Efficient Precision-Adjustable Architecture for Softmax Function in Deep Learning

Danyang Zhu, Siyuan Lu, Meiqi Wang, *Graduate Student Member, IEEE*, Jun Lin, *Senior Member, IEEE*, and Zhongfeng Wang, *Fellow, IEEE*

***Abstract*—The softmax function has been widely used in deep neural networks (DNNs), and studies on efficient hardware accelerators for DNN have also attracted tremendous attention. However, it is very challenging to design efficient hardware architectures for softmax because of the expensive exponentiation and division calculations in it. In this brief, the softmax function is firstly simplified by exploring algorithmic strength reductions. Afterwards, a hardware-friendly and precision-adjustable calculation method for softmax is proposed, which can meet different precision requirements in various deep learning (DL) tasks. Based on the above innovations, an efficient architecture for softmax is presented. By tuning the parameter *P*, the system accuracy and complexity can be adjusted dynamically to achieve a good tradeoff between them. The proposed design is coded using hardware description language (HDL) and evaluated on two platforms, Xilinx Zynq-7000 ZC706 development board and TSMC 28-nm CMOS technology, respectively. Hardware implementation results show that our architecture significantly outperforms other works in speed and area, and that by adjusting *P*, the accuracy can be further increased with little hardware overhead.**

***Index Terms*—Softmax, hardware architecture, deep neural network, transformer, VLSI.**

I. Introduction

DEEP learning (DL) has achieved great success in artificial intelligence area, and efficient hardware architecture design for deep neural networks (DNNs) is also one of the hot topics in both academic and industrial societies. The softmax function is a very widely used function in various DNNs [1]–[3], so efficient hardware architectures of softmax function are desired for high speed DL systems.

Nevertheless, the expensive exponentiation and division calculations in softmax function can cause large hardware complexity, especially when the softmax function requires to be calculated many times such as in the Transformer-based networks [2].

Manuscript received February 28, 2020; revised May 25, 2020; accepted June 11, 2020. Date of publication June 16, 2020; date of current version November 24, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61774082 and Grant 61604068, and in part by the Fundamental Research Funds for the Central Universities under Grant 021014380065. This brief was recommended by Associate Editor E. Vianello. *(Danyang Zhu and Siyuan Lu contributed equally to this work.) (Corresponding authors: Jun Lin; Zhongfeng Wang.)*

The authors are with the School of Electronic Science and Engineering, Nanjing University, Nanjing 210023, China (e-mail: zhudanyang10@foxmail.com; sylu@smail.nju.edu.cn; mqwang@smail.nju.edu.cn; jlin@nju.edu.cn; zfwang@nju.edu.cn).

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

Digital Object Identifier 10.1109/TCSII.2020.3002564

Another fact is that, the accuracy requirements of softmax function in various DL tasks are different. For example, the softmax layer in a CTC-trained RNN inference system does not have to give very accurate results [4], but training a neural network always demands high accuracy of the outputs of the softmax loss function. In addition, during the design and quantization stage of building a hardware accelerator for a DNN, the designers always need to balance the accuracy and the complexity of the softmax module, when the amount of hardware resources is different or the accuracy requirement is different. Recently, several works of hardware designs for softmax have been published [5]–[9]. However, with the bit-width of input and output data determined, most of the fixed-point calculation methods used by these works can only run under a specific precision. None of these works gives a convenient way to adjust the precision. In other words, by merely using the existing methods, it will be hard to achieve a good tradeoff between the precision and the complexity, when a softmax function needs to be integrated in a hardware-based DNN system.

To resolve these issues, a novel architecture for softmax function is designed. Our main contributions include the following: 1) a series of mathematical transformations used to simplify the softmax function and 2) a template architecture that can be configured with different precision to meet requirements of different applications.

Software experimental results prove that, by adjusting the parameter *P*, this method can meet different precision requirements in different DL tasks. In addition, hardware implementation results show that, the proposed architecture has excellent performance compared to prior works. Meanwhile, more accurate calculation results can be achieved with little hardware overhead.

The rest of this brief is organized as follows. Section II gives a brief review of related works. Algorithmic strength reductions are presented in Section III. In Section IV, we introduce the fix-point precision-adjustable calculation method for softmax. Section V presents the proposed architecture. Experimental results are given in Section VI. Section VII concludes this brief.

II. Related Works

A. Log-Sum-Exp-Trick

The softmax function can be expressed as follows:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} (i = 1, 2, \ldots, N), \tag{1}$$

where $x_1, x_2, \ldots, x_N$ are the input values of the softmax layer and the output value $f(x_i)$ represents the probability that the sample belongs to the i-th category.

The log-sum-exp trick [5] is adopted to perform the hardware implementation, which is:

$$f(x_i) = \frac{e^{x_i - x_{max}}}{\sum_{j=1}^{N} e^{x_j - x_{max}}} = exp(x_i - x_{max} - \ln(\sum_{j=1}^{N} e^{x_j - x_{max}})). \quad (2)$$

After the mathematical transformation, the division operation is replaced by a subtraction operation and a logarithm operation. In addition, numerical underflow is avoided.

B. Other Related Works

Existing works on designing the hardware architectures for softmax can be divided roughly into three kinds:

The first kind uses stochastic computation (SC) to simplify the softmax function. Authors of [7] tried to use Integral Stochastic computation [10] for hardware implementation of softmax. The mean absolute error (MAE) of this design is believed to be the biggest, which is on the order of magnitude of 10 to the power of -2. Using SC will make it hard to increase the accuracy, unless a very long computational delay can be tolerated.

The magnitude of the MAE of the second kind is between 10^{-4} and 10^{-7}, including [8] and [9]. These two works use piecewise-linear approximation techniques, and a very high clock frequency has been achieved. The FPGA maximum frequencies of both works are around 400MHz.

The third kind utilizes high-precision LUTs and has the highest accuracy such as [6]. This kind of design also has the lowest speed and the highest hardware complexity. In [6], the MAE is on the order of magnitude of 10 to the power of -8, and the clock frequency is the lowest compared to the first two kinds of architectures, which is about 150MHz on FPGA platform.

In short, we divide these works by their precisions (please refer to Table III). As mentioned earlier, it is hard to achieve a good tradeoff between the precision and the complexity by using these existing fixed-point softmax calculation methods.

III. Mathematical Transformation of Exponential and Logarithmic Calculation

A. Mathematical Transformation of Exponential Calculation

Based on the characteristics of the hardware design, we use 2 as the base number to change the exponential function into:

$$e^{-x_i} = 2^{-x_i \cdot \log_2 e}, x_i \geq 0. \quad (3)$$

Then divide the fix-point number $x_i \cdot \log_2 e$ into the integer part u_i and the decimal part v_i. The exponential calculation can be simplified as follows (Please note that ">>" means a right shift operation):

$$e^{-x_i} = 2^{(-u_i)+(-v_i)} = 2^{-u_i} \cdot 2^{-v_i} = 2^{-v_i} >> u_i. \quad (4)$$

Through the above mathematical transformation, the exponential operation e^{-x_i} is transformed into a base 2 exponentiation 2^{-v_i} and a shift operation, which is more efficient for hardware design. Besides, the range of the input is changed from $-\infty < x_i < +\infty$ to $0 < v_i \leq 1$, which greatly reduces the computational complexity.

B. Mathematical Transformation of Logarithmic Function

First of all, we use F to present $\sum_{j=1}^{N} exp(x_j - x_{max})$ in Equation (2). F is no smaller than 1:

$$F = \sum_{j=1}^{N} e^{x_j - x_{max}} \geq e^{x_{max} - x_{max}} = 1. \quad (5)$$

Define $m \in [1, 2)$, and w as an integer. The real number pair (m, w) satisfies the following equation:

$$\forall F \geq 0, \exists!(m, w) : F = 2^w \cdot m. \quad (6)$$

Then we have,

$$\ln F = \ln 2 \cdot \log_2 F = \ln 2 \cdot (w + \log_2 m). \quad (7)$$

A leading one detector (LOD) is required to find out the highest bit 1's position of F. By giving decimal point and the highest bit 1's position, values of m and w are easy to calculate.

According to Equations (6) and (7), the range of the input of logarithmic operation is limited within $[1, 2)$. To compute $\log_2 m$, a linear fitting is used:

$$\log_2 m \approx m - 1, m \in [1, 2). \quad (8)$$

At last, the calculation of $\ln F$ can be simplified as:

$$\ln F = \ln 2 \cdot (m - 1 + w). \quad (9)$$

IV. Fix-Point Calculation Method for Precision-Adjustable Softmax

Through the mathematical transformation in Equation (4), the exponential operation e^{-x_i} is transformed into the calculation of $2^{-v_i} (0 < v_i \leq 1)$ and shift operations. Considering the range of v_i in $f(v_i) = 2^{-v_i}$ is limited in $(0, 1]$, piecewise linear fitting method can be adopted to fit the function $f(v_i) = 2^{-v_i}$. We use function $f(v_i) = k \cdot v_i + b$ to approximate the function $f(v_i) = 2^{-v_i}$, in which k and b are different for different segments of v_i. By changing the number of segments, the precision-adjustable design is achieved. Through this fitting method, a design with high speed and low complexity for softmax function can be achieved at a certain accuracy.

Parameter P is used to represent the level of design's accuracy. Ranges are divided in binary form for the purpose of hardware-friendly design. The highest $P-1$ ($P \geq 2$) bits of v_i (named as "Flag") are used to determine the range to which it belongs. When $P = 0, 1$, the range $(0, 1]$ is not divided and "Flag" is not required. When $P \geq 2$, the range $(0, 1]$ is evenly divided into 2^{P-1} sections and the bits of the "Flag" are $P-1$.

Detailed precision-adjustable exponential calculation steps are described in Algorithm 1. Two exponential calculations will be performed during the whole process of softmax calculation. Parameters k and b in the function $f(v_i) = k \cdot v_i + b$ can be changed in the second exponential calculation. We use matrices K_1 and B_1 to save all possible slopes k and intercepts b for the first exponential calculation, respectively. Similarly, matrices K_2 and B_2 are used for the second exponential calculation. The sizes of these matrices are the same and determined by the precision parameter P. Besides, The parameters k and b are derived by the annealing algorithm and

Algorithm 1 The Proposed Exponential Calculation Method

input value:$X(i)$,$Stage$,P
output value:$EXP(i)$
$X(i) \leftarrow X(i) * 1.0111$
$u(i) \leftarrow floor(X(i))$
$v(i) \leftarrow X(i) - floor(X(i))$
if $Stage = 2$ *or* $Stage = 4$ **then**
  $s \leftarrow Stage/2$
  switch (P)
    case:0
    $v(i) \leftarrow b_s - (v(i) >> 1)$
    case:1
    $v(i) \leftarrow K_s[0][0] * v(i) + B_s[0][0]$
    case:≥ 2
    $flag \leftarrow$ *highest* $(P-1)$ *bits of* v_i
    $v(i) \leftarrow K_s[P-1][flag] * v(i) + B_s[P-1][flag]$
    end case
  end switch
end if
$EXP(i) \leftarrow v(i) << u(i)$

Algorithm 2 The Proposed Softmax Calculation Method

input value:$X(1), X(2), \ldots, X(N), P$
output value:$Y(1), Y(2), \ldots, Y(N)$
$Stage \leftarrow 1$
$X_{max} \leftarrow$ *the maximum of* $X(1), X(2), \ldots, X(N)$
$Stage \leftarrow 2$
$F \leftarrow 0$
for $i = 1 \ldots, N$ **do**
  $E(i) \leftarrow PaEXP(X_{max} - X(i), Stage, P)$
  $F \leftarrow F + E(i)$
end for
$Stage \leftarrow 3$
$w = LOD(F)$
$m = (F >> w)$
$\ln(F) \leftarrow 0.1011 * (m - 1 + w)$
$Stage \leftarrow 4$
for $i = 1 \ldots, N$ **do**
  $Z(i) \leftarrow X_{max} - X(i) - \ln F$
  $E(i) \leftarrow PaEXP(Z(i), Stage, P)$
  $Y(i) \leftarrow E(i)$
end for

Monte Carlo method. It should be noted that the optimization aim is to reduce the error of entire softmax calculation instead of exponential unit. Therefore, this problem has changed from a one-variable function fitting problem to a multivariate function fitting problem. When $P \geq 1$, the number of different k stored in K_1 is 2^{P-1}. In particular, when $P = 0$, k is set as $-1/2$ and multiplication $k \cdot v_i$ can be replaced by a shift operation. Noticing that since the range of v_i in calculation 2^{v_i} is limited to (0, 1], P does not need to be set very large to meet high accuracy requirements.

The precision-adjustable softmax calculation method is shown in Algorithm 2. We use the function "PaEXP" to represent the proposed precision-adjustable exponential calculation method used in it.

V. Hardware Architecture

A. The Overall Architecture for the Softmax Function

According to Algorithm 2, the overall architecture of the softmax function is designed and its whole computation process is shown in Fig. 1. It consists of a sorting block, multiple

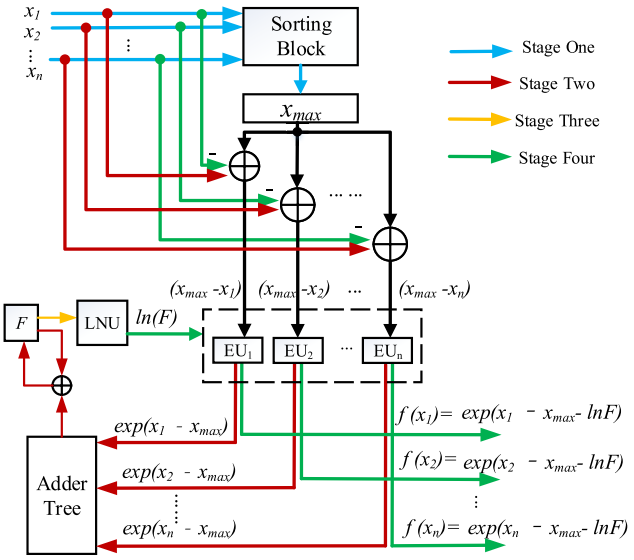

Fig. 1. The overall architecture for the softmax function.

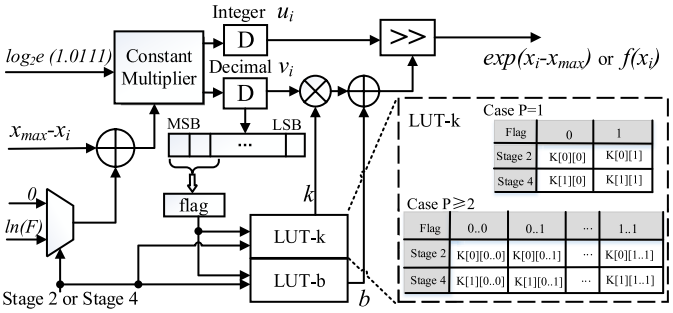

Fig. 2. The architecture of the EU.

exponential units (EUs), an adder tree, a natural logarithmic unit (LNU), and some other simple units. To avoid producing any negative numbers in this process, these EUs are used to calculate e^{-x}, and the inputs of EUs are $x_{max} - x_i$ in Stage 2 and Stage 4. When the number of all the input data N is relatively not very large, we can make the parallelism degree n (the number of EUs) equal to N, by which means the input data will be read for only one time during the four stages.

B. The Architecture of the EU

Based on Algorithm 1, the architecture of the EU is shown in Fig. 2. When $P = 0$, the coefficient k is set as $-1/2$ and the variable multiplier is not needed. When $P \geq 1$, the hardware consumption of the lookup tables (as shown in Fig. 2) and selectors will increase as P increases. However, compared to the lookup table used in pre-storing possible results in prior arts, the size of lookup table required in this method is quite small.

C. The Architecture of the LNU

According to the mathematical transformation in Section III-B, the architecture of the LNU is shown in Fig. 3. The output of LOD unit is w, and $m = F >> w$. Since the value of m is limited in $[1, 2)$, $(m - 1)$ represents the decimal part of m which is limited in $[0, 1)$. Meanwhile,

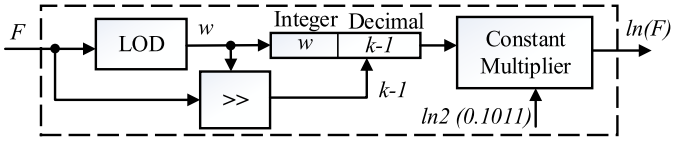

Fig. 3. The architecture of the LNU.

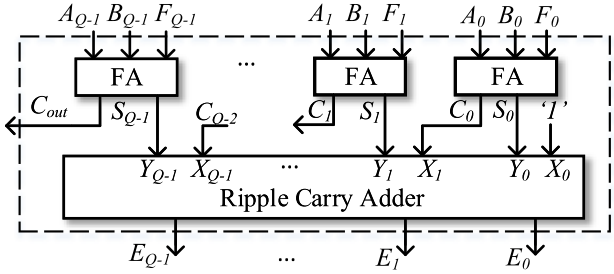

Fig. 4. The architecture of the constant multiplier.

w represents a non-negative integer. Therefore, no adder is needed for the calculation of $(m-1+w)$ in LNU.

D. Improved Architecture for the Constant Multiplier

The EU and the LNU each have a constant multiplier, which consists of the critical path before optimization. In this subsection, we propose an optimization method at both algorithm and architecture level to further boost up the speed and reduce hardware consumption.

Take the constant multiplier in the EU as an example. The binary approximation of $\log_2 e$ is 1.0111. We transform the expression "$1.0111 = 1 + 0.01 + 0.001 + 0.0001$" into "$1.0111 = 1 + 0.1 - 0.0001$". In this way, the three additions and three shift operations are reduced to one addition, one subtraction, and two shift operations.

Suppose the constant multiplier is used to calculate $E = A \times 1.0111$. Write $E = A \times 1.0111$ in the form of $E = A+B-D$ ($B = A >> 1, D = A >> 4$). Since A, B, and D are unsigned numbers in this implementation, this calculation can be further expressed as: $E = A+B+F+1$, $F = \sim D$ (which involves the critical path of the design when P = 0), where $\sim$ means the not operator. Carry save method [11] is adopted to simplify these additions in architecture level. The overall architecture of the constant multiplier is shown in Fig. 4.

A and E are Q-bit unsigned numbers. The critical path of traditional addition for $E = A + B + F + 1$ is the delay of 3Q full adders (FAs). By adopting the carry-save method, the critical path can be reduced to the delay of $Q+1$ FAs. Besides, the total number of FAs can be reduced from 3Q to 2Q.

The inputs of the constant multiplier in LNU are $(k-1+w)$ and the binary approximation of $\ln 2$ (0.1011). Similarly, it is implemented with the carry-save method as in the EU.

VI. Experimental Results and Comparisons

In this section, both software experiments and hardware experiments are presented. All the input and output fixed-point numbers are preserved in 16 bits.

A. The Accuracy Loss of Calculating Random Input Data

Four sets of random values with different ranges are chosen as the input data in the experiment. They are represented as rand0.1, rand1, rand5, and rand10, and each set has 4096 numbers within the corresponding range. For instance, numbers in set rand0.1 are random values lying in the range $[-0.1, +0.1]$, and those in rand1 lie in the range $[-1, 1]$. In each set, all of these values follow a uniform distribution.

TABLE I
MAE and MSE of Calculating Random Input Data

	P=0	P=1	P=2	P=3
Average MAE	3.55e-6	3.46e-6	9.55e-7	5.19e-7
Average MSE	1.06e-10	8.86e-11	6.38e-12	2.28e-12

TABLE II
Using the Proposed Method in the Language Modeling Task

Situation	Baseline	P=+∞[1]	P=0	P=1	P=2	P=3
bpc	1.0565	1.0566	1.0589	1.0580	1.0567	1.0563

[1] In the situation "P=+∞": The Transformer-XL has been quantized to fixed-point values but softmax is still implemented with floating-point numbers.

The average MAE and mean square error (MSE) of calculating these four sets of input data are listed in Table I. These results prove that the error can be reduced by increasing the value P.

B. Use the Proposed Method in Quantized Transformer-XL

We also apply our proposed method in an natural language processing (NLP) task. Transformer-XL [12] is one of the leading architectures in NLP area. It consists of a segment-level recurrence mechanism and a positional encoding scheme, which enables it to capture longer-term dependency than RNNs and the vanilla Transformer [13]. Also, as introduced in [12], it is up to 1800+ times faster than the vanilla Transformer during evaluations, which makes it more suitable for practical applications. However, the softmax function is one of the hardest parts to deal with in the hardware implementation of Transformer-XL.

We choose the language modeling task with the dataset enwik8 [14] for the experiment. The weights and the intermediate results of the network are quantized into 16-bit fixed-point values. Based on the quantized model, the inference results are presented in TABLE II. Please note that, the lower bit-per-character (bpc) is, the higher the accuracy is.

As shown in TABLE II, as P increases, the system accuracy increases, and the bpc of $P = 3$ is even lower than original model that has not been quantized. These results show that the proposed architecture for softmax can help efficiently implement the Transformer-XL network on hardware platforms, and that the proposed precision-adjustable scheme is able to obtain a good tradeoff between system complexity and accuracy.

C. Hardware Implementation Results

The parameter P is set from 0 to 3, and our design is evaluated on two platforms, Xilinx Zynq-7000 ZC706 development board and TSMC 28-nm CMOS technology, respectively.

As shown in Table IV, by increasing the size of P, more accurate calculation results can be reached with little hardware overhead, so a good tradeoff between the precision and the complexity can be reached conveniently. When P is greater than 0, an extra variable multiplier becomes the critical path

TABLE III
COMPARISONS WITH RELATED WORKS

Characteristic	[7]	[8]	P=0		[9]	P=2		P=3	[6]
Parallelism degree (Number of input / output)	1	1	1		1	1		8	1
FPGA Frequency(MHz)	-	396.04	500		400	357		294	150
FPGA LUT	10746	4779	395		300	401		1858	17870
FPGA Slice Register	-	738	498		558	504		2086	16400
FPGA DSP	0	0	0		5	1		8	-
FPGA BRAM	0	0	0		72Kbits	0		0	-
FPGA Throughput(G/s)	-	0.4	0.5		0.4	0.36		2.35	0.15
Average MAE	2.92e-2	1.5e-5	3.55e-6		-	9.55e-7		5.19e-7	e-8 magnitude
Average MSE	-	-	1.06e-10		1.12e-11	6.38e-12		2.28e-12	-

TABLE IV
KEY FEATURES OF THE PROPOSED HARDWARE ARCHITECTURE

Characteristic	P=0	P=1	P=2	P=3	[15][1]
Number of input / output	8	8	8	8	1
FPGA Frequency(MHz)[2]	500	294	294	294	-
FPGA LUT	1580	1536	1648	1858	-
FPGA Slice Register	1800	1818	1850	2086	-
FPGA DSP	0	8	8	8	-
FPGA Throughput(G/s)	4	2.35	2.35	2.35	-
ASIC Frequency(GHz)	2.78	1.64	1.64	1.64	1.16
ASIC Area(um^2)	10081	15926	18086	18392	118760
ASIC Throughput(G/s)	22.24	13.12	13.12	13.12	1.16

[1] The architecture in [15] is sythesised under 65nm CMOS, and its results are scaled here.

[2] This frequency is a representation of how fast the architecure can run. The frequency in practical applications will be limited by the speed of data transmission and the clock speed of the entire system.

in the circuits (when P = 0, the critical path is in the constant multiplier of the EU as mentioned earlier). However, the clock frequency is still high enough (over 290MHz on FPGA, and over 1.6GHz on 28nm ASIC) to meet the requirement of almost all the DL tasks.

Comparison results with existing works [6]–[9], [15] are described in TABLE III and TABLE IV. In these two tables, results are arranged from left to right in the order of the error (MAE or MSE). All the comparisons are made under similar precision. Analyses are given as follows.

Firstly, the architectures implemented in both [7] and [8] have lower precision than our "P = 0", which also outperforms them in both speed and area. Secondly, the results of [9] are compared with our "P = 2". The numbers of LUTs and slice registers used by [9] are similar to our "P = 2", but their utilization of DSP and BRAM is much higher than all the existing works. The large number of parameters is the primary disadvantage of their work. Thirdly, our "P = 3" is compared with [6], which has the highest precision. However, the huge hardware resource consumption and low speed of the design in [6] will make it difficult to be employed in practical DL applications. Finally, the ASIC synthesis results also prove our advantages in terms of both speed and area compared to [15]. In brief, our design is the most efficient for DL hardware accelerators among these works.

VII. CONCLUSION

In this brief, a hardware-friendly and precision-adjustable calculation method for softmax is proposed, which can meet different precision requirements in various DL tasks by adjusting the parameter P. Based on this calculation method, an efficient precision-adjustable hardware architecture for softmax function is presented. Hardware implementation results show that, our design not only significantly outperforms prior arts, but also allows a good tradeoff between the precision and the complexity to be achieved conveniently.

REFERENCES

[1] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 1764–1772.

[2] A. Vaswani *et al.*, "Attention is all you need," in *Proc. 31st Annu. Conf. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[3] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Proc. 31st Annu. Conf. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3856–3866.

[4] S. Lu, J. Lu, J. Lin, and Z. Wang, "A hardware-oriented and memory-efficient method for CTC decoding," *IEEE Access*, vol. 7, pp. 120681–120694, 2019.

[5] B. Yuan, "Efficient hardware architecture of softmax layer in deep neural network," in *Proc. System-On-Chip Conf.*, 2017, pp. 323–326.

[6] Q. Sun *et al.*, "A high speed softmax VLSI architecture based on basic-split," in *Proc. 14th IEEE Int. Conf. Solid-State Integr. Circuit Technol. (ICSICT)*, Qingdao, China, 2018, pp. 1–3.

[7] R. Hu, B. Tian, S. Yini, and S. Wei, "Efficient hardware architecture of softmax layer in deep neural network," in *Proc. IEEE 23rd Int. Conf. Dig. Signal Process. (DSP)*, Shanghai, China, 2018, pp. 1–5.

[8] Z. Li, H. Li, X. Jiang, B. Chen, Y. Zhang, and G. Du, "Efficient FPGA implementation of softmax function for DNN applications," in *Proc. 12th IEEE Int. Conf. Anti Counterfeiting Security Identif. (ASID)*, Xiamen, China, 2018, pp. 212–216.

[9] M. K. Chughtai, M. B. Babar, M. Y. Qadri, and U. Qayyum, "A high speed and resource efficient approximation of softmax loss function," in *Proc. 16th Int. Bhurban Conf. Appl. Sci. Technol. (IBCAST)*, Islamabad, Pakistan, 2019, pp. 526–530.

[10] A. Ardakani, F. Leduc-Primeau, and W. J. Gross, "Hardware implementation of FIR/IIR digital filters using integral stochastic computation," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Shanghai, China, 2016, pp. 6540–6544.

[11] T. G. Noll, "Carry-save architectures for high-speed digital signal processing," *J. VLSI Signal Process. Syst. Signal Image Video Technol.*, vol. 3, nos. 1–2, pp. 121–140, 1991.

[12] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context," 2019. [Online]. Available: http://arxiv.org/abs/1901.02860.

[13] R. Al-Rfou, D. Choe, N. Constant, M. Guo, and L. Jones, "Character-level language modeling with deeper self-attention," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, pp. 3159–3166, 2019.

[14] M. Mahoney. (2011). *Large Text Compression Benchmark*. [Online]. Available: http://www.mattmahoney.net/text/text.html

[15] G. Du, C. Tian, Z. Li, D. Zhang, Y. Yin, and Y. Ouyang, "Efficient softmax hardware architecture for deep neural networks," in *Proc. Great Lakes Symp. VLSI*, 2019, pp. 75–80.