 dataset compared to the NN with exact multipliers (with accuracy 82.53%). We can obtain up to 1.4% accuracy improvement by using ILM-0 instead of the exact multipliers. The double-sided (signed) error distribution and the low error magnitude of the proposed design help mitigate any overfitting in Alexnet. In fact, the double-sided errors with lower magnitudes effectively introduce noise into the proposed ILM. Hence, by using the ILM multiplier we are adding noise to the NN. It has already been shown that adding noise is often an effective way of improving the performance of NNs [46], [47], [48]. This result indicates that higher classification accuracies can be obtained with lower hardware costs by replacing the exact multipliers with appropriately chosen approximate multipliers.

6.3 Hardware analysis

The hardware characteristics of the LMs were discussed in Section 5. In this section, an artificial neuron is implemented using different types of LMs to replace the conventional 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 [11], [49].

The hardware characteristics for the implemented neuron are given in Table 6. The results show that the neuron constructed using the proposed ILM-5 has the lowest energy consumption. It is 21.85% more energy-efficient than the neuron using ALM-SOA-5 while being 2.25% smaller. Truncation needs to be considered in the implementation of

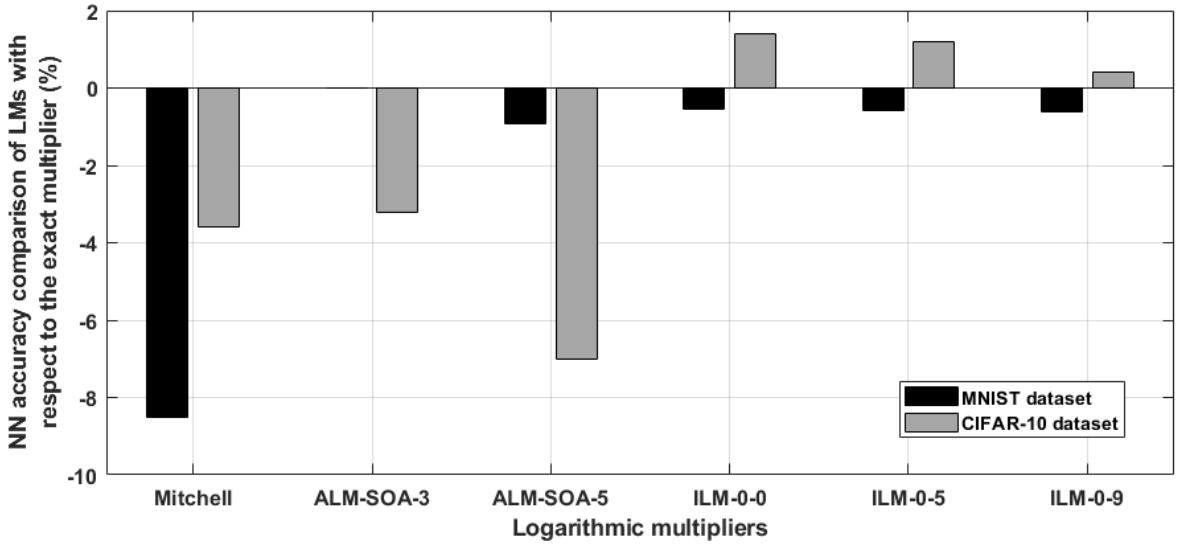


Fig. 9: Comparison of the top-1 classification accuracy of the MNIST and CIFAR-10 datasets with LMs (decrease or increase in accuracy from using accurate multipliers).

TABLE 6: Hardware characteristics of the artificial neuron when implemented using four different LMs.

Multiplier used in the neuron	**Energy** (fJ)	**Area** (μm^2)
Mitchell [21]	93.85	1019.5
ALM-SOA-5 [17]	68.23	915.1
ILM-0	87.17	1004.8
ILM-5	**53.32**	894.5

the artificial neuron. Eight-bit precision is used for each of the three inputs and their corresponding synaptic weights and, therefore, the multiplication products would be 16-bit. However, since the output will be connected to another layer of neurons, truncation to 8 bits is required. The truncation is done by performing hard-limiting, i.e. using the maximum 8-bit number (-127 or 127) for all output values that need more than 8-bit precision.

7 Conclusion

This work proposes a novel approximation method to efficiently compute $log_2 N$. Using this method, an improved logarithmic multiplier (ILM) is designed. The proposed ILM is more accurate and has the smallest MRED values compared to other logarithmic designs in the literature. We observed that a few LSBs can be approximated in the ILM for saving hardware without a significant effect on its accuracy. For example, ILM with five approximation bits, ILM-5 can be 6.78%-17.48% more power-efficient and up to 6.07% smaller than the recent design in [17]. Finally, two well-known NNs were considered as benchmark applications, for which the proposed designs show a higher top-1 classification accuracy than the other designs. The exact multipliers in both NNs were replaced with LMs and the ILM-5 resulted in the most energy-efficient NN structure. Interestingly, higher classification accuracies are obtained for the CIFAR-10 dataset by using the ILM compared to the use of exact (and other LM) multipliers.

References

[1] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[2] L. Shao, D. Wu, and X. Li, "Learning deep and wide: A spectral method for learning deep networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 12, pp. 2303–2308, 2014.

[3] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations." *Journal of Machine Learning Research*, vol. 18, pp. 187–1, 2017.

[4] N. Sudha, A. Mohan, and P. K. Meher, "A self-configurable systolic architecture for face recognition system based on principal component neural network," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 8, pp. 1071–1084, 2011.

[5] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," *Proceedings of the 28th International Conference on Machine Learning*, pp. 265–272, 2011.

[6] B. Rajendran, Y. Liu, J. S. Seo, K. Gopalakrishnan, L. Chang, D. J. Friedman, and M. B. Ritter, "Specifications of nanoscale devices and circuits for neuromorphic computational systems," *IEEE Transactions on Electron Devices*, vol. 60, no. 1, pp. 246–253, 2013.

[7] 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," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 151–156, 2016.

[8] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," *ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12, 2017.

[9] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," *International Conference on Machine Learning*, pp. 1737–1746, 2015.

[10] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv:1609.08144. [Online]. Available: https://arxiv.org/abs/1609.08144*, 2016.

[11] S. S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, and K. Roy, "Energy-efficient neural computing with approximate multipliers," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, no. 2, p. 16, 2018.

[12] U. Lotrič and P. Bulić, "Applicability of approximate multipliers in hardware neural networks," *Neurocomputing*, vol. 96, pp. 57–65, 2012.

[13] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, 2016.
[14] V. Paliouras and T. Stouraitis, "Low-power properties of the logarithmic number system," *15th IEEE Symposium on Computer Arithmetic*, pp. 229–236, 2001.
[15] S. Gandhi, M. S. Ansari, B. F. Cockburn, and J. Han, "Approximate leading one detector design for a hardware-efficient Mitchell multiplier," *IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, pp. 1–4, 2019.
[16] J. Y. L. Low and C. C. Jong, "Unified Mitchell-based approximation for efficient logarithmic conversion circuit," *IEEE Transactions on Computers*, vol. 64, no. 6, pp. 1783–1797, 2015.
[17] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi, "Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 9, pp. 2856–2868, 2018.
[18] D. De Caro, N. Petra, and A. G. Strollo, "Efficient logarithmic converters for digital signal processing applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 10, pp. 667–671, 2011.
[19] T. B. Juang, S.-H. Chen, and H.-J. Cheng, "A lower error and ROM-free logarithmic converter for digital signal processing applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, no. 12, pp. 931–935, 2009.
[20] B.-G. Nam, H. Kim, and H.-J. Yoo, "Power and area-efficient unified computation of vector and elementary functions for handheld 3D graphics systems," *IEEE Transactions on Computers*, vol. 57, pp. 490–504, 2008.
[21] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.
[22] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2001.
[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.
[24] A. Krizhevsky, V. Nair, and G. Hinton, "The CIFAR-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.
[25] M. S. Ansari, B. Cockburn, and J. Han, "A hardware-efficient logarithmic multiplier with improved accuracy," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 922–925, 2019.
[26] C. Teuscher, *Turing's connectionism: an investigation of neural network architectures*. Springer Science & Business Media, 2012.
[27] B. C. Csáji, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, p. 48, 2001.
[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[29] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," *Neural computing and applications*, pp. 1–31, 2018.
[30] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected CRFs," *Proceedings of the 3rd International Conference on Learning Representations, [online] Available: http://arxiv.org/abs/1412.7062*, 2015.
[31] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.
[32] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," *Proceedings of the 32nd International Conference on Machine Learning*, pp. 1613–1622, 2015.
[33] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR, abs/1207.0580. [online] Available: http://arxiv.org/abs/1207.0580*, 2012.
[34] H. Saadat, H. Bokhari, and S. Parameswaran, "Minimally biased multipliers for approximate integer and floating-point multiplication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2623–2635, 2018.
[35] M. S. Kim, A. A. Del Barrio, R. Hermida, and N. Bagherzadeh, "Low-power implementation of Mitchell's approximate logarithmic multiplication for convolutional neural networks," *23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 617–622, 2018.
[36] M. S. Kim, A. A. D. B. Garcia, L. T. Oliveira, R. Hermida, and N. Bagherzadeh, "Efficient Mitchell's approximate log multipliers for convolutional neural networks," *IEEE Transactions on Computers*, 2018.
[37] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 4, p. 60, 2017.
[38] H. Jiang, F. J. Santiago, M. S. Ansari, L. Liu, B. F. Cockburn, F. Lombardi, and J. Han, "Characterizing approximate adders and multipliers optimized under different design constraints," pp. 393–398, 2019.
[39] M. S. Ansari, H. Jiang, B. F. Cockburn, and J. Han, "Low-power approximate multipliers using encoded partial products and approximate compressors," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3, pp. 404–416, 2018.
[40] K. H. Abed and R. E. Siferd, "CMOS VLSI implementation of a low-power logarithmic converter," *IEEE Transactions on Computers*, vol. 52, no. 11, pp. 1421–1433, 2003.
[41] K. Kunaraj and R. Seshasayanan, "Leading one detectors and leading one position detectors-an evolutionary design methodology," *Canadian Journal of Electrical and Computer Engineering*, vol. 36, no. 3, pp. 103–110, 2013.
[42] Y. Xie, S. Liao, B. Yuan, Y. Wang, and Z. Wang, "Fully-parallel area-efficient deep neural network design using stochastic computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 12, pp. 1382–1386, 2017.
[43] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Transactions on computers*, vol. 62, no. 9, pp. 1760–1771, 2013.
[44] Z. Babić, A. Avramović, and P. Bulić, "An iterative logarithmic multiplier," *Microprocessors and Microsystems*, vol. 35, no. 1, pp. 23–33, 2011.
[45] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.
[46] C. Wang and J. C. Principe, "Training neural networks with additive noise in the desired signal," *IEEE Transactions on Neural Networks*, vol. 10, no. 6, pp. 1511–1517, 1999.
[47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
[48] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019.
[49] S. S. Sarwar, S. Venkataramani, A. Raghunathan, and K. Roy, "Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing," *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pp. 145–150, 2016.

Mohammad Saeed Ansari received the B.Sc. and M.Sc. degrees in electrical and electronic engineering in 2013 and 2015, respectively, from Iran University of Science and Technology, Tehran, Iran. In 2019 he received the Ph.D. degree in Integrated Circuits and Systems from the University of Alberta, Edmonton, AB, Canada. He is presently a digital design engineer at Eideticom Computational Storage, Calgary, AB, Canada. His research interests include approximate computing and developing hardware accelerator IP cores for data compression/decompression, neural networks, and digital signal processing applications.

Bruce F. Cockburn (S'86-M'90) received the B.Sc. degree in engineering physics in 1981 from Queens University, Kingston, ON, Canada. In 1985 and 1990 he received the M.Math. and Ph.D. degrees, respectively, in computer science from the University of Waterloo, Waterloo, ON. He is presently a Professor in the Department of Electrical and Computer Engineering at the University of Alberta, Edmonton, AB, Canada. From 1981 to 1983 he was a Test Engineer and Software Designer at Mitel Corporation in Kanata, ON. In 2001 he was a sabbatical visitor at Agilent Technologies Inc. in Santa Clara, CA, USA. From 2014 to 2015 he was a sabbatical visitor at the University of British Columbia in Vancouver, BC, Canada. His research interests include the testing and verification of integrated circuits, application-specific hardware accelerators, applications of high-level synthesis and field-programmable gate arrays, heterogeneous parallel computing, stochastic and approximate computing, and genetic data processing. Dr. Cockburn is a member of the Association for Computing Machinery and is registered as a Professional Engineer with the Association of Professional Engineers and Geoscientists of Alberta.

Jie Han (S02-M05-SM16) 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, The Netherlands, in 2004. He is currently a Professor in the Department of Electrical and Computer Engineering at the University of Alberta, Edmonton, AB, Canada. His research interests include approximate computing, stochastic computation, reliability and fault tolerance, nanoelectronic circuits and systems, 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 was nominated for the 2006 Christiaan Huygens Prize of Science by the Royal Dutch Academy of Science. His work was recognized by Science, for developing a theory of fault-tolerant nanocircuits (2005). He is currently an Associate Editor for the IEEE Transactions on Emerging Topics in Computing (TETC), IEEE Transactions on Nanotechnology, IEEE Circuits and Systems Magazine and Microelectronics Reliability (Elsevier Journal). He served as a General Chair for GLSVLSI 2017 and the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT 2013), and a Technical Program Committee Chair for GLSVLSI 2016 and DFT 2012.