Hardware-aware Softmax Approximation for Deep Neural Networks

Xue Geng[§*], Jie Lin[§*], Bin Zhao[♮], Anmin Kong[§], Mohamed M. Sabry Aly[♯], Vijay Chandrasekhar[§]

[§]I^{2}R, A*STAR, Singapore [♮]IME, A*STAR, Singapore
[♯]School of CSE, NTU, Singapore

Abstract. There has been a rapid development of custom hardware for accelerating the inference speed of deep neural networks (DNNs), by explicitly incorporating hardware metrics (*e.g.*, area and energy) as additional constraints, in addition to application accuracy. Recent efforts mainly focused on linear functions (matrix multiplication) in convolutional (Conv) or fully connected (FC) layers, while there is no publicly available study on optimizing the inference of non-linear functions in DNNs, with hardware constraints.

In this paper, we address the problem of cost-efficient inference for *Softmax*, a popular non-linear function in DNNs. We introduce a **hardware-aware linear approximation** framework by algorithm and hardware co-optimization, with the goal of minimizing the cost in terms of area and energy, without incurring significant loss in application accuracy. This is achieved by simultaneously reducing the operand bit-width and approximating cost-intensive operations in Softmax (*e.g.* exponential and division) with cost-effective operations (*e.g.* addition and bit shifts). We designed and synthesized a hardware unit for our approximation approach, to estimate the area and energy consumption. In addition, we introduce a training method to further save area and energy cost, by reduced precision. Our approach reduces area cost by 13× and energy consumption by 2× with 11-bit operand width, compared to baseline at 19-bit for VOC2007 dataset in Faster R-CNN*.

Keywords: Softmax · Nonlinear Operation · Power · Area.

1 Introduction

Modern deep neural networks (DNNs) have achieved remarkable success in a broad of computer vision applications including image classification, object detection and instance segmentation, at the cost of huge amount of weights and activations. For instance, AlexNet-7 [19] has about 60M parameters, VGG-16 [27] 138M and ResNet-101 [14] 45M. There has been a rapid development of custom hardware for accelerating inference of DNNs at scale, such as Application-Specific Integrated Circuits (ASICs) - EIE [11], Eyeriss [5] and Google Tensor

[*]Both authors contributed equally to this work.

Processing Unit (TPU) [18]. On the other hand, plenty of work has been proposed to save hardware resources in terms of chip area and power, by reducing precision of operands (*e.g.*, DoReFa-Net [30], BinaryNet [6], XNOR-Net [25] and ternary quantization [31]), or reducing number of matrix multiplication operations (*e.g.*, compact networks like SqueezeNet [17] and MobileNet [16], or network pruning [12, 15]). This in turn enables higher inference throughput, given limited hardware resources. For example, TPU [18] performs 8-bit integer multiplication for Conv and FC layers, enabling order of magnitude reduction in energy consumption. EIE [11] supports higher inference rate by pruning weights in FC layers, with lower energy and area cost.

However, these work mainly focused on linear matrix operations in Conv and FC layers which account for over 99% of total operations in modern DNNs [29]. To the best of our knowledge, no public study is available on optimizing the inference of non-linear blocks (*e.g.*, Softmax and Sigmoid) in DNNs with hardware constraints. Compared to linear operations in Conv and FC, nonlinear blocks pose unique challenges for hardware implementation. First, a Conv operation using a multiplier and an adder, is much cheaper than nonlinear functions like exponential, which usually requires either a complex hardware unit or an iterative routine in programmable hardware. Second, in current hardware accelerators, the compute ratio per unit area for nonlinear blocks is order of magnitude lower than that for Conv and FC layers, resulting in inefficient use of the chip area (*e.g.*, 20% of the total area allocated for compute operations is used by nonlinear unit, which only account for less than 1% of total operations in DNNs).

In this paper, we aim to address the problem of cost-efficient inference for *Softmax*, a popular non-linear function in a wide range of network architectures. We make the following contributions.

– We propose a framework of **hardware-aware linear approximation** for Softmax, with the goal of minimizing the design cost in terms of area and energy consumption, without incurring significant loss in application accuracy. The framework simultaneously reduces the operand bit-width and approximates cost-intensive operations in Softmax (*e.g.* exponential and division) with cost-effective operations (*e.g.*, addition and bit shifts).
– We develop a training approach to further save the chip area and energy cost with aggressively reduced bit-width of Softmax operands, by clipping Softmax input in a small range.
– We design and synthesize a hardware unit for our approximation approach to estimate area and energy costs. We present a comprehensive analysis of the impact of operand bit-width and operations on performance accuracy and hardware metrics. With comparable object detection and instance segmentation accuracy, our approach reduces area cost by 13$\times$ and energy cost by 2$\times$ with 11-bit operand width, compared to baseline at 19-bit.

We argue that our approach is not explicit to Softmax function. Many other nonlinear blocks in DNNs such as $Tanh, Sigmoid$ can leverage our linear approximation approach. For instance, several of these nonlinear functions include

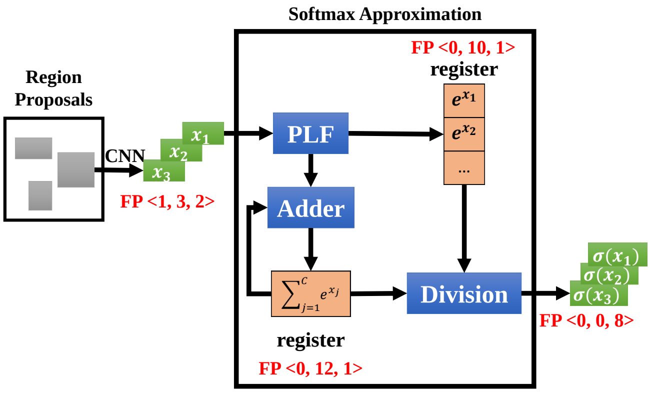


Fig. 1: The overall architecture for hardware-aware Softmax approximation. PLF - Piecewise Linear Function. FP - Fixed-point. Numbers in point bracket represent sign bit, integer bits and fractional bits respectively.

exponential operation, rendering our approach a natural fit for approximating them at lower hardware cost.

The paper is organized as follows. Section 2 describes background, motivation and overview of our approach. In Section 3, we explore the search space of operand bit-widths. Section 4 introduces several cost-effective operations for approximating Softmax. Section 5 presents a training approach for further reduction of hardware cost. We evaluate the performance in Section 6. Section 7 and Section 8 are related work, conclusions and future work.

2 Background and Motivation

Softmax is a common building block for many DNN-based computer vision applications [13, 26, 8],

$$\sigma(x_c) = \frac{e^{x_c}}{\sum_{j=1}^{C} e^{x_j}}, \tag{1}$$

where C represents the number of classes and x is a C-dimensional input vector with arbitrary real-values. For image classification task, Softmax normalizes the C-dimensional input vector, so that all the entries add up to 1. For region-based object detection (*e.g.*, Faster R-CNN [26] and R-FCN [8]) and instance segmentation (*e.g.*, MNC [7] and Mask R-CNN [13]), Softmax generates a C-dimensional normalized confidence scores for each region proposal independently. Subsequently, for each class, the proposals are sorted by their scores, followed by non-maximum suppression (NMS) to filter overlapped ones.

Different from linear operations (*e.g.* matrix-matrix (matrix-vector) multiplication in Conv and FC layers), it is costly to implement nonlinear Softmax function in hardware. A single Conv operation is actually cheaper than Softmax or exponential. Conv uses a multiplier and an adder, while exponential requires either a complex hardware unit or an iterative routine in programmable hardware. As a result, more multipliers and adders can be placed in the same area to sustain the required CONV operations, compared to units that perform Softmax. Integrating numerous nonlinear operations, which may be required to improve the application runtime, consumes considerable hardware resources.

Take Google TPU [18] as an example, 20% of the total area allocated for computing resources is for non-linear functions – the remaining 80% is for CONV and FC operations. On the other hand, Conv (FC) layers account for over 99% of total operations in modern DNNs, versus less than 1% for nonlinear operations. One can see that the Compute Ratio Per Unit Area for nonlinear unit is 25× smaller than matrix multiply for CONV (FC). Thus, reducing the area of nonlinear unit by 10× leads to more efficient use of chip area, *e.g.*,

- faster nonlinear operation by integrating many such modules (*i.e.*, fostering parallel execution); or
- higher convolution rate by reclaiming the saved area to add more CONV hardware resources.

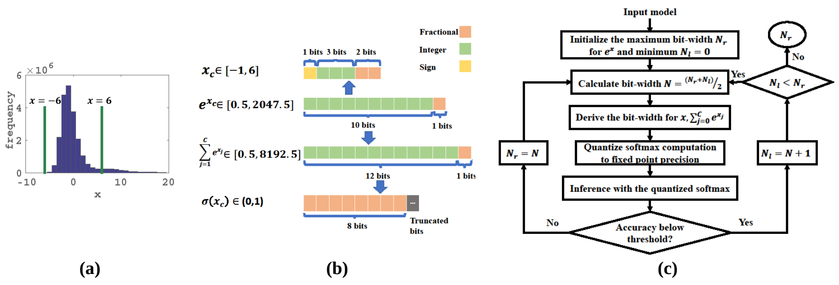
This motivates us to develop cost-efficient approximation method for Softmax in DNNs, from both algorithm and hardware perspectives. Towards lower area and energy cost, the key is to approximate Softmax with simple yet efficient linear operations - 1) approximate cost-intensive arithmetic operations (*e.g.*, exponential, division in Equation 1.) with cost-effective operations (*e.g.* addition, logical operations) and 2) minimize the bit-width of operands in the operations. Figure 1 shows the overall flow for computing Softmax in simplified hardware architecture. This architecture contains several modules including the piecewise function (PLF), accumulation (Adder) and division (Division) with respective memory registers for storing intermediate operands.

It is worthy noting that we do not consider the complex polynomial approximation methods such as Taylor series [24] for exponential function, which would introduce many multiplication operations in order to preserve high accuracy. As compared to simple arithmetic operations like addition, multiplication leads to much higher energy and area cost, especially when the operand bit-width is large. For instance, 32-bit fixed-point addition can be 30× less energy and 25× less area than 32-bit fixed-point multiplication [9].

3 Exploring the search space of operand bit-width

Representing operands in fixed-point precision with short bit-width would reduce energy consumption and area cost ref. Figure 1. Figure 2 (b) shows an example of fixed-point representations for Softmax input x_c, output $\sigma(x_c)$ and intermediate values (*i.e.* the numerator e^{x_c} and the denominator $\sum_{j=1}^{C} e^{x_j}$). Once the number of integer bits of operand e^{x_i} is fixed as N, the integer bits for input $\mathbf{x}$ can be derived as follows.

$$N_{\mathbf{x}}^{I} = \lceil \log_2(\ln 2^N) \rceil \tag{2}$$



x = −6
x = 6
frequency
x
(a)
Fractional
Integer
Sign
1 bits 3 bits 2 bits
$x_c \in [-1, 6]$
$e^{x_c} \in [0.5, 2047.5]$
10 bits
1 bits
$\sum_{j=1}^{c} e^{x_j} \in [0.5, 8192.5]$
12 bits
1 bits
$\sigma(x_c) \in (0,1)$
8 bits
Truncated bits
(b)
Input model
Initialize the maximum bit-width N_r for e^x and minimum $N_l = 0$
Calculate bit-width $N = (N_r+N_l)/2$
Derive the bit-width for $x, \sum_{j=0}^{C} e^{x_j}$
Quantize softmax computation to fixed point precision
Inference with the quantized softmax
Accuracy below threshold?
No
Yes
$N_r = N$
$N_l = N + 1$
$N_l < N_r$
Yes
No
N_r
(c)

Fig. 2: (a) Distribution of Softmax input x. (b) Example of fixed-point representations for Softmax input x_c, output $\sigma(x_c)$ and intermediate values (*i.e.* the numerator e^{x_c} and the denominator $\sum_{j=1}^{C} e^{x_j}$). (c) Flow of binary search for the minimal operand bit-width that does not cause loss in application accuracy.

As we know, the negative part of x_i will decide the fractional part of e^{x_i} [*]. It is an asymmetric boundary. It is noteworthy that theoretically the bit-width for denominator is decided by the number of additions and bit-width of e^{x_i}. To shorten its bit-width, a data-driven manner is deployed to explore the minimum value. The range of $\sigma(x_c)$ is in $(0, 1)$, thus it only requires a small number of fractional bits truncated from the number computed by the division operation.

There is a trade-off between operand bit-width and application accuracy. We introduce an accuracy-aware approach to explore the minimal operand bit-width in Softmax. Figure 2 (c) shows the flow of the minimal operand bit-width searching. It starts from an initialized integer bit N for e^{x_i}. With an input network model, each loop consists of reducing the integer bits for e^{x_i}, x_c and $\sum_{j=1}^{C} e^{x_j}$, quantizing Softmax with derived precision, followed by inference with the quantized Softmax.

4 Approximating Softmax operation

The computation of Softmax can be roughly decomposed into 3 steps: the exponential function, the accumulation in the denominator and division. In this section, we introduce various cost-efficient strategies to approximate the exponential function and division respectively (see Figure 3 (a)). Finally, we integrate different operations for approximating Softmax.

Lookup Table of e^x (LUT-EXP) To approximate e^{x_i}, a straight-forward approach is to create a lookup table (LUT) which stores the fixed-point number of e^{x_i} over a discrete subset of x_i (see Figure 3 (b)). As analyzed in Section 3, once the integer bits of fixed-point operand e^{x_i} is fixed, we can derive the number of integer bits for the fixed-point input x_i as well as its range $x \in [x_l, x_r]$ that precise to a quantum of 1 (Figure 2 (a)-(b)). This would also determine the size

[*]The fractional bits of e^{x_i} and x are fixed at small numbers (less than 2) as they contribute less to the precision, as compared to the integer bits.

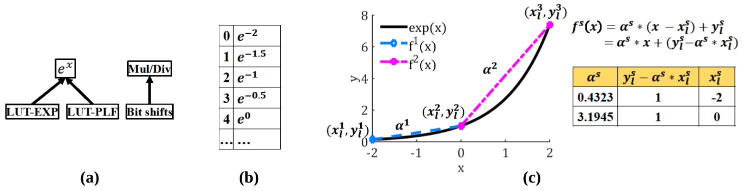


Fig. 3: (a) Approximate the exponential function and the division operation in Softmax function, respectively. (b) Lookup Table of e^x (LUT-EXP). (c) Piecewise Linear Function (LUT-PLF) for approximating e^{x_i}. Note that the entries in LUT are stored with fixed-point representations.

of LUT-EXP, *i.e.* 2^{N_x} entries computed by e^{x_i}, where x_i is uniformly sampled in $[x_l, x_r]$ with a fixed step size.

Piecewise Linear Function (LUT-PLF) One drawback of LUT-EXP is the number of entries in the table increases exponentially with the bit-width of fixed-point input x. The larger the table size, the higher energy cost for memory access and bigger area cost for storage. To alleviate this issue, an alternate approach is Piecewise Linear Function (PLF), which typically approximates non-linear function with a small number of line pieces [1, 23]. In geometry, PLF approximates e^x with S continuous pieces uniformly defined over a finite range of $x \in [x_l^1, x_r^S]$, each of those pieces is an affine function with slope α^s

$$\begin{gathered} f^s(x) = \alpha^s * (x - x_l^s) + y_l^s = \alpha^s * x + (y_l^s - \alpha^s * x_l^s) \\ x \in [x_l^s, x_r^s], \quad y_l^s = e^{x_l^s}, \quad s \in [1, S] \end{gathered} \tag{3}$$

A LUT of affine functions is built to store the value of x_l^s, α^s and the pre-computed $y_l^s - \alpha^s * x_l^s$ (see Figure 3 (c)). The number of bits for x_l^s and α^s is the same as input x, while $y_l^s - \alpha^s * x_l^s$ is the same as e^{x_i}. Given x, the entry nearest to x is chosen for computing the approximate value $f^s(x)$ in Eq 3. The cost of computing $f^s(x)$ contains 1 multiply ($\alpha^s * x$) and 1 addition. At last, the minimal number of pieces S is determined in a similar way to the data-driven binary search of minimal operand bit-width introduced in Section 3.

Implement Multiplication and Division by Bit Shifts. The multiply term $\alpha^s * x$ in Equation 3 can be simply implemented by bit shifts, via approximating α^s with the closet 2^b where b is an integer constant. Similarly, bit shifts is able to approximate the division operation in Equation 1 by replacing the denominator with 2^b. Bit shifts can largely reduce energy/ares cost induced by multiplication and division, especially when the operand bit-width is large.

Approximate Softmax as a whole. By assembling the 3 steps together, there are 6 possible combinations for Softmax approximation in total show in Table 1. And all method names appeared in text would refer to this table. In Section 6, we evaluate the performance (accuracy, energy/area cost) of Softmax approximation variants with varied operand bit-widths, on a wide range of network architectures as well as benchmarks. We observe that method Approx_D

Table 1: Method naming and definitions.

Method names	Definition
Approx_A	Exponential function is implemented by the look-up table.
Approx_B	Exponential function is implemented by the look-up table, while Division operation is implemented by bit shifting.
Approx_C	Exponential function is implemented by piecewise linear function.
Approx_D	Exponential function is implemented by piecewise linear function, in which the coefficients are power of two;
Approx_E	Exponential function is implemented by piecewise linear function, while Division operation is implemented by bit shifting.
Approx_F	Exponential function is implemented by piecewise linear function, in which the coefficients are power of two; While Division operation is implemented by bit shifting.

Algorithm 1: Training with Softmax Approximation through STE

```
1  Initialization: DNN model weights $M$, input range $(-\hat{\gamma}, \gamma)$ and number of
   pieces $S$.
2  foreach $s$ in $S$ pieces do
3      Derive $x_l^s$,$x_r^s$,
4      $y_l^s = e^{x_l^s}$, $y_r^s = e^{x_r^s}$,
5      $\alpha^s = \frac{y_r^s - y_l^s}{x_r^s - x_l^s}$.
6  end
7  foreach batch do
8      foreach $x_i$ in input vector $\boldsymbol{x}$ do
9          Find relevant piece in LUT-PLF $s^* = \underset{s}{\text{argmin}}(x_i - x_l^s)$.   $s.t., x_i > x_l^s$.
           $e^{x_i} \leftarrow y^{s^*}(x_i) = \alpha^{s^*} * (x_j - x_l^{s^*}) + y_l^{s^*}$
10     end
11     Calculate Softmax $\sigma(x_i) = \frac{y^{s^*}(x_i)}{\sum_j y^{s^*}(x_j)}$.
12     Compute gradients of Softmax layer using STE.
13     Update parameters $M$.
14 end
15 return $M$
```

achieves the optimal tradeoff between accuracy and energy/area cost, especially at low operand bit-width.

5 Training with Softmax approximation

The integration of LUT-PLF operation with the minimal operand bit-width trades off energy/area cost against application accuracy. If further reducing the operand bit-width, the energy/area cost is even lower, but there is a significant drop in accuracy. In this section, we propose a hybrid training method to simulate

Softmax approximation error in the forward pass during training, which is able to adapt the network to the approximation method and retain accuracy even aggressively reducing operand bit-width.

Clipped input for aggressive bit-width reduction. One can see from Eq 3 that the range of y_l^s depends on the range of x_l^s. For instance, to avoid the loss of range and precision when comparing with floating point number representations, the minimal number of integer bits for $e^{y_l^s}$ is 18-bit for $x_l^s \in [-12, 12]$. This number dramatically drops to 10-bit when $x_l^s \in [-6, 6]$. To reduce bit-width for lower energy/area cost, a clipping operation is operated on input x, which is bounded in an asymmetric range $[-\hat{\gamma}, \gamma]$

$$h(x) = min(max(x, -\hat{\gamma}), \gamma), \tag{4}$$

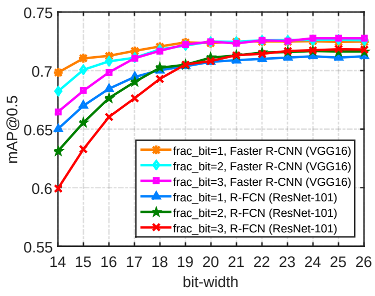
where $\hat{\gamma}$ and γ are pre-defined positive thresholds. Note that $\hat{\gamma}$ is usually smaller than γ due to the asymmetric distribution of input x (see Figure 2 (a)).

Hybrid training. One approach to compensate the loss in accuracy induced by Softmax approximation is to train the network by taking the approximation error into account. This can be accomplished by simulating the Softmax approximation in the forward pass, while the backward pass remains as usual. More specifically, the forward pass computes Softmax using the approximation method Approx_D with the clipped x as input. The hybrid training is described in Algorithm 1. As Equation 4 and Equation 3 are non-differentiable functions, we use Straight Through Estimator (STE) [2] to enable the back-propagation. Note that the hybrid training method can be used to study the effect of aggressively reducing the minimal number of pieces S in LUT-PLF.

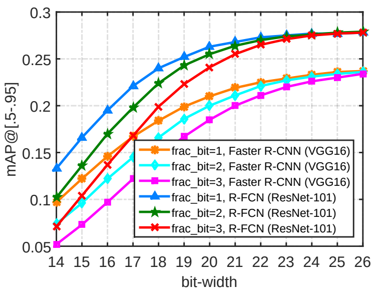
6 Experiments

We evaluate the performance on two challenging tasks: object detection and instance segmentation. First, we study the impact of operand bit-width on application performance, without approximating Softmax operation. Second, we evaluate the 6 Softmax approximation variants with floating point precision. Third, we perform the ASIC RTL simulation to measure the energy/area cost, as a function of operand bit-width and approximation operations. Finally, we perform extensive experiments on various benchmarks to verify the advantages of the training with aggressive bit-width reduction via clipped input x using Caffe on a NVIDIA Tesla V100 GPU.

Datasets For object detection, we evaluate our method on 2 standard benchmark datasets - PASCAL VOC2007 [10] and MS COCO2014 [20]. The VOC2007 dataset consists of about 5k train-val images and 5k test images over 20 object categories, while the MS COCO2014 dataset contains 80 foreground object classes and one background class. The training set has about 80K images, and 5K images for mini validation set. For instance segmentation, the models are trained on the PASCAL VOC 2012 training set, and evaluated on validation set.



(a)



(b)

Fig. 4: (a) Object detection accuracy for Faster R-CNN (VGG16) and R-FCN (ResNet-50) on VOC2007, with varied bitwidths for operand e^{x_i}, including sign bit, integer bits and fractional bits. (b) Object detection accuracy for Faster R-CNN (VGG16) and R-FCN (ResNet-101) on COCO2014, with varied bit-widths for operand e^{x_i}. The number of fractional bits of e^{x_i} is varied from $[1, 2, 3]$.

Models and Metrics For object detection, we integrate the Softmax approximation into Faster R-CNN [26] and R-FCN [8]. Following the standard protocol, we report results on VOC2007 test set using mean Average Precision (mAP) with IoU thresholds at 0.5 (denoted as mAP@0.5) and MS COCO2014 minival set using mAP averaged for IoU $\in [0.5 : 0.05 : 0.95]$ (denoted as mAP@[.5, .95]). For instance segmentation, the model MNC [7] is selection. The results are reported on VOC 2012 validation set using mAP with thresholds at 0.5 and 0.7 (denoted as mAP@0.5 and mAP@0.7).

6.1 Impact of operand bit-width

Figure 4a and Figure 4b evaluate the impact of operand bit-width (for e^{x_i}) on object detection, in terms of mAP@0.5 and mAP@[.5, .95] for VOC2007 and COCO2014 respectively. Besides, we also study the effect of the fractional bits for the fixed point of e^{x_i}. For clarity, we do not approximate Softmax operation when evaluating operand bit-width. We make the following observations - 1) the detection accuracy increases with the number of operand bit-width. 2) when the bit-width is at low range (*e.g.*, 15), the model with less fractional bits performs better. 3) when the bit-width is at large range (*e.g.*, 23), the model with more fractional bits performs slightly better. 4) for Faster-RCNN (VGG16), the bit-widths with comparable accuracy on VOC2007 and COCO2014 are different (the latter is larger). The possible reason is when the dataset becomes more challenging (*e.g.*, the number of targeted classes increases), the model would require more bits to preserve the application accuracy. 5) to maintain the detection accuracy, Faster-RCNN (VGG16) requires 19 bits (18 bits for integer part and 1 bit for fractional part), while R-FCN (ResNet-50) 22 bits, due to the fact that the distribution of input x from ResNet-50 is wider than that from VGG16.

Table 2: Object detection accuracy on VOC2007 and COCO2014 and instance segmentation accuracy on VOC 2012, with the 6 Softmax approximation variants.

Methods		min. table size	VOC2007 mAP@0.5		COCO2014 mAP@[.5-.95]		VOC2012 mAP@0.5
e^{x_i}	$\sum e^x$		Faster R-CNN	R-FCN (ResNet50)	Faster R-CNN	R-FCN (ResNet101)	MNC (VGG16)
LUT-PLF	2^n	32	69.84	70.31	21.30	26.50	63.75
	/	32	72.30	71.69	24.10	28.10	64.65
LUT-PLF-B	2^n	32	68.12	68.86	20.80	25.10	64.27
	/	32	72.06	71.37	23.70	27.90	62.96
LUT-EXP	2^n	128	69.95	69.82	20.20	25.50	62.52
	/	128	72.68	71.64	24.10	27.40	64.40
e^{x_i}	/	-	72.52	71.76	24.20	28.20	63.50

6.2 Evaluations on Softmax approximation variants

Table 2 reports object detection accuracy on VOC2007 and COCO2014, and instance segmentation accuracy on VOC 2012, with the 6 Softmax approximation variants. At inference stage, we replace the exponential function in Softmax layer with either LUT-PLF or LUT-EXP, and the Division operation with bit shifts. The minimal LUT size for each Softmax approximation is derived by using the binary search algorithm described in Section 3. All experiments are performed in floating point precision.

We observe that a) all LUT based approaches perform comparable to Softmax (the last row) in terms of accuracy, across different datasets and tasks; b) bit shifts for approximating the division operation introduces considerable loss in accuracy, while it works well for approximating the multiplication in LUT-PLF (*i.e.* LUT-PLF-B). This might be because the bit shifts for division brings in higher approximation error, and c) the table size of 4 times smaller than LUT-EXP, while with comparable accuracy.

6.3 Tradeoff between Energy/Area Cost and Accuracy

Hardware evaluation methodology We design ASIC RTL to simulate the Softmax approximation algorithms and measure the energy and area cost. All approaches are written in Verilog and is kept consistent with the algorithm model. We adopted logic gates to store LUT, because using memory to store lookup table is not efficient for the silicon area if table size is less than $1k$ entries. Registers are deployed for temporary storage of exponential value. The Cadence RTL compiler is deployed for synthesis with UMC 65nm standard cell library. The power and area consumption are estimated with synthesis result under the condition of 500MHz clock Synthesizers.

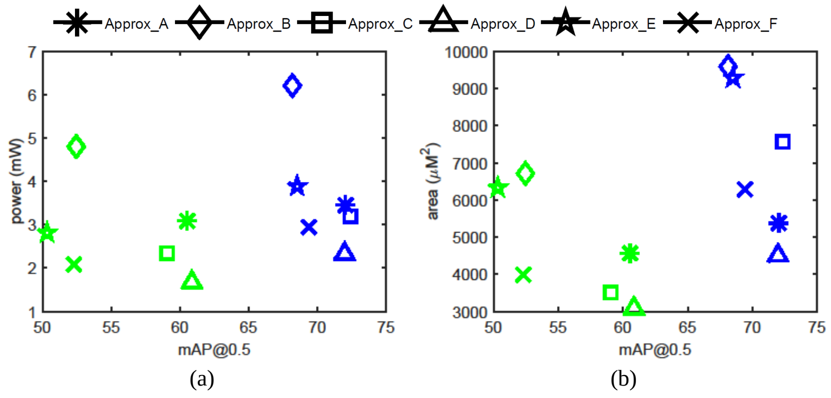


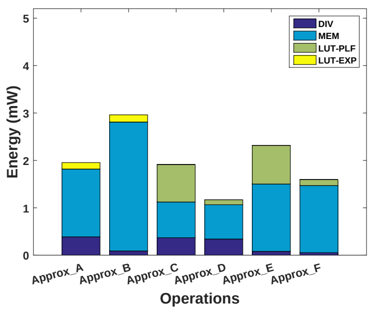
Fig. 5: Comparisons of 6 Softmax approximation variants with 2 operand bit-widths for Faster-RCNN (VGG16) model. (a) Trade-off between accuracy and power; and (b) Trade-off between accuracy and area. **Green** color represents 11-bit width for operand e^{x_i}, **Blue** 19-bit.

Energy and Area Cost Figure 5 reports energy and area cost with 2 operand bit-widths (11-bit and 19-bit) for e^{x_i}, for Faster-RCNN (VGG16) on VOC2007. We evaluate all the Softmax approximation variants. One can observe that a) the energy/area cost is reduced as bit-width increases, b) in most cases, the performance trend of area cost is consistent with energy cost, c) among the 6 Softmax approximation variants, Approx_D achieves the optimal tradeoff between accuracy and energy/area cost, compared to the rest, d) Approx_E and Approx_B performs the worst in terms of energy and area cost, as the search of n that 2^n is closet to the summation value in denominator costs considerable power.

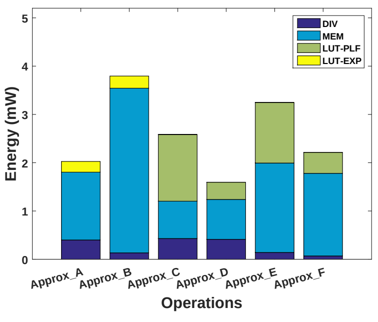
Energy Breakdown Figure 6 further studies the energy breakdown into cost-intensive operations including memory access, division and LUT-PLF (or LUT-EXP), for the 6 Softmax approximation variants. We observe that a) memory access, LUT-PLF and division consume most of energy during inference; b) the energy consumption of all operations increases with bit-width, and memory access is the one with the most significant increase; c) the shift operation in LUT-PLF-B saves some energy cost, compared to LUT-PLF and d) DIV-B costs more power as compared to original DIV. It is because that memory access increases when it dynamically computes the closest value to the denominator.

6.4 Evaluations on Clipped Training

We evaluate the aggressive bit-width reduction with or without training using the clipped input x (*i.e.* Bounded Approx_D), and compare to the best Softmax approximation (*i.e.* Approx_D). Results are reported on 3 benchmarks for both object detection and instance segmentation, with the minimal number of LUT



(a)



(b)

Fig. 6: Energy breakdown into cost-intensive operations on operand e^{x_i}. (a) 11-bit width; and b) 19-bit width. There are 4 operations being evaluated, including division (DIV), memory access (MEM), LUT-PLF and LUT-EXP.

Table 3: Performance of aggressive bit-width reduction with or without training, in terms of accuracy and energy/area cost, for Faster R-CNN (VGG16) on VOC2007. $*$ denotes the measurements of area and energy are performed on the approximation of e^{x_i} only.

Methods	mAP @0.5	min. table size	bit-width				*area** (μm^2)	power* (mW)
			x_i	e^{x_i}	$\sum_{j=1}^{C} e^{x_j}$	$\sigma(x_i)$		
Clip ($\gamma = 12$)	71.98	6	(1,4,2)	(0,18,1)	(0,20,1)	(0,0,8)	262	0.116
Clip ($\gamma = 6$)	54.18	2	(1,3,2)	(0,10,1)	(0,12,1)	(0,0,8)	19	0.067
Clip + Train ($\gamma = 6$)	70.58	2	(1,3,2)	(0,10,1)	(0,12,1)	(0,0,8)	19	0.067

size as usual. The numbers in brackets represent the number of sign bit, integer bits and fractional bits for fixed-point number representations, respectively. It is worthy noting that the measurements of area and energy are performed on the approximation of e^{x_i} only, in order to fairly evaluate the advantages of aggressive bit-width reduction with clipped input (Clip) in Approx_D.

Table 3 reports the detection accuracy and energy/area on VOC2007. We observe that the Clip with training requires 2-3 times fewer LUT size to achieve comparable accuracy with the best baseline. The benefits of training on clipped input x is two-fold: a) it leads to less area and energy (2×-70×), and b) the accuracy is dramatically improved compared to the aggressive bit-width reduction without training. These conclusions are consistent with the observations from Table 4 and Table 5. The number of minimal LUT size is varied for different benchmarks. For instance, VOC2007 requires 6 entries to preserve accuracy while COCO2014 demands more. One may note that Clip + Train dropped about 5% in mAP on COCO2014, suggesting that the distribution of exponential value

Table 4: Performance of aggressive bit-width reduction with or without training, in terms of energy and area cost, for R-FCN (ResNet-101) on COCO2014.

Methods	mAP @[.5-.95]	min. table size	bit-width				*area** (µm^2)	power* (mW)
			x_i	e^{x_i}	$\sum_{j=1}^{C} e^{x_j}$	$\sigma(x_i)$		
Clip ($\gamma = 16$)	27.50	8	(1,5,2)	(0,24,1)	(0,28,1)	(0,0,8)	1020	0.598
Clip ($\gamma = 6$)	4.90	4	(1,3,2)	(0,10,1)	(0,14,1)	(0,0,8)	38	0.086
Clip + Train ($\gamma = 6$)	22.60	4	(1,3,2)	(0,10,1)	(0,14,1)	(0,0,8)	38	0.086

Table 5: Performance of aggressive bit-width reduction with or without training, in terms of accuracy and energy and area cost, for MNC (VGG16) on VOC 2012.

Methods	mAP @0.5	mAP @0.7	min. table size	bit-width				*area** (µm^2)	power* (mW)
				x_i	e^{x_i}	$\sum_{j=1}^{C} e^{x_j}$	$\sigma(x_i)$		
Clip ($\gamma = 34$)	63.51	43.93	8	(1,5,2)	(0,23,1)	(0,26,1)	(0,0,8)	1049	0.641
Clip ($\gamma = 6$)	50.37	34.52	4	(1,3,2)	(0,10,1)	(0,13,1)	(0,0,8)	43	0.102
Clip + Train ($\gamma = 6$)	62.96	42.18	4	(1,3,2)	(0,10,1)	(0,13,1)	(0,0,8)	43	0.102

tends to be diverse with challenging datasets and complex tasks. Finally, results from all 3 tables show that the Softmax approximation approach generalize well in the wide range of tasks, datasets and models.

6.5 Discussion

In our experiments, we did not use the original softmax as a baseline as it would cost more energy. To demonstrate this, we conduct a software-based experiment. We measured the power consumption of softmax operation using a C-based implementation on CPUs using the Intel performance counter monitor toolset [28]. We find that such software-based softmax implementation consumes more than $10^3\times$ higher power (12.80W on CPU vs. 3.78mW with 19-bit Approx_A approximation in Table 1) than the baseline we consider in our work.

7 Related Work

Rectified linear unit (ReLU). A noteworthy example of function approximation in modern DNNs is ReLU, which uses piecewise linear function to replace saturated activation functions (*e.g. sigmoid* and *tanh*) [22]. ReLU is composed of 2 discontinuous pieces, the first assigns 0 to negative part of input features and the second retains the positive part. ReLU is to alleviate the vanishing gradient problem and accelerate the convergence speed during training, while our objective in this work is to design hardware-aware Softmax approximation that minimizes energy/area cost required during inference.

Softmax approximation. In natural language modeling (*e.g.* word embedding), many Softmax approximation approaches [4] have been proposed to reduce the high complexity of computing Softmax, as the number of Softmax computation is proportional to the vocabulary size with typically more than million words. One group of work is to shortlist a subset of frequent words, thus avoid computing the normalization term over all words in the denominator of Softmax function (see Equation 1). Examples include hierarchical tree structured Softmax [21] and differentiated Softmax [4]. Another group is to skip the expensive computation of normalization term by directly approximating it with sampling techniques like Monte-Carlo method [3].

Our work differs from these work in three-fold. (1) Objective - Most of these Softmax approximations are designed to accelerate neural network training only. Our approach is able to approximate Softmax for both inference and training. (2) Methodology - These Softmax approximation methods still compute the nonlinear Softmax function, while our method directly approximate the nonlinear function with cost-efficient linear operations in fixed-point precision, from a hardware perspective. (3) Task - Instead of language modeling, we target computer vision applications. Therefore, the design of approximation method requires different domain knowledge, for instance, hierarchical Softmax [21] builds tree structure based on word frequency in text, while our method is driven by the input and output distribution from images.

8 Conclusions and Future Work

In this work, we explored hardware-software co-optimization mechanism to approximate Softmax function in DNNs and performed extensive algorithmic and hardware synthesis experiments to demonstrate the trade-off between area/energy costs and application accuracy.

Future work includes benchmarking the Softmax approximation approach against other accelerators, extending the algorithm hardware co-optimization to other nonlinear functions such as Sigmoid and Tanh, and evaluating the hardware-aware approximation method in other application domains like language modeling. We believe that having a light-weight area-and-energy-efficient implementation of nonlinear operations is an innovative key to boost the inference rate of different DNNs – for image classification, language modeling and translation, etc. – particularly for resource constraints embedded devices.

9 Acknowledgement

This research is supported by the Agency for Science, Technology and Research (A*STAR) under its Hardware-Software Co-optimization for Deep Learning (Project No.A1892b0026).

References

1. Amin, H., Curtis, K.M., Hayes-Gill, B.R.: Piecewise linear approximation applied to nonlinear function of a neural network. IEE Proceedings-Circuits, Devices and Systems (1997)
2. Bengio, Y., Léonard, N., Courville, A.C.: Estimating or propagating gradients through stochastic neurons for conditional computation. CoRR (2013)
3. Bengio, Y., Senecal, J.S.: Adaptive importance sampling to accelerate training of a neural probabilistic language model. IEEE Trans. Neural Networks (2008)
4. Chen, W., Grangier, D., Auli, M.: Strategies for training large vocabulary neural language models. In: ACL (2016)
5. Chen, Y.H., Krishna, T., Emer, J.S., Sze, V.: Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits (2017)
6. Courbariaux, M., Bengio, Y.: Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. In: NIPS (2016)
7. Dai, J., He, K., Sun, J.: Instance-aware semantic segmentation via multi-task network cascades. In: CVPR (2016)
8. Dai, J., Li, Y., He, K., Sun, J.: R-fcn: Object detection via region-based fully convolutional networks. In: NIPS (2016)
9. Dally, W.: High-performance hardware for machine learning. In: Cadence ENN Summit (2016)
10. Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. International journal of computer vision (2010)
11. Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M.A., Dally, W.J.: Eie: efficient inference engine on compressed deep neural network. In: ISCA (2016)
12. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In: ICLR (2016)
13. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: ICCV (2017)
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
15. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: ICCV (2017)
16. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
17. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. arXiv preprint arXiv:1602.07360 (2016)
18. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al.: In-datacenter performance analysis of a tensor processing unit. In: ISCA (2017)
19. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPs (2012)
20. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: ECCV (2014)
21. Morin, F., Bengio, Y.: Hierarchical probabilistic neural network language model. In: Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (2005)

22. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: ICML (2010)
23. Namin, A.H., Leboeuf, K., Muscedere, R., Wu, H., Ahmadi, M.: Efficient hardware implementation of the hyperbolic tangent sigmoid function. In: ISCAS (2009)
24. Nilsson, P., Shaik, A.U.R., Gangarajaiah, R., Hertz, E.: Hardware implementation of the exponential function using taylor series. In: NORCHIP, 2014 (2014)
25. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: ECCV (2016)
26. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: NIPS (2015)
27. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
28. Willhalm, T., Dementiev, R., Fay, P.: Intel performance counter monitor. (2017), https://software.intel.com/en-us/articles/intel-performance-counter-monitor
29. Yang, T.J., Chen, Y.H., Sze, V.: Designing energy-efficient convolutional neural networks using energy-aware pruning. In: CVPR (2017)
30. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. CoRR (2016)
31. Zhu, C., Han, S., Mao, H., Dally, W.J.: Trained ternary quantization. In: ICLR (2017)