Lightweight Approximation of Softmax Layer for On-Device Inference

Ihor Vasyltsov and Wooseok Chang

1 Introduction

Due to the rapid growth of wireless communication technology, the number of Internet of Things (IoT) devices has increased dramatically in recent years. According to a Cisco, 50 billions of IoT devices will be connected to the Internet by 2020 [3]. In addition, it was estimated that data volume generated yearly by those devices will be more than 40 times bigger than the current global traffic, which will amount to 850 Zettabytes [7]. Thus, existing cloud computing infrastructure would be unable to provide good service for the analysis of new data as it simply has not enough computing power and network capacity for a large number of computation tasks. In addition, many AI applications (e.g., autonomous driving) have strict requirement on the latency of computation.

Therefore, it makes more sense to locate computations closer to the data sources, so called edge computing or on-device computing. On-device computing has better capabilities in terms of privacy, latency, scalability, reliability, diversity, and costs compared with traditional cloud computing [8, 14, 18, 19].

Deep learning (DL) tasks are usually computationally intensive and require large memory footprints. At the same time, end-devices have limited computing power and small memories to support raw large-scale DL models. Therefore, original DL models are optimized, compressed, distilled, and quantized to reduce the resource cost. There are already many methods for optimizing DL models [1, 6, 12], but most of them are related to quantization of matrix multiplication operations, while quantization of activations (built as nonlinear functions (NLFs)) has not been

I. Vasyltsov (✉) · W. Chang
Samsung Advanced Institute of Technology, Samsung Electronics, Suwon-si, Gyeonggi-do, South Korea
e-mail: ihor.vasiltsov@samsung.com

H. R. Arabnia et al. (eds.), *Advances in Artificial Intelligence and Applied Cognitive Computing*, Transactions on Computational Science and Computational Intelligence, https://doi.org/10.1007/978-3-030-70296-0_41

studied enough. Softmax layer is one of the most popular and important NLFs, but the complexity of implementation in the platform with limited hardware resources can be a bottleneck of application performance. Thus, we will focus on the usage of softmax layer in computer vision tasks as the main application.

In this paper we propose a lightweight method for efficient computation of the softmax layer at the devices with limited computational power. The method is based on the approximation of softmax by taking reciprocal of natural exponential function, which is implemented as 1-dimensional look-up-table (1-D LUT). In Sect. 2, we consider some preliminaries for understanding softmax, drawbacks of existing approximation methods, and propose our method. Section 3 shows the experimental validation of the proposed method with human segmentation tasks. Section 4 describes a plan for further extension of our research, and Sect. 5 concludes the paper.

2 Softmax Approximation

2.1 Preliminaries

In mathematics, the softmax is a function that takes a vector x of n real numbers as an input, and normalizes it into a probability distribution $P(x)$ consisting of n probabilities proportional to the exponential of each input number. Thus, after applying softmax, each component will be in the interval $\sigma(x_i) \in (0, 1)$, and the components will add up to 1, so that they can be interpreted as probabilities. Softmax is often used in neural networks, to map the non-normalized output of a network to a probability distribution over predicted output classes. There are different representations of the softmax function depending on the application [4], but the most well-known and widely accepted version is as follows [11]:

$$\sigma(x_i) = \frac{e^{x_i}}{\Sigma e^{x_i}} \tag{1}$$

In the real hardware the range of number representation is limited, thus e^x computations can often lead to overflow or underflow. Therefore for practical implementation to provide stable computation, a normalization of input by $x_i^* = x_i - max(x)$ is used [11][1] as shown below:

$$\sigma(x_i) = \frac{e^{x_i - max(x)}}{\Sigma e^{x_i - max(x)}} \tag{2}$$

[1]All major DL frameworks (TensorFlow v1.7, PyTorch (with Caffe2) v0.4.0, MXNET v1.1.0, Microsoft Cognitive Toolkit v2.5.1, and Chainer v5.0.0a1) are using this safe version for softmax computation [11].

2.2 Previous Arts

Softmax layer is one of the most important and widely used NLF in modern AI models, due to its smooth properties. However, many modern neural processing unit (NPU) architectures are focused on the acceleration of matrix multiplications only, as they are a majority of computational functions of a DL model. As a result, for computation of complex NLF (i.e. softmax layer), the data must be sent out of NPU to an external host (CPU, GPU, or DSP), which complicates the software development, and can also negatively impact on the overall performance and power consumption. In addition, since general purpose interface is used for data transmission, the internal data can be exposed to malicious user, which may cause an issue of data privacy.[2]

To avoid involvement of the host for softmax computation, some NPU proposed dedicated HW accelerators. In many of those implementations, each of the numerator and the denominator in Eq. (1) are computed first, and then a division operation is performed, e.g., [10, 15, 17]. In such case, the HW accelerator should contain a divider circuit (fixed-point, or even floating point), which requires an additional HW cost. To avoid a big area cost for traditional dividers, the authors in [5] propose to replace the denominator with closest 2^b value.[3] Then division can be implemented just as a simple bit-shift operation. Although the method described above is decreasing the hardware complexity of softmax computation it still relies on the division operation, which is not always feasible for end-devices with limited computational power.

2.3 Proposed Method

In general case, we can consider alternative softmax function $\sigma^*(x_i)$ as

$$\sigma^*(x_i) = \frac{score(x_i)}{norm(x)} \tag{3}$$

where $score(x_i)$ and $norm(x)$ are some scoring and normalization factors (for original function $score(x_i) = e^{x_i}$, and $norm(x) = \Sigma e^{x_i}$).

As described in [9], we can list some desirable properties of the alternative softmax function as below:

- **Nonlinearity:** for better selectivity of the scored values.
- **Numerical stability:** to avoid overflow, or underflow during computation.
- **Positive:** output values all should be positive, to be used for scoring.

[2]For example, for CCTV or industrial data sensing application.

[3]Where b is a certain integer constant.

Table 1 Softmax approximation methods and their properties

#	Method of approximation	Nonlinearity	Numerical stability	Positive	Bounded	Computational complexity
1	x_i	Bad	Bad	Bad	Bad	Best
2	e^{x_i}	Best	Bad	Good	Bad	Good
3	$\frac{x_i}{max(x)}$	Bad	Bad	Bad	Good	Bad
4	$\frac{x_i^2}{max^2(x)}$	Good	Bad	Good	Good	Bad
5	$\frac{x_i^2}{\Sigma x_i^2}$	Good	Bad	Good	Best	Worst
6	$e^{x_i - max(x)}$	Best	Best	Best	Best	Good
7	$\frac{1}{e^{max(x)-x_i}}$	Best	Best	Best	Best	Good

– **Bounded:** output values should be bounded by some constant, ideally, $\sigma^*(x_i) \in (0, 1)$.
– **Computational complexity:** should be feasible for the implementation into platform with limited HW resources.

Since we consider softmax approximation for inference task in computer vision applications where softmax layer is mostly used for *scoring* the outputs for classification, the requirements to normalization factor $norm(x)$ can be softer compared with the original formula Eq. (1), where inputs are mapped into the corresponding probabilities. Thus, we can use more various factors for normalization in $\sigma^*(x_i)$.

We have experimented with different approximations for $score(x_i)$ and $norm(x)$ factors, and summarized some of the methods and their properties in Table 1.[4]

First, we have started with the simple approximations, ignoring normalization factor at all, i.e. applying $norm(x) = 1$. We have obtained *identity* (i.e., $\sigma^*(x_i) = x_i$, method 1 in Table 1), and natural *exponentiation* function ($\sigma^*(x_i) = e^{x_i}$, method 2 in Table 1) for approximation. However, despite their low computational complexity, the numerical stability was not good as the output of function was not bounded. To counter this issue, we have applied some normalization factors (see methods 3 to 5 in Table 1), but the numerical stability was still poor. At the same time we have noticed that method 2 (exponentiation) is showing good selectivity due to its nonlinear property (refer to the corresponding image in Table A.1 in Appendix), and can be a good candidate if normalized appropriately. For this purpose we have performed several transformations as shown in Eq. (4) below:

$$\frac{e^{x_i}}{max(e^x)} = \frac{e^{x_i}}{e^{max(x)}} = e^{x_i - max(x)} = e^{-(max(x)-x_i)} = \frac{1}{e^{max(x)-x_i}} \tag{4}$$

[4]For more details, please refer to Table A.1 in Appendix, where statistical results and examples of images from initial tests are shown.

First, we kept e^{x_i} as a scoring factor $score(x_i)$ and then we have used $max(e^x)$ as a normalization factor to bound the output by 1 and thus we have obtained method 6. However, in such case, the input values to the exponential function e^x will be all negative due to the $x_i - max(x)$ term, and if e^x is implemented by LUT (which is a common approach for HW with limited computation resources), then additional affine transformation is required to compensate for negative input values. To avoid this drawback, we propose to use max-normalization in the inverse way as $x_i^* = max(x) - x_i$, then input values to e^x will be all positive, which allows them to be used directly as indices to LUT values. Second, to compensate for the inverse way of max-normalization, we have used the reciprocal version of $e^x \to 1/e^x$ as shown in Eq. (4).

In such case neither divider nor multiplier is needed, and only 1-D LUT is required to compute the approximated value of softmax. As a result, the computational complexity is reduced significantly, and it becomes more feasible for the implementation in HW with limited computational power.

Thus, we propose to substitute the original method for softmax computation with the inverse way of max-normalization and the reciprocal of exponential function:

$$\sigma(x_i) = \frac{e^{x_i}}{\Sigma e^{x_i}} = \frac{e^{x_i - max(x)}}{\Sigma e^{x_i - max(x)}} \to \sigma^*(x_i) = \frac{1}{e^{max(x) - x_i}} \qquad (5)$$

The properties of the proposed method $\frac{1}{e^{max(x)-x_i}}$ are as below:

- **Nonlinearity** is satisfied with the reciprocal of exponential function $1/e^x$: $\frac{1}{e^{\alpha x}} \neq \alpha \frac{1}{e^x}$
- **Numerical stability** is satisfied by $max(x) - x_i$ term: $(max(x) - x_i) \in [0, max(x) - min(x)] \to \frac{1}{e^{max(x)-x_i}} \in (0, 1]$
- **Positive** output values are due to the exponential function: $\frac{1}{e^x} > 0\ \forall x \in (-\infty, +\infty)$
- **Bounded** $\sigma^*(x_i) \in (0, 1]$ due to the inverse normalization term $max(x) - x_i$ used together with the reciprocal of exponential function $1/e^x$: $(max(x) - x_i) \geq 0\ \forall x \to \frac{1}{e^{max(x)-x_i}} \in (0, 1]$
- **Computational complexity** is low, as $1/e^x$ can be implemented with LUT-based method, where the size of LUT is small.

Indices in LUT can be directly calculated by rounding operation as $i = \lfloor max(x) - x \rceil$. When input data are quantized by w bits then *efficient quantization boundary* x_q can be defined as[5]

[5]Efficient quantization boundary x_q defines the biggest input value, which can be mapped into w quantization bits.

$$\begin{aligned} e^{-x_q} &= \frac{1}{2^w - 1} \\ ln(e^{-x_q}) &= ln(\frac{1}{2^w - 1}) \\ -x_q &= ln(1) - ln(2^w - 1) \\ x_q &= ln(2^w - 1) \\ x_q &= \lceil ln(2^w - 1) \rceil \end{aligned} \tag{6}$$

Content of LUT is computed as shown below:

$$LUT_{1/e}[i] = \left\lfloor \frac{1}{e^i} \cdot (2^w - 1) \right\rceil, \forall i = 0, 1, \ldots, x_q + 1 \tag{7}$$

Note, that $LUT[i] = 0, \forall i > x_q$ due to quantization, as no value can be encoded with w bits after efficient quantization boundary x_q.

If selectivity (precision of computation) is not enough, then LUT can be scaled linearly by α as

$$LUT_{1/e}[i] = \left\lfloor \frac{1}{e^{i/\alpha}} \cdot (2^w - 1) \right\rceil, \forall i = 0, 1, 2, \ldots, \alpha \cdot (x_q + 1) \tag{8}$$

3 Experimental Validation

To validate the proposed method we have used a pre-trained Unet model for human segmentation and internally prepared dataset with 1000 images. In this model softmax computation is used to predict the class of every pixel, thus requiring 307,200 computations for typical 640 × 480 image. This model takes the image as an input, and produces the predicted grey-scale image for segmentation, where each pixel P_i is in uint8 precision (with values from 0 to 255). To get the binary mask for segmentation, every pixel in those images was binarized into two classes (class 0 for "background," and class 1 for "human") by using threshold $thr = 127$, as follows:

$$B_i^m = \begin{cases} 0, \ \forall P_i < thr \\ 1, \ \forall P_i \geq thr \end{cases} \tag{9}$$

In the model we have substituted the conventional softmax layer with the computation method as described above in Sect. 2. For practical implementation, we have selected three different precisions (uint8, uint4, uint2) and prepared the LUTs accordingly to Eq. (7). For evaluation accuracy of segmentation we used the well-known bit-wise intersection-over-union metric [13, 20] as shown below:

Table 2 Accuracy of different approximation methods. Full test over 1000 images

#	Method of approximation	Precision	Size of LUT (Bytes)	IoU (class 0)	IoU (class 1)	mIoU
0	Reference	FP32		0.9847	0.9799	0.9823
1	$\frac{x_i^2}{max^2(x)}$	FP32		0.9817	0.9746	0.9781
2	$\frac{x_i^2}{\Sigma x_i^2}$	FP32		0.9828	0.9765	0.9797
3	$e^{x_i - max(x)}$	FP32		0.9845	0.9800	0.9823
4	$\frac{1}{e^{max(x)-x_i}}$	uint8	8	0.9845	0.9799	0.9822
5	$\frac{1}{e^{max(x)-x_i}}$	unit4	5	0.9845	0.9799	0.9822
6	$\frac{1}{e^{max(x)-x_i}}$	uint2	3	0.9842	0.9792	0.9817

$$IoU = \frac{area(B^m_{gt,i} \bigcap B^m_{p,i})}{area(B^m_{gt,i} \bigcup B^m_{p,i})} \tag{10}$$

where $B^m_{gt,i}$ is a pixel-group of ground-truth image, and $B^m_{p,i}$ is that of predicted segmentation image. The mIoU value was computed as a mean value among two classes.

Table 2 shows the results of our experiments for different methods of approximation and selected precision (for LUT-based method). As it comes from the table, the accuracy of human segmentation task based on the proposed approximation of softmax layer is as high as the FP32 reference. There is no, or negligibly small accuracy drop ($< 0.1\%$ for 2-bit quantization) even for very small size of LUT (3 to 8 bytes).

4 Future Work

Despite its extremely low computational complexity, the current version of the softmax approximation can be applied only to the applications where softmax layer is used for scoring (typically last layer in CNN models), calculated within one input tensor only. Thus, it cannot be directly applied to more complicated and softmax-intensive applications such as Natural Language Processing (NLP) tasks where cross-tensor probabilities must be computed more often (e.g., multi-head attention block in Transformer [16], and BERT [2] models). Therefore, we will work forward in order to extend the proposed method to other classes of AI applications.

5 Conclusion

In this paper we have proposed an efficient method for softmax approximation, which can be implemented at the platform with limited hardware resources (mobile, IoT, edge devices) for AI inference tasks. We have applied max-normalization to the input data in the inverse way, which together with the application of LUT-based method for computation of the reciprocal exponential function $1/e^x$ has significantly reduced the complexity of softmax layer computation. It also has the additional benefits as follows:

- does not require any additional multiplier, divider, or adder.
- scalable in terms of accuracy and precision (appropriate LUTs can be pre-computed off-line).
- fixed latency of computation, which depends only on the size of the tensor.

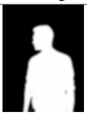
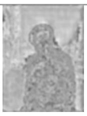
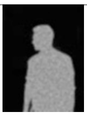
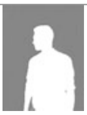
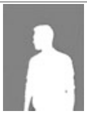
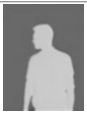
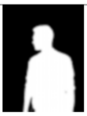
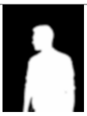
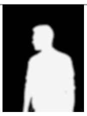
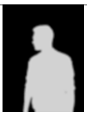
Thus, the proposed approach provides a good alternative for HW accelerator design, simplifying the overall process of computing the softmax layer, while maintaining identical accuracy to the conventional FP32 based computation.

Appendix

In this section there are presented more details about the research on softmax layer approximation. There are shown more methods for approximation, as well their results over initial test. The experiments for initial tests were conducted the same way as in Sect. 3, but over sub-set of 100 images. Also, examples of images generated by human segmentation model for different methods of softmax computation are shown (Table A.1).

As it can be seen from the table, imag e generated by human segmentation model with method 2 ($\sigma^*(x_i) = e^{x_i}$) shows good selectivity of the method. Thus, we used it as a base for creating the finally proposed method $\sigma^*(x_i) = \frac{1}{e^{max(x)-x_i}}$.

Table A.1 Accuracy of different approximation methods. Initial test over 100 images

#	Method of approximation	Precision	IoU (class 0)	IoU (class 1)	mIoU	Image (example)
0	Reference	FP32	0.9844	0.9833	0.9838	
1	x_i	FP32	0.1731	0.2274	0.2002	
2	e^{x_i}	FP32	0.7320	0.4989	0.6154	
3	$\frac{x_i}{max(x)}$	FP32	0.9667	0.9557	0.9612	
4	$\frac{x_i^2}{max^2(x)}$	FP32	0.9811	0.9778	0.9795	
5	$\frac{x_i^2}{\Sigma x_i^2}$	FP32	0.9825	0.9800	0.9812	
6	$e^{x_i - max(x)}$	FP32	0.9838	0.9828	0.9833	
7	$\frac{1}{e^{max(x)-x_i}}$	uint8	0.9838	0.9827	0.9833	
8	$\frac{1}{e^{max(x)-x_i}}$	uint4	0.9838	0.9827	0.9833	
9	$\frac{1}{e^{max(x)-x_i}}$	uint2	0.9837	0.9822	0.9829	

References

1. Y. Cheng, D. Wang, P. Zhou, T. Zhang, A survey of model compression and acceleration for deep neural networks (2017). *CoRR*, abs/1710.09282
2. J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding (2018). *CoRR*, abs/1810.04805
3. Fog computing and the internet of things: extend the cloud to where the things are (2015). https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf, Accessed 10 March 2020
4. B. Gao, L. Pavel, On the properties of the softmax function with application in game theory and reinforcement learning (2017, preprint). arXiv:1704.00805
5. X. Geng, J. Lin, B. Zhao, A. Kong, M.M. Sabry Aly, V. Chandrasekhar, Hardware-aware softmax approximation for deep neural networks, in *Computer Vision – ACCV 2018*, ed. by C.V. Jawahar, H. Li, G. Mori, K. Schindler (Springer, Cham, 2019), pp. 107–122
6. Y. Guo, A survey on methods and theories of quantized neural networks (2018). *CoRR*, abs/1808.04752
7. Y. Han, X. Wang, V.C.M. Leung, D. Niyato, X. Yan, X. Chen, Convergence of edge computing and deep learning: a comprehensive survey (2019). *CoRR*, abs/1907.08349
8. A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, L.V. Gool, AI benchmark: running deep neural networks on android smartphones (2018). *CoRR*, abs/1810.01109
9. S. Kanai, Y. Fujiwara, Y. Yamanaka, S. Adachi, Sigsoftmax: reanalysis of the softmax bottleneck (2018, preprint). arXiv:1805.10829
10. Z. Li, H. Li, X. Jiang, B. Chen, Y. Zhang, G. Du, Efficient FPGA implementation of softmax function for DNN applications, in *2018 12th IEEE International Conference on Anti-Counterfeiting, Security, and Identification (ASID)* (2018), pp. 212–216
11. M. Milakov, N. Gimelshein, Online normalizer calculation for softmax (2018). *CoRR*, abs/1805.02867
12. Model compression papers (2018). https://github.com/chester256/Model-Compression-Papers, Accessed 10 March 2020
13. S.H. Rezatofighi, N. Tsoi, J.Y. Gwak, A. Sadeghian, I.D. Reid, S. Savarese, Generalized intersection over union: a metric and a loss for bounding box regression (2019). *CoRR*, abs/1902.09630
14. M.G. Sarwar Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, F. Hussain, *Machine Learning at the Network Edge: A Survey* (Open Access Archive of Cornell University, New York, USA, 2019) https://arxiv.org/abs/1908.00080. Accessed June 15, 2021
15. Q. Sun, Z. Di, Z. Lv, F. Song, Q. Xiang, Q. Feng, Y. Fan, X. Yu, W. Wang, A high speed softmax VLSI architecture based on basic-split, in *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)* (2018), pp. 1–3
16. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need (2017). *CoRR*, abs/1706.03762
17. K. Wang, Y. Huang, Y. Ho, W. Fang, A customized convolutional neural network design using improved softmax layer for real-time human emotion recognition, in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)* (2019), pp. 102–106
18. S. Wang, A. Pathania, T. Mitra, Neural network inference on mobile SoCs. IEEE Des. Test **37**, 50–57 (2020)
19. X. Zhang, Y. Wang, S. Lu, L. Liu, L. Xu, W. Shi, OpenEI: an open framework for edge intelligence (2019). *CoRR*, abs/1906.01864
20. D. Zhou, J. Fang, X. Song, C. Guan, J. Yin, Y. Dai, R. Yang, *IoU loss for 2d/3d object detection* (Open Access Archive of Cornell University, New York, USA, 2019) https://arxiv.org/abs/1908.03851. Accessed June 15, 2021