JKIICE

Short Paper

한국정보통신학회논문지 Vol. 25 No. 2: 330~333, Feb. 2021

CNN의 SoftMax 연산을 위한 연속 근사 방식의 로그 연산 회로

Successive Approximated Log Operation Circuit for SoftMax in CNN

Hyeong-Ju Kang*

*Associate Professor, School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan, 31253 Korea

ABSTRACT

In a CNN for image classification, a SoftMax layer is usually placed at the end. The exponentinal and logarithmic operations in the SoftMax layer are not adequate to be implemented in an accelerator circuit. The operations are usually implemented with look-up tables, and the exponential operation can be implemented in an iterative method. This paper proposes a successive approximation method to calculate a logarithm to remove a very large look-up table. By substituing the large table with two very small tables, the circuit can be reduced much. The experimental results show that the 85% area reduction can be reached with a small error degradation.

Keywords : Accelerator, CNN, Logarithm, SoftMax

Ⅰ. 서 론

최근 다양한 영상 인식 분야에서 CNN이 높은 성능을 보이며 널리 사용되고 있다[1]-[6]. 그러나 CNN이 많은 양의 연산을 요구함에 따라 임베디드 환경에서 사용되기에는 어려움이 있다. 이러한 단점을 극복하기 위해 CNN을 위한 전용 가속 회로를 설계하는 연구가 많이 이루어지고 있다.

CNN을 구성하는 여러 종류의 연산들은 대부분 가속 회로에서 구현하기에 적합하다. 그러나 CNN에서 인식된 객체의 종류에 대해 확률을 계산하기 위해 사용되는 SoftMax 연산의 경우, 가속 회로에서 구현하기 어려운 지수 연산과 나눗셈 연산으로 구성되어 있다.

SoftMax 연산을 구성하는 지수 연산과 나눗셈 연산을 구현하기 위해 기존에 많이 이용한 방법은 나눗셈 연산을 로그 연산과 뺄셈으로 변환하여 구현하는 방법이다. 이 경우 SoftMax 연산은 지수 연산과 로그 연산으로 구성되며, 각각을 LUT (Look-Up Table)로 구현한다[7]. 그러나 이 경우 LUT의 개수가 많아지므로 이를 줄이기 위해 지수 연산을 순차적으로 계산하는 방법도 제안되었다[8].

이 논문에서는 지수 연산을 순차적으로 계산하는 기존 방법을 분석하여, 로그 연산을 위한 LUT가 더 큰 면적을 차지함을 보이고, 로그 연산 LUT를 없애기 위해 로그 연산을 순차적으로 계산하는 방법을 제안한다. 연속 근사 방식을 응용하였으며 이 방법을 통해 하나의 큰 로그 연산 LUT를 두 개의 작은 LUT와 한 개의 곱셈기로 대체할 수 있다.

이 논문은 다음과 같이 구성되어 있다. 2장에서는 기존의 SoftMax 층 구현 방법을 소개하고 3장에서는 제안하는 방식을 설명한다. 4장에서 실험 결과를 보이고 5장에서 결론을 낸다.

Ⅱ. SoftMax 층의 구현

CNN에서 사용되는 SoftMax는 N개의 입력값 x_n (0 ≤n<N)에 대해 N개의 출력값 z_n을 아래 식(1)과 같이 계산한다.

$$z_n = e^{x_n} / \sum_{j=0}^{N-1} e^{x_j} \tag{1}$$

결과값인 z_n은 0에서 1 사이의 값을 가진다. 모든 출력값 z_n을 계산할 수도 있으나, 실제로 의미가 있는 것은 입력값 x_n중 가장 큰 값인 x_{MAX}에 해당하는 출력값

Received 13 December 2020, **Revised** 11 January 2021, **Accepted** 26 January 2021

*** Corresponding Author** Hyeong-Ju Kang(E-mail: hjkang@koreatech.ac.kr, Tel:+82-41-560-1420)
Associate Professor, School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan, 31253 Korea

Open Access http://doi.org/10.6109/jkiice.2020.25.2.330 print ISSN: 2234-4772 online ISSN: 2288-4165

z_{MAX}이므로 이 논문에서는 z_{MAX}만을 계산하는 것을 가정한다.

이러한 SoftMax에 필요한 지수 연산과 나눗셈은 가속 회로로 구현하기에 적합하지 않다. 이를 해결하기 위해 일반적으로 아래 식(2)와 같이 지수 연산과 로그 연산으로 이루어진 식으로 변환한다.

$$\log z_n = \log \frac{e^{x_n}}{\sum_{j=0}^{N-1} e^{x_j}} = x_n - \log \sum_{j=0}^{N-1} e^{x_j} \quad (2)$$

위의 식 (2)대로 계산한 결과에 다시 지수 연산을 취하면 z_n을 얻을 수 있다.

식(2)에 따라 계산을 하기 위해서는 지수 연산과 로그 연산이 필요하며, 이 연산들은 일반적으로 LUT로 구현한다. 그림 1이 그 블록도이다. 이 그림에서 MAX는 최대값을 골라내는 블록이며, EXP는 지수 연산 회로, LOG는 로그 연산 회로이다. 입력 값에 지수 연산을 한 뒤 SUM에서 합을 구하고 로그 연산을 한다. 그 다음에 최대 입력값과 로그 출력값을 뺀 뒤에 다시 지수 연산을 취한다.

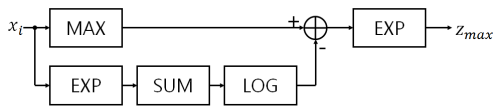

Fig. 1 SoftMax Circuit

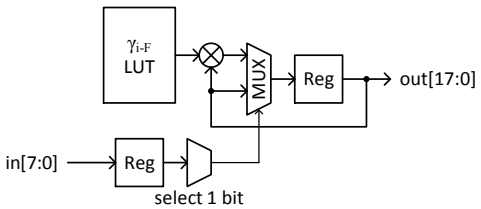

Fig. 2 Iterative EXP Circuit

그림 1 회로에서 EXP와 LOG는 일반적으로 LUT를 이용하여 구현한다. 그러므로 그림 1 회로는 세 개의 LUT를 사용한다. 이러한 LUT의 개수를 줄이기 위해 EXP를 순차적으로 계산하는 방법도 제안되었다. 이 방법에서는 식(3)과 (4)를 이용한다.

$$x = \sum_{i=0}^{w-1} a_i 2^{i-F} \quad (3)$$

$$e^x = e^{\sum_{i=0}^{w-1} a_i 2^{i-F}} = \prod_{i=0}^{w-1} e^{a_i 2^{i-F}} = \prod_{i=0}^{w-1} \gamma_{i-F}^{a_i} \quad (4)$$

입력값의 데이터 비트수가 w이고 그 중 소수 부분의 비트수가 F일 때, 입력값을 비트 단위로 분해하면 2^{i-F}들의 합이 되므로, 입력값의 지수승은 $\gamma_{i-F} = e^{2^{i-F}}$의 곱으로 표현할 수 있다. 그러므로 입력값의 i번 비트를 순차적으로 검사하여 그 비트가 1일 때 γ_{i-F}을 곱해 나간다. 모든 비트에 대해 이 과정을 반복하면 입력값의 지수승이 출력된다.

그림 2가 순차적으로 지수 연산을 수행하는 회로이다. 기존의 큰 LUT를 대신하여 γ_{i-F}를 저장하는 작은 LUT와 곱셈기로 구성된다.

Ⅲ. 연속 근사 방식의 로그 연산 회로

일반적인 CNN 가속회로에서 데이터의 비트수는 8비트로 충분하다고 알려져 있다. 그러므로 보통 SoftMax 회로의 입력도 8비트를 가정한다. 하위 4비트가 fractional 부분이고 Signed 값을 가정하면 값의 범위는 [-8,8)이다. 이 입력값의 지수승을 구하면 $[e^{-8}, e^{8})$로 그 범위가 매우 넓어진다. 매우 큰 값과 작은 값을 모두 표현하기 위해서는 더 많은 비트를 사용해야 하므로 지수 연산의 결과는 비트 폭이 커지게 된다. 지수 연산의 결과가 16비트라고 가정했을 때, LUT로 지수 연산을 구현하면 LUT의 한 entry는 16비트이고 256 entry가 필요해서, LUT의 총 비트 수는 256x16 = 4096비트가 된다.

이에 비해 로그 연산의 LUT는, 지수 연산의 결과의 합이 입력이므로 입력의 비트 수가 16비트이다. 그러므로 LUT의 entry수도 65,536개로 매우 많다. 로그 연산의 결과는 지수 연산을 하기 전의 값과 범위가 비슷하므로 8비트로 표현 가능하다. 그러므로 LUT의 크기는 65,536x8 = 524Kbit이다. 즉, 지수 연산을 위한 LUT에 비해 로그 연산을 위한 LUT가 큰 것이다.

문헌 [8]에 제안된 방법은 지수 연산의 LUT를 대체하여 작게 만들었으나 그 보다 큰 면적을 차지하는 로그 연산의 LUT는 대체하지 못한다. 이 논문에서는 큰 면적을 차지하는 로그 연산의 LUT를 순차적인 연산으로 대체하려 한다. 기반이 되는 수식은 아래와 같다. 로그 연산의 입력을 y라고 했을 때 그 로그 연산의 결과는 식(5)와 (6)과 같이 표현할 수 있다.

$$u = \log y = \sum_{i=0}^{w-1} b_i 2^{i-F} \quad (5)$$

$$y = e^u = e^{\sum_{i=0}^{w-1} b_i 2^{i-F}} = \prod_{i=0}^{w-1} e^{b_i 2^{i-F}} = \prod_{i=0}^{w-1} \gamma_{i-F}^{b_i} \quad (6)$$

여기서 최상위 비트인 b_{w-1}를 결정하는 것은 간단하다. b_{w-1}가 －1이면 z가 음수이므로 y가 1보다 작을 때이다. 즉 y가 1보다 작으면 b_{w-1}가 －1이고, 1보다 크거나 같으면 b_{w-1}가 0이다. 다음으로 이후 진행을 위해 아래와 같이 y_{w-2}을 구한다.

$$y_{w-2} = y/\gamma_{w-1-F}^{b_{w-1}} = \prod_{i=0}^{w-2} \gamma_{i-F}^{b_i} = \gamma_{w-2-F}^{b_{w-2}} \prod_{i=0}^{w-3} \gamma_{i-F}^{b_i} \quad (7)$$

이 식에서 주목할 것은 $b_i 2^{i-F}$는 항상 양수이므로 $\gamma_{i-F}^{b_i}$는 항상 1 이상 이라는 것이다. y_{w-2}를 식(7)의 오른쪽과 같이 분해할 수 있는데, 여기서 $\gamma_{i-F}^{b_i}$가 모두 1 이상이므로 이들의 곱인 $\prod$의 결과는 1 이상이고, 만일 b_{w-2}가 1이면 y_{w-2}는 γ_{w-2-F}보다 크거나 같게 된다.

만일 b_{w-2}가 0이면 식(8)에 의해 y는 γ_{w-2-F}보다 작게 된다.

$$\sum_{i=0}^{w-3} b_i 2^{i-F} < 2^{w-2-F} \quad (8)$$

즉 y를 γ_{w-2-F}과 비교하여 y가 더 크거나 같으면 b_{w-2}가 1이고 작으면 b_{w-2}가 0인 것이다. 이렇게 b_{w-2}를 결정하면 y를 $\gamma_{w-2-F}^{b_{w-2}}$로 나누어 $y_{w-3} = y/\gamma_{w-2-F}^{b_{w-2}}$를 구한다.

$$y_{w-3} = \prod_{i=0}^{w-3} \gamma_{i-F}^{b_i} = \gamma_{w-3-F}^{b_{w-3}} \prod_{i=0}^{w-4} \gamma_{i-F}^{b_i} \quad (9)$$

다음으로 b_{w-2}도 비슷하게 결정할 수 있다. y_{w-3}을 식(9)의 오른쪽과 같이 표현할 수 있으므로 γ_{w-3-F}과 비교한다. 위에서 기술한 것과 동일한 이유로 y_{w-3}가 더 크거나 같으면 b_{w-2}가 1이고 작으면 b_{w-2}가 0이다. 이렇게 b_{w-2}를 결정한 뒤 $\gamma_{w-3-F}^{b_{w-3}}$로 나누어 $y_{w-4} = y/\gamma_{w-3-F}^{b_{w-3}}$를 구한다.

이러한 과정을 반복하면 나머지 비트들도 구할 수 있다. 계산 과정에서 매번 나눗셈을 하여야 하나, 나누는 수가 상수이므로 그 역수를 LUT에 저장한 뒤 곱해서 수행할 수 있다. 그러므로 γ_{i-F}를 저장할 LUT와 그 역수를 저장할 LUT, 그리고 역수를 곱할 곱셈기로 구현할 수 있다.

그림 3이 그 회로도이다. 매 싸이클마다 y_i 를 γ_{i-F}와 비교하여 더 크거나 같으면 b_i 는 1이고 작으면 b_i 는 0이다. 다음으로 y_i에 γ_{i-F}의 역수를 곱하여 y_{i-1}를 구한 뒤 다음 비트에 대한 과정을 되풀이한다.

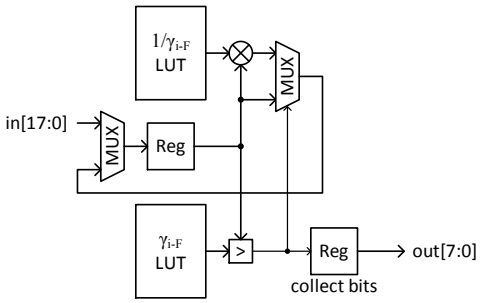

Fig. 3 Iterative LOG Circuit

이 때 필요한 LUT의 크기는, w가 8비트라고 가정할 때 entry의 수가 8개가 되고, 각 entry의 내용이 지수승 값들이므로 비트 폭은 16비트가 된다. 그러므로 LUT의 크기는 8x16= 128비트이다. 이 크기는 γ_{i-F} LUT와 그 역수 LUT가 같다. 즉 매우 큰 로그 연산 LUT가 아주 작은 두 개의 LUT와 곱셈기로 대체되는 것이다.

Ⅳ. 실험 결과

세 가지 방식으로 SoftMax 층을 구현할 경우에 대해 실험하였다. 첫 번째 방식 Conventional1은 지수 연산과 로그 연산을 모두 LUT로 구현한 것이고, Conventional2는 지수 연산을 순차적으로 연산하는 것이고, 마지막으로 Proposed는 제안한 방법과 같이 로그 연산도 순차적으로 연산한 것이다. 입력값의 폭은 8비트이며, 이 중 1비트는 부호비트, 3비트는 정수 부분, 4비트는 소수부분이다. 지수승 결과의 폭은 로그 연산에서 테이블 방식을 이용할 경우 16비트, 제안하는 방식으로 할 경우에는 18비트를 이용하였다. 테이블 방식의 경우 지수승 결과의 폭이 18비트이면 테이블의 entry 수가 너무 많아져서 (2^{18} = 256K개) 구현이 어려워진다.

Table. 1 Area comparison (㎛2)

	Conventional1[1]	Conventional2[2]	Proposed
EXP	1658	4802	13116
LOG	146436	146432	8067
Total	149839	152932	22975

표 1에서는 각 방식으로 SoftMax 층 연산 회로를 구현한 뒤 GlobaFoundry 65nm공정 라이브러리로 합성하여 EXP 블록, LOG 블록, 그리고 전체 면적을 비교하였다. 표에 따르면 지수 연산을 순차적 방식으로 교체할 경우 면적이 오히려 증가한다는 것을 알 수 있다. 이것은 지수 연산을 위한 LUT의 크기보다 순차적 방식에 사용되는 곱셈기의 크기가 더 크기 때문이다. 만일 지수 연산 회로의 입력 비트수가 16비트이면 순차적 방식이 더 적은 면적을 보일 것 같으나, 이 논문과 같이 8비트 입력이면 순차적 방식이 오히려 면적이 더 크다.

제안한 방식의 경우 지수 연산 결과의 비트수가 18비트이므로 EXP 블록의 크기가 늘어나지만, LOG 블록의 면적이 크게 줄어들어서 전체 면적은 기존의 15% 이하이다. 즉, 제안한 방식을 통해 SoftMax 층 구현 회로의 면적을 크게 줄일 수 있는 것이다.

Table. 2 RMS error

	Conventional1[7]	Conventional2[8]	Proposed
RMS Error	0.027	0.077	0.045

기존의 방식과 제안하는 방식의 연산 결과 에러를 표 2에서 비교하였다. 실제와 비슷한 연산을 하기 위해 VOC 데이터셋으로 객체 검출하는 상황을 가정하여 21개의 데이터(N=21)에 대해 SoftMax 연산을 한다고 가정하였고, 임의로 21개의 데이터를 10000번 생성하였다. 이 데이터들의 지수승을 구하고 합한 뒤 로그 연산을 수행하였으며, 부동소수점 연산을 한 결과와의 RMS (root-mean-square) 에러를 비교하였다. 제안한 방식은 Conventional2에 비해서 에러가 더 작으며, Conventional1에 비해서는 조금 더 크나 표 1과 같이 면적을 크게 줄일 수 있다.

V. 결 론

이 논문에서는 CNN 등에서 많이 사용되는 SoftMax 층의 연산 회로 면적을 줄이기 위해 연속 근사 방식의 로그 연산 회로를 제안하였다. 출력 비트 자리 각각에 해당하는 지수승 값보다 현재의 값이 더 크면 출력의 해당 비트는 1이 되며, 해당 지수승 값의 역수를 곱하여 현재 값을 조절한다. 이 방법을 이용하면 하나의 LUT 대신 작은 두 개의 LUT와 곱셈기로 로그 연산을 수행할 수 있다. 제안된 회로는 로그 연산이 필요한 다양한 분야에 활용될 수 있다.

ACKNOWLEDGEMENT

This work was also supported by the 2019 Professor Education and Research Promotion Program of KOREATECH and also supported by IDEC (EDA Tool).

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proceedings of Advances in Neural Information Processing Systems*, pp. 1097-1105, 2012.

[2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *Proceedings of International Conference on Learning Representations*, pp. 1-14, 2015.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions,“ in *Proceedings of Computer Vision and Pattern Recognition*, pp. 1-9, 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of Computer Vision and Pattern Recognition*, pp. 770-778, 2016.

[5] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “SSD: Single shot multibox detector,” in *Proceedings of European Conference on Computer Vision*, pp. 21-37, 2016.

[6] Y.-J. Kim and E.-G. Kim, “Image based fire detection using convolutional neural network,” *Journal of the Korea Institute of Information and Communication Engineering*, vol. 20, no. 9, pp. 1649-1656, Sep. 2016.

[7] B. Yuan, “Efficient hardware architecture of softmax layer in deep neural network,” in *Proceedings of IEEE International System-on-Chip Conference (SOCC)*, pp. 323-326, 2016.

[8] J. Park and D. Jeon, “Designing neuromorphic processor with on-chip learning,” *IDEC Journal of Integrated Circuits and Systems*, vol. 6, no. 2, pp. 1-6, Apr. 2020.