UNO: Virtualizing and Unifying Nonlinear Operations for Emerging Neural Networks

Di Wu, Jingjie Li, Setareh Behroozi, Younghyun Kim, and Joshua San Miguel
Department of Electrical and Computer Engineering, University of Wisconsin–Madison, Madison, Wisconsin, 53706
di.wu@ece.wisc.edu, {jingjie.li, sbehroozi, younghyun.kim, jsanmiguel}@wisc.edu

***Abstract*—Linear multiply-accumulate (MAC) operations have been the main focus of prior efforts in improving the energy efficiency of neural network inference due to their dominant contribution to energy consumption in traditional models. On the other hand, nonlinear operations, such as division, exponentiation, and logarithm, that are becoming increasingly significant in emerging neural network models, have been largely underexplored. In this paper, we propose *UNO*, a low-area, low-energy processing element that *virtualizes* the Taylor approximation of nonlinear operations on top of off-the-shelf linear MAC units already present in inference hardware. Such virtualization approximates multiple nonlinear operations in a unified, MAC-compatible manner to achieve dynamic run-time accuracy-energy scaling. Compared to the baseline, our scheme reduces the energy consumption by up to 38.4% for individual operations and increases the energy efficiency by up to 274.5% for emerging neural network models with negligible inference loss.**

I. Introduction

Deep neural networks (DNNs) have overwhelmed many approaches for various classification and optimization problems, such as computer vision, natural language processing and speech recognition, over recent years. To boost the energy efficiency of DNN inference, a large body of work has focused on linear multiply-accumulate (MAC) operations, which dominate the total computation in most convolutional neural networks (CNNs) [1]. On the other hand, other nonlinear operations such as the max pooling and rectified linear unit (ReLU) [1], [2] operations have received little attention because of their relatively low contribution to the area and energy overheads. However, there is a recent trend towards new sophisticated models that boost accuracy by incorporating more nonlinear operations, including division (*div*), exponentiation (*exp*), logarithm (*log*), sigmoid (*sigmoid*), softmax (*softmax*), and hyperbolic tangent (*tanh*) as in Table I, and they commonly require more than one type of nonlinear operation.

Though boosting the application accuracy, encapsulating multiple nonlinear operations together poses several challenges to the existing inference hardware. **First, nonlinear operations are much less frequent than linear operations, but they counterintuitively hinder the overall DNN efficiency on general-purpose hardware, such as CPUs and GPUs.** Sophisticated nonlinear operations in emerging DNNs (Table I) are often computation-heavy and spread out in the entire model, unlike those in conventional CNNs that either exist at the output or are simple enough to implement. We profile the inference of a Seq2Seq model with attention for neural machine

Table I
Nonlinear operations used in emerging DNNs.

DNN Model	Nonlinear Operations
CapsNet [8]	*div*, *exp*, *log*, *sigmoid*, *softmax*
GNN [9]	*div*, *exp*, *log*, *softmax*
NMT [10]	*log*, *sigmoid*, *softmax*, *tanh*

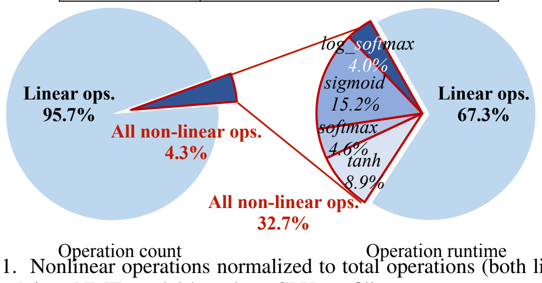


Figure 1. Nonlinear operations normalized to total operations (both linear and nonlinear) in a NMT model based on CPU profiling.

translation (NMT) on a CPU in Fig. 1, showing the operation counts and runtimes of linear and nonlinear operations. It is observed that though nonlinear operations only count for 4.3% of the total operation count, they require almost one third of the total execution time. **Second, existing DNN accelerators are not forward-adaptable to new models because they only support a small subset of nonlinear operators they needed at the time.** Authors in [3], [4] present the ASIC and FPGA implementations of the basic cell for a NMT model, long short-term memory (LSTM), which includes two unary (single-input) nonlinear operations, *sigmoid* and *tanh*, approximated by look-up tables (LUTs) and piece-wise linear approximation. However, those single-input approximations are not compatible with non-unary nonlinear operations such as *div* (two-input) and *softmax* (many-input). **Third, in general, nonlinear operators require bulkier hardware than linear operators, making it expensive to simply slab on a new nonlinear operator, especially considering it is not likely to be used often across all emerging DNNs.** Among these nonlinear operations in Table I, for unary *sigmoid* and *tanh* in LSTM, bulky LUTs or extra MAC units are applied [3], [4]; for many-input *softmax* in CapsNet, extra LUTs, adder and dividers are introduced [5]. All those add-ons encumber the savings from the energy-optimized MAC arrays [1], [6], [7] (SIMD arrays in our work).

The above challenges, together with the fact that minor numerical errors do not necessarily degrade the DNN accuracy [1], motivate two key insights in our work: 1) implement nonlinear operations *by effectively virtualizing them on top of the handy MAC array in DNN accelerators* to achieve high efficiency; 2) allow to further improve the efficiency *by offering dynamic run-time accuracy-energy scaling in nonlinear oper-*

978-1-6654-3922-0/21/$31.00 ©2021 IEEE

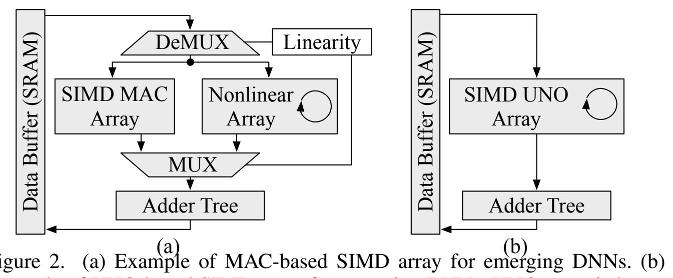


Figure 2. (a) Example of MAC-based SIMD array for emerging DNNs. (b) Example of UNO-based SIMD array for emerging DNNs. UNO array is larger than MAC array, but it outperforms when counting the nonlinear support. Here, the circle indicates multi-cycle nonlinear operations.

ations. Fig. 2(a) shows an example architecture using a MAC-based SIMD array for emerging DNNs. Here, the nonlinear array is in parallel to the SIMD array, since the nonlinear operations found in emerging DNNs can be cascaded. DeMUX and MUX select the operation to be performed according to mode configuration. Though nonlinear hardware prospers in recent years [11]–[15], they share little hardware from each other and the MAC array, limiting the efficiency. As complex nonlinear operations can be decomposed into a combination of linear and three primitive nonlinear operations, *div*, *exp* and *log*, *Taylor approximation is able to enforce maximal hardware sharing between varying nonlinear operations and existing MAC array* and minimize the overhead.

Leveraging Horner's rule [16], we schedule the Taylor approximation in a MAC-compatible flow so that it operates as *unified nonlinear operations (UNO)*. We propose the UNO architecture to *virtualize* all identified nonlinear operations on top of off-the-shelf MAC arrays with low overhead and encapsulate all arithmetic operations in emerging DNNs. An example SIMD array based on UNO is shown in Fig. 2(b). There is no explicit linearity selection on the data path, as the UNO array itself can be configured to execute either linear MAC or nonlinear operations. Furthermore, the multi-cycle behavior of Taylor approximation naturally offers dynamic run-time accuracy-energy scaling, i.e., more Taylor terms produces higher accuracy [11], [12], [15]. Unit- and application-level experiments show that, without losing benefits in accuracy and efficiency, UNO achieves compatibility for all mentioned nonlinear operations simultaneously and much lower area compared to existing designs. Contributions are listed as follows:

- We identify and characterize the inefficiencies of prior implementations of complex nonlinear operations, particularly in the context of emerging neural network models.
- We propose *unified nonlinear operations (UNO)*, which *virtualizes* complex nonlinear operations on top of off-the-shelf MAC arrays with low overhead and achieves dynamic run-time accuracy-energy scaling.
- We demonstrate superior area and energy efficiency with UNO compared to prior designs while maintaining low accuracy loss at both operation-level (i.e., per individual nonlinear operation implementation) and application-level (i.e., analytical case study on three emerging DNNs).

II. Approximate Computing

Various application domains can leverage approximate computing to trade off accuracy for energy efficiency, such as computer vision [11], [17], [18] and neuron simulation [12]. And both linear and nonlinear operations are supported, e.g., *add* [17], *mul* [18], *div* [11], [14], *exp* [12], and *log* [13]. Based on these primitive, non-decomposable nonlinear operations, complex nonlinear operations can be obtained, i.e., *sigmoid*, *softmax* and *tanh* [4], [19]. Besides naively utilizing LUTs [3], these primitive nonlinear operations can also be approximated with Taylor approximation [11], [12], piece-wise linear approximation [4], logarithmic transform [14], and DNN approximation [13]. Among these, LUTs exhibit exponential overheads with the bit width [11] and will not be elaborated.

Taylor approximation of degree n, $f_n(x)$, is a series expansion of a function f over input x about an expansion point a:

$$f_n(x) = \sum_{i=0}^{n} \frac{f^{(i)}(a)}{i!} \cdot (x-a)^i = \sum_{i=0}^{n} c_i \cdot (x-a)^i, \qquad (1)$$

where $f^{(i)}(a)$ is the i-th derivative of f evaluated at $x = a$, and $c_i = f^{(i)}(a)/i!$ denotes the coefficient of the i-th term. The approximation accuracy increases as the degree n increases. Inherently, a Taylor approximation consists of only addition and multiplication, under a fixed point a. Prior works [11], [12] further approximate Eq. (1) by recursively derive it as

$$f_n(x) \approx f_{n-1}(x) + \frac{c_n}{c_{n-1}} \cdot pow_{n-1} \cdot (x-a), \qquad (2)$$

where $pow_{n-1} = c_{n-1} \cdot (x-a)^{n-1}$, and $f_0(x) = c_0$. The benefit of this approach is that the calculation can be terminated at any cycle for dynamic accuracy-energy trade-off at run time, as the computation is performed from low order to high order. However, the potential to integrate those standalone hardware units to MAC arrays is not revealed.

Piece-wise linear approximation divides the function curve into multiple pieces, and then approximates each piece with a line segment, which requires LUTs to store the coefficients and MAC units to calculate the result. Existing accelerators utilize this technology to approximate *sigmoid* and *tanh* [4], [20] but introduce large hardware overhead, as they do not leverage the existing MAC arrays for matrix operations. Furthermore, unlike the Taylor approximation increases the accuracy over time, the accuracy increase of this method requires more pieces, i.e., more LUTs to store the coefficients.

Logarithmic transform [14] performs simpler computation in logarithmic domain instead of binary domain. For example, division can be performed in logarithmic computation as subtraction. However, this method is not optimal when applied to emerging DNN models. First, it requires extra hardware logic, like leading zero detection and barrel shifter, which are absent in normal MAC arrays. Second, existing literature does not provide hints to apply this technique to more sophisticated nonlinear operations like *exp*.

DNN approximation [13] can approximate more complicated transcendental functions. Though the method is beneficial than CPU/GPU software libraries for energy efficiency, it is not appropriate for accelerators due to unacceptable hardware cost.

Among those methods, only Taylor approximation is able to explore the accuracy-energy trade-off dynamically at run time[1]

[1]Stochastic computing [21] also allows dynamic run-time accuracy-energy scaling, but works on serial unary bitstreams, instead of parallel binary bits.

by adjusting the number of execution cycles/Taylor terms, while others only permit the static trade-off at design time. In this work, UNO applies Taylor approximation to unify nonlinear operations based on Horner's rule [16], so as to support dynamic run-time accuracy-energy scaling with minimal hardware overhead by fully utilizing the MAC units. Note that other polynomial approximations [22], [23] also fit UNO well.

III. Unified Nonlinear Operations

Taylor approximations in Eq. (2) is *not MAC-compatible*, since it requires two successive multiplications but as a MAC unit supports only one multiplication per cycle (unless $c_n = c_{n-1}$ as in [11]). To ensure single-cycle updates in Eq. (2), extra hardware overhead and/or accuracy loss may occur [12]. To make our Taylor approximation MAC-compatible, based on Horner's rule [16] where polynomials are evaluated from high to low order, UNO expresses Eq. (1) as

$$f_n(x) = \mathit{offset} + \mathit{mac}_n \cdot \mathit{scale}, \tag{3}$$

where mac_n is the cascaded MAC result for Taylor approximation of degree n, and *scale* and *offset* are used to tune the mac_n towards the final result. Then mac_n is calculated as

$$mac_i = \begin{cases} |c_{n-i}| + mac_{i-1} \cdot var & \text{if } 1 < i \leq n, \\ |c_{n-1}| + |c_n| \cdot var & \text{if } i = 1, \end{cases} \tag{4}$$

where *var* is an affine transformation of input x and the index i starts from 1. At each round of cascaded MAC operations, the coefficients are simply converted to the absolute values to take effect. Note that the intermediate result mac_i is not a valid approximation. Therefore, UNO takes $n+1$ cycles to finish computation in Eq. (3, 4), the same as that in prior works that compute from low order to high order [11], [12].

An example of UNO is shown in Eq. (5). When $x \in [0.5, 1.0]$, $\log(x)$ with a degree of $n = 2$ at expansion point $a = 1$ results in $scale = -1$, $offset = 0$ and $var = 1 - x$. The three underlines indicate that three MAC operations are required for the Taylor approximation of degree 2, and the MAC computation is performed as marked by underlines from short to long, i.e., from high- to low-order terms. The two shorter underlines are for mac_n in Eq. (4) and the longest is for final $f_n(x)$ in Eq. (3).

$$\begin{aligned} \log(x) &= \ln a + \frac{1}{a}(x-a) - \frac{1}{2a^2}(x-a)^2 \\ &= 0 + (x-1) - \frac{1}{2}(x-1)^2 \\ &= \underline{0 + \underline{\left(0 + \underline{\left(1 + \frac{1}{2}(1-x)\right)}(1-x)\right)}(-1)} \end{aligned} \tag{5}$$

We generalize UNO for *div*, *exp* and *log* by deriving their *scale*, *offset* and *var* in Table II, where x and y are the inputs with arbitrary value ranges, and a_{div}, a_{exp} and a_{log} are the Taylor expansion points for *div*, *exp* and *log*, respectively. To ensure high accuracy in Eq. (4), an integer e_x normalizes x as $x_{norm} \in [0.5, 1.0]$ such that $x = e_x \cdot x_{norm}$ [11]. We also use $x - \lfloor x \rfloor$ in *var* of *exp* to reduce the input range to $[0.0, 1.0]$.

Table II
Parameter configuration for *div*, *exp* and *log*

Parameter	y/x	$\exp(x)$	$\log(x)$
scale	$y \cdot 2^{-e_x}$	$\exp(\lfloor x \rfloor + a_{exp})$	-1
offset	0	0	$\log(2^{e_x})$
var	$a_{div} - x_{norm}$	$x - \lfloor x \rfloor - a_{exp}$	$a_{log} - x_{norm}$

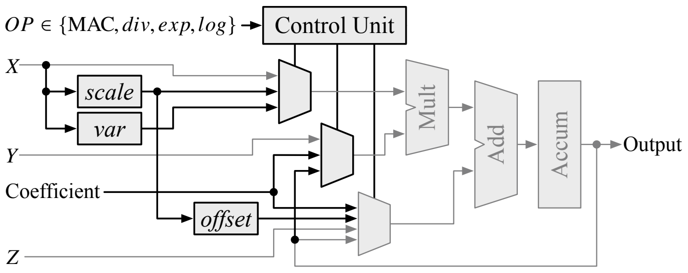


Figure 3. UNO PE architecture that virtualizes nonlinear operations on top of a standard MAC unit. The gray blocks are the base MAC unit, while the black blocks are added for UNO.

IV. Hardware and Implementation

In this section, we present the hardware architecture of UNO. We describe the fixed-point processing element (PE) design and articulate how UNO integrates into a system. We then evaluate the implementation and highlight UNO's advantages.

A. Architecture

Since the UNO algorithm is fully MAC-compatible, its PE architecture is designed by extending a standard MAC unit with additional logic as highlighted in Fig. 3. The control unit determines the operating mode of the UNO PE. When performing nonlinear operations, for each cycle, the UNO PE selects the inputs to the normal MAC logic according to Eq. (3, 4) and Table II. For example, at the first cycle, $|c_{n-1}|$, $|c_n|$ and *var* are fed to the MAC unit, and mac_{i-1} instead of $|c_n|$ is selected according to Eq. (4) before the last cycle, at which *offset*, mac_n and *scale* are used as in Eq. (3). The absolute values of coefficients are pre-stored and indexed at run-time. Then *scale*, *offset* and *var* are calculated with addition, shift or tiny LUTs. In this implementation, addition is used for the calculation of *var*. Though extra additions are used in *scale* and *var* for *exp* compared to *div* and *log*, they are not mandatory since a_{exp} is fixed to 0, as discussed in Section V. Shift is used in *scale* for *div* [11] and x_{norm} for *div* and *log*. Finally, LUTs are used for *scale* of *exp* and *offset* of *log*. UNO LUTs require much fewer entries compared to 2,048 in [3], as they feed on the much fewer integer bits in *exp* and the shift offset in *log*. Furthermore, as *scale* and *offset* are merely used in the last cycle, the same LUT can be queried in series by multiple operations, further reducing the overhead.

With the proposed architecture, integrating UNO in a system can be done with minimal architecting and programming efforts. First, as UNO is an enhancement to a MAC unit, each MAC unit in the conventional MAC array can be simply extended with the black blocks as in Fig. 3 to support parallel nonlinear operations. The parallelism of nonlinear operations, i.e., the throughput of nonlinear operations, is directly determined by the ratio of UNO-enhanced MAC units to the array size and the correspondent cycle count for each nonlinear operation. Second, UNO provides programming flexibility due to the indigenous dynamic accuracy-energy trade-off. As

Table III
Comparison of UNO hardware with baseline.

Design		Accu. (%)	Delay (ns)	Area (μm^2)	Power (mW)	Energy (pJ/op)
Baseline (MAC+SAADI *div*+SECO *exp*+NN *log*)	*div*	99.93	15.73	3,068	1.09	17.22
	exp	99.70	17.50	1,118	0.22	3.73
	log	98.79	50.66	3,578	0.48	24.81
	Total	—	—	9,323	2.35	—
UNO	*div*	99.92	14.04	—	—	13.10
	exp	99.91	11.70	—	—	10.92
	log	99.15	16.38	—	—	15.29
	Total	—	—	4,221	0.93	—

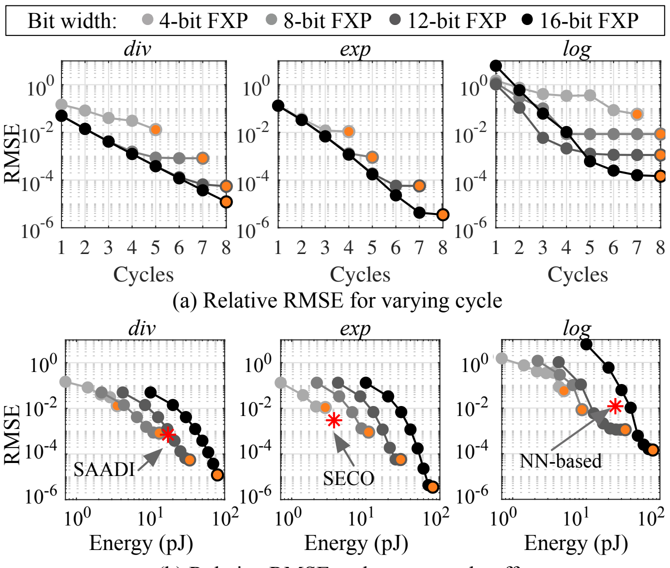
indicated by the Horner's rule, UNO algorithm starts from the high order Taylor terms to low till completion, implying the total cycle count is pre-defined. And such pre-definition can be programmed in a layer-wise manner for simplicity.

The UNO PE illustrated in Fig. 3 is synthesized using TSMC 45 nm technology at 400 MHz, and the hardware implementation result is shown in Table III. Due to the rarity of a prior design targeting multiple nonlinear operations, we intentionally set up a baseline PE to 1) exhibit identical functionality and 2) maximally support dynamic run-time accuracy-energy scaling as in the proposed UNO PE. As such, the baseline PE consists of a MAC unit (not shown in the table), a SAADI *div* [11] unit, a SECO *exp* [12] unit, and a neural network (NN)-based *log* [13] unit. Among these nonlinear units, both SAADI *div* and SECO *exp* support dynamic run-time accuracy-energy scaling using Taylor approximation, while NN *log* does not.

The comparison between baseline and UNO PEs are twofold. First, for individual nonlinear operations, UNO achieves comparable or even better final accuracy for each operation, and so for area. The accuracy here is the root mean square error among all possible, uniformly distributed inputs, as described in Section V. In the baseline PE, SAADI *div*, SECO *exp* and NN *log* take inputs with bit width of 8, 8 and 7, while our UNO implementation uses 8 bits for all operations. The minor variance in bit width does not incur game-changing difference in accuracy, area and power between the baseline and UNO PEs. The resultant energy consumption of *div* and *log* using UNO is reduced by 23.9% and 38.4% compared to the baseline, respectively. One exception is that SECO *exp* [12] has better area and power numbers due to employing approximate multiplier additionally, which can also boost the area and power efficiency of UNO up to a similar level once deployed. Second, for the entire PE, the total area and power of the baseline are the sum of those in all involved units for MAC, *div*, *exp* and *log*. We observe that the UNO PE shows around 54.7% area and 60.4% power reduction over the baseline PE. Thus, adopting UNO arrays in a system instead of normal MAC arrays has significant area, power and energy advantages over past works, which do not leverage the existing MAC units.

V. Dynamic Run-time Accuracy-Energy Scaling

Before evaluating the accuracy-energy scaling of our fixed-point UNO PE, we first identify the best Taylor expansion points. Assuming varying input distributions as in [12], we construct four different inputs between 0 and 1: one uniform distribution, $\mathcal{U}(0.0, 1.0)$, and three normal distributions with different centers, $\mathcal{N}(0.25, 0.1)$, $\mathcal{N}(0.50, 0.1)$, and $\mathcal{N}(0.75, 0.1)$.



(a) Relative RMSE for varying cycle

(b) Relative RMSE and energy trade-off

Figure 4. (a) Relative RMSE of 4-, 8-, 12-, and 16-bit fixed point (FXP) UNO for varying cycle. (b) Energy-accuracy trade-off comparison to SAADI-*div*, SECO-*exp*, and NN-*log*.

We use *relative* root mean square error (RMSE ranging within [0,1]) as the error metric. The best expansion points for *div*, *exp* and *log* based on UNO with 8-bit fraction are identified as 0.75, 0.00 and 0.75, respectively. These points minimize the overall RMSE and are used even when the bit width varies.

We then examine *the relationship between cycle count and accuracy, as well as between energy and accuracy,* for each nonlinear operation. Fig. 4(a) presents the trend of RMSE when increasing the cycle count of a UNO operation with input $\mathcal{U}(0.0, 1.0)$. Each dot on the curve represents the simulated RMSE at that cycle. Given 8 as maximum cycle count, the best RMSE dot is highlighted in orange for each bit width. Both monotonic accuracy-latency scaling at run time and accuracy-bitwidth trade-off at design time are achieved in UNO, while the former is unavailable in [12] due to approximated coefficients. Fig. 4(b) further presents the accuracy-energy scaling. Similarly, energy consumption is monotonically increasing as accuracy increases. Additionally, the accuracy-energy points for prior works in Table III are also stared. We observe that UNO, based on algorithm-level scheduling, can provide comparable or even better accuracy-energy scaling than others, except for SECO *exp*, which applies more aggressive approximation.

VI. Emerging DNN Models using UNO

Currently, there rarely exists architecture exploration into emerging nonlinearity-intensive DNNs, especially considering both the compatibility of multiple nonlinear operations and dynamic run-time accuracy-energy scaling enabled in UNO. Therefore, the hardware efficiency evaluation on SIMD arrays will be analytically performed using the baseline and UNO PEs in Table III, using their original models and datasets.

A. Emerging DNN Models

Capsule Neural Network (CapsNet) [8] contains four types of layers, namely normal convolutional layer, primary capsule layer, convolutional capsule layer and class capsule layer. The

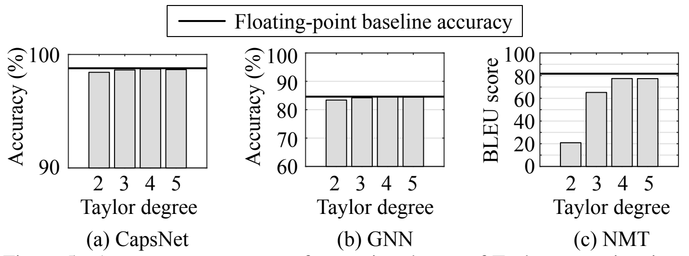


Figure 5. Accuracy measurement for varying degree of Taylor approximation. The correspondent cycle count equals the degree plus 1 as in Section III.

normal convolutional layer performs MAC operations. The primary capsule layer contains *sigmoid*, while both convolutional and class capsule layers involves the Expectation-Maximization (EM) algorithm, which covers *div*, *log*, *sigmoid* and *softmax*. The floating-point model, based on the original implementation in [8], is evaluated on the MNIST dataset. The 16-bit fixed-point UNO model is obtained by directly replacing the operations in the floating-point model with no extra training involved. Note that all UNO models in this work are achieved by operation replacement with no training.

Graph Neural Network (GNN) [9], more specifically, the graph attention network [24] is evaluated in this work. Each layer in the network consumes nonlinear operations, including *softmax*, *exponential linear unit*, and *log_softmax* for the attention mechanism and output activation. Following the implementation suggested by Velivckovic et al. [24], the model has 8 head attention layers, each with 8 hidden units. The model is trained on the Cora dataset [25], which contains 2708 nodes, 5429 edges, 7 classes, and 1433 features, for document classification. The sample count for training, validation, and testing are 140, 300, and 1000, respectively. For inference, UNO accepts 16-bit fixed-point data.

Neural Machine Translation (NMT) [10], using Seq2Seq learning with attention, involves nonlinear operations in its three major components: encoder, decoder, and attention layers. First, both the encoder and decoder layers are constructed on recurrent neural network (RNN) cells such as LSTM, involving nonlinear operations like *sigmoid* and *tanh*. Then attention weights are regulated by *softmax*. Finally, *log_softmax*, performing *softmax* on the data after the *log*, is used at the output. We build a NMT model as in [26], where the 2-layer encoder has total 46 LSTM cells, and the 2-layer decoder has 48. We train a floating-point model with 110,000 English-French sentence pairs and test with 2,000 unseen pairs. The fixed-point UNO model also applies 16-bit data.

B. Accuracy and Efficiency Evaluation

a) ***Accuracy****:* We evaluate the inference accuracy of three UNO models by comparing it with the floating-point baseline individually using PyTorch. Top-1 accuracy is used for two classification tasks, i.e., CapsNet and GNN. For NMT, we measure the bilingual evaluation understudy (BLEU) score, ranging from 0 to 100. A higher BLEU score indicates more similar translation to a human translator. Fig. 5 shows the results for three models, respectively. As plotted in Fig. 5 (a) and (b), for CapsNet/GNN, UNO achieves minimal accuracy loss (0.35%/1.42%) even only with a Taylor degree of two, compared to floating-point accuracy (98.78%/84.60%). While with an increasing degree, UNO saturates at the degree of four, at which UNO's accuracy is 98.70%/84.60% for CapsNet/GNN. Fig. 5 (c) shows that the fixed-point BLEU score increases with the Taylor degree and saturates at 77.5 at the degree of four. Due to the different types of task and accuracy metric, the accuracy gap between UNO and floating-point baseline is relatively more noticeable when using fewer Taylor degrees for NMT compared to classification by CapsNet/GNN.

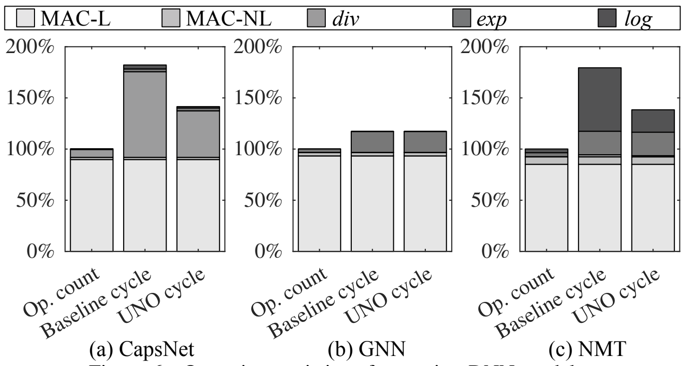


Figure 6. Operation statistics of emerging DNN models.

To sum up, UNO can achieve minor accuracy loss for diverse emerging models. When measured by classification accuracy, UNO maintains a good accuracy with heavy approximation by a Taylor degree of two; while evaluated on the NMT task, UNO attains good translation adequacy after the degree of two. The inference accuracy using UNO can be further boosted by fine-tuning the integer and fraction bit widths of nonlinear operations in each layer and UNO-aware training.

b) ***Efficiency****:* We perform an analytical evaluation on hardware area, power, throughput and energy efficiency of SIMD arrays using the baseline and UNO PE in Table III, corresponding to designs in Fig. 2(a) and (b). As in Fig. 2(a), the baseline SIMD array is split into linear MAC and nonlinear arrays, and the nonlinear array contains SIMD arrays of *div*, *exp* and *log* units, each having identical SIMD size as the MAC array. Following are specific hedges. ① The computing kernel, excluding the data SRAM in Fig. 2, with SIMD size of 64, data width of 16 (8 bits for fraction) is synthesized with TSMC 45 nm technology at 400 MHz. ② The count of each operation is proportional to its runtime, as the utilization of the baseline and UNO SIMD array for linear operations is set to be 50%, which resides reasonably in $[20\%, 95\%]$ as reported by other actual accelerators [1], [6], [7], while the nonlinear utilization is set to 80% due to their nearly elementwise behavior. We categorize operations as in Fig. 6, where MAC operations have two sources. MAC-L denotes MAC operations in linear matrix multiplication/convolution, while other operations, namely MAC-NL, *div*, *exp* and *log*, are obtained by decomposing nonlinear operations in Table I accordingly.

Based on above categorization, the per-model operation statistics in percentage are stacked in Fig. 6. For each model, the actual operation count and the cycle count executed on one individual baseline and UNO PE (100% MAC utilization), normalized to the actual total operation count, are presented.

In terms of the actual operation count, we observe that 1) each nonlinear operation occupies varying ratios in different models, and 2) each model requires varying ratio of different nonlinear operations. CapsNet contains 7.6% *div*, GNN has 3.4% *exp*, while NMT requires 3.8% *exp* and 3.7% *log*. Note that unmentioned nonlinear operations in each model count for less than 1.0%. Then the cycle count on baseline and UNO are compared to the actual operation count directly, reflecting the model execution efficiency. The cycle count for individual *div/exp/log* is 11/6/17 according to [11]–[13] on baseline and 6/6/6 on UNO as in Fig. 5, with 1 cycle for both MAC-L and MAC-NL. A similar gap between operation count and runtime as in Fig. 1 can be seen for all models. Compared with baseline, UNO saves total runtime by 22.23%/0.01%/22.89% for CapsNet/GNN/NMT. The marginal runtime reduction for GNN is due to the fact that GNN involves mostly *exp*, which takes the same cycle count in the baseline and UNO.

Table IV
HARDWARE EFFICIENCY OF DNN MODELS.

	Model	**Baseline**	**UNO**	**Increase** (%)
Area (mm^2)	—	0.659	0.283	-57.0
Power (mW)	—	205.5	66.5	-67.6
Throughput (samples/s)	CapsNet	470.3	567.1	+20.6
	GNN	6.0	6.0	+0.0
	NMT	132.5	160.6	+21.2
Energy Efficiency (samples/J)	CapsNet	2288/4648	8527/10953	+272.7/+135.6
	GNN	29/68	91/116	+209.1/+70.3
	NMT	645/1534	2415/3036	+274.5/+97.9

With the cycle count analysis, we compare the hardware performance in Table IV, where power represents the maximum power without power gating and energy efficiency includes the numbers for without/with power gating on the left/right, respectively. Compared to the baseline, UNO reduces both the area and power by over half. 2.0%, 16.4%, 6.2% and 75.4% of total baseline-based area are occupied by adder tree, MAC array, DeMUX/MUX and nonlinear array, while 4.7% and 95.3% of total UNO-based area is consumed by adder tree and UNO array, indicating a higher computing kernel density. Without power gating, the throughput of both CapsNet and NMT increases by 20% with UNO, while that of GNN remains the same, as *exp* dominates the GNN nonlinear operation, as elaborated previously. UNO nearly triples the energy efficiency for all evaluated models, as a joint effect of both shorter runtime and lower power consumption.

c) ***Influence of manufacturing techniques:*** The analytical evaluation above implies the upper bound of our proposal. However, physical design techniques, e.g., power gating, can be applied to mitigate the gap between baseline and UNO. With power gating enabled, only the relevant hardware for the operation consumes energy. For the baseline array, individual computing units are always disabled when not used while it's the *scale*, *offset* and *var* logic in UNO-based design to be powered off selectively. Note that the adder tree and DeMUX/MUX logic is constantly running. Assuming no overheads for power gating, the area benefit of UNO remains, and the energy benefit of the UNO-based design slightly decreases according to the right-hand numbers from energy efficiency in Table IV, but still almost doubles that of baseline. Such decreases come from the large ratio of nonlinear hardware in baseline. Furthermore, with power gating, even for simpler CNNs, UNO will not fall short significantly in energy efficiency, implying the potential of UNO for broader use cases.

VII. CONCLUSION

We present UNO, a unified architecture that virtualizes nonlinear operations on top of conventional DNN hardware to minimize additional hardware costs. UNO is designed to be compatible with MAC operations to fully utilize off-the-shelf MAC units and flexibly provide dynamic run-time accuracy-energy scaling, thus reducing the area and energy overheads for nonlinear operations. Simulation results on both individual operations and emerging neural network models indicate that UNO is suitable for sophisticated neural network models without a significant loss of accuracy. To conclude, UNO can be integrated onto existing systems seamlessly for future machine learning applications with promising area and energy benefits.

REFERENCES

[1] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017.
[2] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *ICML*, 2010.
[3] S. Han *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *FPGA*, 2017.
[4] S. Wang *et al.*, "C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs," in *FPGA*, 2018.
[5] A. Marchisio *et al.*, "CapsAcc: An Efficient Hardware Accelerator for CapsuleNets with Data Reuse," in *DATE*, 2019.
[6] H. Kwon *et al.*, "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects," in *ASPLOS*, 2018.
[7] A. Yazdanbakhsh *et al.*, "GANAX: A Unified MIMD-SIMD Acceleration for Generative Adversarial Networks," in *ISCA*, 2018.
[8] G. E. Hinton *et al.*, "Matrix Capsules with EM Routing," in *ICLR*, 2018.
[9] Z. Zhang *et al.*, "Deep learning on graphs: A survey," *TKDE*, 2020.
[10] D. Bahdanau *et al.*, "Neural machine translation by jointly learning to align and translate," in *ICLR*, 2015.
[11] S. Behroozi *et al.*, "SAADI: A Scalable Accuracy Approximate Divider for Dynamic Energy-quality Scaling," in *ASPDAC*, 2019.
[12] D. Wu *et al.*, "SECO: A scalable accuracy approximate exponential function via cross-layer optimization," in *ISLPED*, 2019.
[13] S. Eldridge *et al.*, "Neural Network-based Accelerators for Transcendental Function Approximation," in *GLVLSI*, 2014.
[14] H. Saadat *et al.*, "Approximate Integer and Floating-Point Dividers with Near-Zero Error Bias," in *DAC*, 2019.
[15] T. Kemp *et al.*, "MIPAC: Dynamic Input-Aware Accuracy Control for Dynamic Auto-Tuning of Iterative Approximate Computing," in *ASPDAC*, 2021.
[16] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation.* Wiley, 1999.
[17] V. Gupta *et al.*, "IMPACT: Imprecise adders for low-power approximate computing," in *ISLPED*, 2011.
[18] S. Hashemi *et al.*, "DRUM: A dynamic range unbiased multiplier for approximate applications," in *ICCAD*, 2015.
[19] B. Y., "Efficient Hardware Architecture of Softmax Layer in Deep Neural Network," in *SOCC*, 2016.
[20] Aurangzeb and R. Eigenmann, "HiPA: History-based piecewise approximation for functions," in *ICS*, 2017.
[21] D. Wu *et al.*, "uGEMM: Unary Computing Architecture for GEMM Applications," in *ISCA*, 2020.
[22] J. Rust and S. Paul, "Exploiting special-purpose function approximation for hardware-efficient qr-decomposition," in *DATE*, 2017.
[23] S. Chevillard *et al.*, "Sollya: An environment for the development of numerical codes," in *ICMS*, 2010.
[24] P. Veličković *et al.*, "Graph Attention Networks," in *ICLR*, 2018.
[25] A. K. McCallum *et al.*, "Automating The Construction of Internet Portals with Machine Learning," *Information Retrieval*, 2000.
[26] PyTorch, "Translation with a Sequence to Sequence Network and Attention," 2019. [Online]. Available: $https : //pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html$