PositNN: Tapered Precision Deep Learning Inference for the Edge

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

The performance of neural networks, especially the currently popular form of deep neural networks, is often limited by the underlying hardware. Computations in deep neural networks are expensive, have large memory footprint, and are power hungry. Conventional reduced-precision numerical formats, such as fixed-point and floating point, cannot accurately represent deep neural network parameters with a nonlinear distribution and small dynamic range. Recently proposed posit numerical format with tapered precision represents small values more accurately than the other formats. In this work, we propose an ultra-low precision deep neural network, PositNN, that uses posits during inference. The efficacy of PositNN is demonstrated on a deep neural network architecture with three datasets (MNIST, Fashion MNIST and Cifar-10), where an 8-bit PositNN outperforms other {5-8}-bit low-precision neural networks and a 32-bit floating point baseline network.

1 Introduction

Hierarchical representation learning [1] frameworks such as deep neural networks (DNNs) have achieved state-of-the-art accuracy in a wide spectrum of applications, such as computer vision [2], medical applications [3], and natural language processing [4]. Typically, DNN inference requires millions of parameters and billions of floating point operations (*e.g.*, $\approx$ 12.5 billion FLOPs for DenseNet [2] with 264 layers). DNNs are pushing towards memory bandwidth of $\approx$ 900GB/s for training and $\approx$ 400GB/s for inference, pushing designers to choose high-bandwidth memory. Implementing these high-bandwidth, memory intensive DNN operations on the edge, such as IoT and smart wearables, is progressively onerous due to the limited compute and memory available on-device [5, 6]. Applications that impose near instantaneous inference, demand decentralization for improved security or reduced bandwidth on servers, or customization on the edge. In several of these application scenarios, deploying conventional DNNs on the end device is prohibitive because of the data movement cost, long latencies, and increased power dissipation. Recent studies address this problem using new compute and memory efficient DNN architectures, parameter efficient neural networks [7, 8], pruning and truncation, distillation and low-precision arithmetic [9]. Amongst the techniques to compress neural network parameters, low-precision arithmetic has been most successful in reducing latency, memory requirements, and power consumption, especially for DNN inference [10, 9].

Linear and nonlinear quantization are the common approaches in low-precision arithmetic [11, 9]. However, the accuracy of inference is degraded when the DNN parameters are quantized to ultra-low bit-precision $\leq$ 8-bit [12]. To recover this accuracy degradation, the network can be retrained or the number of parameters can be significantly increased [11, 13]. This results in increased computational complexity, as DNN training requires $\approx 3\times$ more computation compared to DNN inference [14]. Reducing the precision of learned parameters directly to a lower precision numerical format (fixed-point, floating point, posit [15]) has shown to mitigate the overhead of quantization and retraining

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

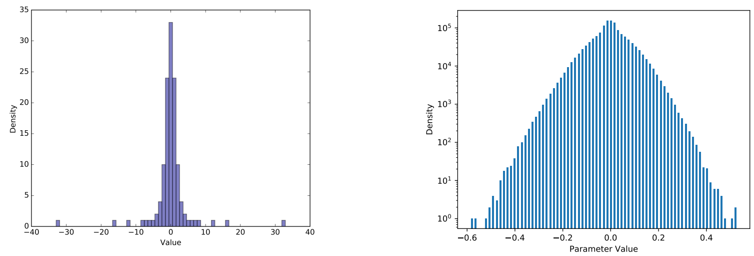

Figure 1: (a) 7-bit posit numerical value distribution; and (b) ConvNet weight distribution. Both show heavy clustering in [-1,+1] range.

[16]. In this paper, we present ultra low precision deep neural networks using posit numerical format (PositNN). The advantage of this network is that the posit numbers are represented in a nonlinear tapered precision similar to the DNN inference parameters (shown in Fig. 1). Furthermore, we address the accuracy degradation of rounding by using exact dot product algorithm for multiply and accumulate operations. The key concept is to postpone the rounding operation to the accumulation phase.

2 Posit Numerical Format

The posit numerical format, a Type III unum, is a tapered accuracy numerical format that represents real numbers [15]. Numbers with a smaller exponent are represented more accurately in comparison to the numbers with large absolute exponents since the exponent has approximately a Gaussian distribution.

The value of a posit number is represented by Equation 1, where s represents the sign, es and fs represent the maximum number of bits allocated for the exponent and fraction, respectively, e and f indicate the exponent and fraction values, respectively, and k, as computed by Equation 2, represents the regime value.

$$x = \begin{cases} 0, & \text{if } (\texttt{00...0}) \\ NaR, & \text{if } (\texttt{10...0}) \\ (-1)^s \times 2^{2^{es} \times k} \times 2^e \times \left(1 + \frac{f}{2^{fs}}\right), & \text{otherwise} \end{cases} \tag{1}$$

The regime bit field is encoded based on the *runlength* m of identical bits $(r...r)$ terminated by either a *regime terminating bit* $\overline{r}$ or the end of the n-bit value. Note that there is no requirement to distinguish between negative and positive zero since only a single bit pattern $(\texttt{00...0})$ represents zero. Furthermore, instead of defining a NaN for exceptional values and infinity by various bit patterns, a single bit pattern $(\texttt{10...0})$, "Not-a-Real" (NaR), represents exception values and infinity. More details about the posit number format can be found in [15].

$$k = \begin{cases} -m, & \text{if } r = 0 \\ m + 1, & \text{if } r = 1 \end{cases} \tag{2}$$

3 PositNN Architecture

The PositNN architecture is shown in Fig. 2 wherein each feature F_i of the convolutional layers is extracted by $F_i = B_i + \sum_{i=0}^{C \times R \times S} A_i \times W_i$ where B_i indicates the bias term, W_i is the weights matrix, A_i represents the activation, and (C, R, S) are filter parameters: the number of filter channels, the filter height, and the number of filter weights, respectively. The extracted features are used for classification, computed by $Y = B_i + \sum_{i=0}^{N} A_i \times W_i$ where Y and N represents the number of nodes and the number of outputs in each fully connected layer, respectively. Based on these equations, the fundamental computation in the convolutional and fully connected layers is the MAC operation. In this work, as shown in Fig. 2, the MAC operations of the convolutional and fully connected layers in the DNN are customized. Specifically, the MAC operation, which is commonly performed by the inexact fused-multiply accumulation algorithm, is calculated by using the Exact Dot Product (EDP) algorithm where the rounding operation within MAC operations is postponed until every product has been accumulated, which minimizes the MAC arithmetic error.

To perform the EDP, firstly, the 32-bit high-precision floating point weights and activations are quantized to the $\leq$ 8-bit low-precision posit weights and activations. To manage the overflow and underflow during conversion, the high-precision values that lie outside posit dynamic range are clipped ($\text{clip}(x)$) to either the format's maximum or minimum. The unexceptional values during conversion are rounded to the nearest number ($\phi(\cdot)$) that can be represented in the desired posit number system. To compute the products, the posit weights and activations are multiplied in a posit format without rounding or truncation at the end of multiplications to preserve precision. To avoid rounding during accumulation, the products are stored in a wide register with a width of $Q_r = 2^{es+2} \times (n-2) + 2 + \lceil \log_2(N_{op}) \rceil$. The products are then converted to fixed-point format $FX_{(m_k, n_k)}$, where $m_k = 2^{es+1} \times (n-2) + 2 + \lceil \log_2(N_{op}) \rceil$ and $n_k = 2^{es+1} \times (n-2)$. Finally, the N_{op} fixed-point products are accumulated, and the result is converted back to the posit format.

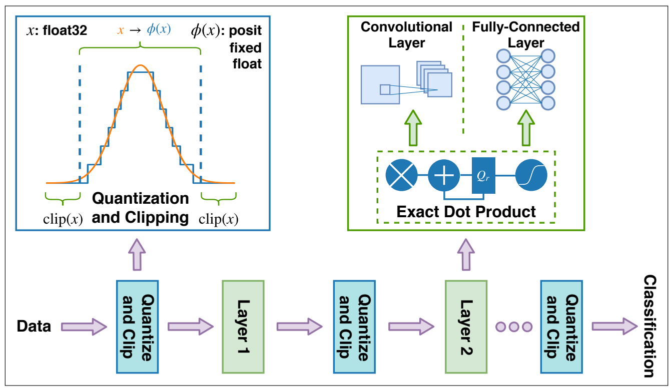

Figure 2: The proposed tapered precision PositNN architecture.

4 Experimental Results

The accuracy of PositNN for the two benchmark tasks on 32-bit floating point and for {5-8}-bit low precision formats are shown in Table 1 and 2. All the networks are implemented using Keras [17] and TensorFlow [18]. Universal library [19] is used for posit and floating point number exact dot product operation. PositNN with 8-bit ($es = 2$) precision outperformed {5-8}-bit floating point, fixed point and 32-bit floating point numerical format. Furthermore, when comparing 5-bit precision networks, PositNN demonstrates 6.67% improvement over 5-bit floating-point on MNIST dataset. We hypothesize that PositNN has better accuracy at lower-bit precisions, as the non-linear distribution of posit numbers is similar to the DNN inference parameters unlike in fixed or float. This holds true, especially in the range $[-4, 4]$, where most of the DNN inference parameters take place. For 1% accuracy degradation in PositNN (5-bit), we estimate that there is 84.4% reduction in memory storage for the respective inference parameters when compared to a 32-bit floating point numerical format.

Table 1: Specifications of the benchmark tasks and performance on a baseline 32-bit floating point network. Each inference set consists of 10,000 samples.

Task	Dataset	Network	Layers[1]	# Parameters	# EDP OPs[2]	# Memory	Top-1 Accuracy
Digit classification	MNIST	Convnet	2 Conv, 2 FC and 1 PL	1.40 M	58.7 k	5.84 MB	99.32%
Image classification	Fashion MNIST	Convnet	2 Conv, 3 FC, 2 PL and 1 BN	1.88 M	69.8 k	7.77 MB	92.54%
Image classification	CIFAR-10	Convnet	7 Conv, 1 FC, 3 PL	0.95 M	312.6 k	6.23 MB	76%

[1] Conv: 2D convolutional layer; FC: fully-connected layer; PL: max/Average pooling layer; BN: batch normalization layer.

[2] The number of EDP operations for a single sample.

Table 2: PositNN accuracy on two datasets with {5-8}-bit precision compared to fixed and float (Respective best results are when posit has $es \in \{0, 1, 2\}$ and floating point $w_e \in \{3, 4\}$).

	Posit				Float				Fixed			
Datasets	8-bit	7-bit	6-bit	5-bit	8-bit	7-bit	6-bit	5-bit	8-bit	7-bit	6-bit	5-bit
MNIST	**99.35%**	**99.33%**	**99.16%**	**98.94%**	99.34%	99.25%	98.82%	92.27%	99.18%	97.14%	97.08%	96.96%
Fashion MNIST	**92.70%**	**92.60%**	**91.64%**	**88.92%**	92.63%	91.77%	84.21%	68.21%	89.59%	88.63%	85.31%	83.46%
Cifar-10[1]	**76%**	**76%**%	**70%**%	**48%**	75%	**76%**	51%	13%	19%	17%	14%	15%

[1] 100 randomly selected samples.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, vol. 1, p. 3, 2017.

[3] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[4] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado, *et al.*, "Google's multilingual neural machine translation system: Enabling zero-shot translation," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 339–351, 2017.

[5] M. Verhelst and B. Moons, "Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to iot and edge devices," *IEEE Solid-State Circuits Magazine*, vol. 9, no. 4, pp. 55–65, 2017.

[6] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks," *Nature Electronics*, vol. 1, no. 4, p. 216, 2018.

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[8] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun, "Sbnet: Sparse blocks network for fast inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8711–8720, 2018.

[9] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[10] S. Hashemi, N. Anthony, H. Tann, R. Bahar, and S. Reda, "Understanding the impact of precision quantization on the accuracy and energy of neural networks," in *Proceedings of the Conference on Design, Automation & Test in Europe*, pp. 1478–1483, European Design and Automation Association, 2017.

[11] A. Mishra and D. Marr, "Wrpn & apprentice: Methods for training and inference using low-precision numerics," *arXiv preprint arXiv:1803.00227*, 2018.

[12] A. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," *arXiv preprint arXiv:1711.05852*, 2017.

[13] Z. He, B. Gong, and D. Fan, "Optimize deep convolutional neural network with ternarized weights and high accuracy," *arXiv preprint arXiv:1807.07948*, 2018.

[14] F. Iandola, "Exploring the design space of deep convolutional neural networks at large scale," *arXiv preprint arXiv:1612.06519*, 2016.

[15] J. L. Gustafson and I. T. Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, pp. 71–86, 2017.

[16] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

[17] F. Chollet *et al.*, "Keras." `https://github.com/keras-team/keras`, 2015.

[18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[19] T. Omtzigt *et al.*, "Universal: a c++ template library for universal number arithmetic." `https://github.com/stillwater-sc/universal`, 2015.