Deep Learning Inference on Embedded Devices: Fixed-Point vs Posit

Seyed H. F. Langroudi, Tej Pandit, Dhireesha Kudithipudi
Neuromorphic AI Lab, Rochester Institute of Technology
Rochester, New York 14623
{*sf3052,tnp5438,dxkeec*}*@rit.edu*

***Abstract*—Performing the inference step of deep learning in resource constrained environments, such as embedded devices, is challenging. Success requires optimization at both software and hardware levels. Low precision arithmetic and specifically low precision fixed-point number systems have become the standard for performing deep learning inference. However, representing non-uniform data and distributed parameters (e.g. weights) by using uniformly distributed fixed-point values is still a major drawback when using this number system. Recently, the posit number system was proposed, which represents numbers in a non-uniform manner. Therefore, in this paper we are motivated to explore using the posit number system to represent the weights of Deep Convolutional Neural Networks. However, we do not apply any quantization techniques and hence the network weights do not require re-training. The results of this exploration show that using the posit number system outperformed the fixed point number system in terms of accuracy and memory utilization.**

***Keywords*-Deep convolutional neural network; Low precision representation; Posit number system**

I. Introduction

Deep learning, as a particular form of hierarchical representational learning [1], has shown promise in several applications such as computer vision [2], natural language processing [3], speech recognition [4], robotics [5] and medical applications [6]. The success of deep learning stems from its ability to learn from raw and unstructured data [1]. Deep Convolutional Neural Networks (DCNN) are commonly used in Deep learning, with stochastic gradient descent as their learning mechanism [7].

Although DCNNs achieve state-of-the-art accuracy as compared to other machine learning approaches, they exhibit shortcomings such as long latency, power inefficiency, and long training durations. For instance, training ResNet-50 (50 layers) [8] on the ImageNet dataset [9] requires 256 GPUs [10]. Another example, AlphaGo, was trained for months with 1202 CPUs and 176 GPUs to beat Lee Sedol, an 18-time world champion, at the strategic board game "Go" [11]. According to these examples, training deep neural networks even with the resources in data centers has a lot of limitations. On the other hand, deep learning inference is less complex than deep learning training. Moreover, the limitations for implementing deep learning inference on conventional hardware such as CPUs and GPUs has been addressed by digital neuromorphic chips such as TPU [12]. However, this chip is specially designed for data centers. Therefore, designs for digital neuromorphic chips to implement DCNNs with real-time performance on low-power embedded platforms, mobile devices, and IoT devices, are currently in the research exploration phase [13], [14]. Low precision arithmetic is a common approach to reduce power consumption and improve the real-time performance of deep learning applications on embedded devices.

Among different number systems used for performing deep learning inference with low precision arithmetic, the fixed-point number system shows the most promising trade-off between accuracy and computational complexity [15]–[18]. However, real numbers are represented uniformly by a fixed-point number system which is not suitable for deep learning applications since the weights and data have a non-uniform distribution [16]. Recently, a posit number system was proposed as an alternative to the floating point number system [19]. This number system has a unique non-linear numerical representation characteristic for all numbers in a dynamic range which distinguishes it from other number systems such as fixed and floating point. As a result, in this introductory paper, we are motivated to explore the use of the posit representation in DCNNs for digit recognition and image classification tasks.

We compare the fixed-point number system and the posit number system to represent weights of three DCNNs with 4, 5 and 8 layers on MNIST [20], Cifar-10 [21] and ImageNet [9] datasets respectively. The posit number system outperformed the fixed-point number system in terms of accuracy and memory utilization when the two number systems are compared under the constraint that they both have the same dynamic range ([-1,1]).

The rest of this paper presents previous works on low precision deep learning inference and introduces the posit number system in section II, the proposed DCNNs using the posit number system are discussed in section III, the comparative results, in terms of accuracy and memory utilization, between the posit number system and the fixed-point number system to represent weights of proposed DCNNs are presented in section IV.

978-1-5386-7367-6/18/$31.00 ©2018 IEEE
DOI 10.1109/EMC2.2018.00012

II. Background

A. Low precision deep learning inference

In recent literature studies on DCNNs, focusing on improving the computational efficiency during inference by using limited precision for weights and activations, Judd et al. represent weights by the dynamic fixed-point number system and perform computations using the floating point number system [22]. In this approach, the energy consumption for memory access operations obtained while implementing different deep neural networks on various datasets is reduced by an average of 15% [22]. Following this research, the 8-bit floating point number system used to represent weights of AlexNet and VGG-16 [23] and was evaluated on the ImageNet dataset [24]. The results indicated that it is possible to represent 20% of the weights in the 8-bit floating point representation with less than 1% degradation of accuracy. Finally, Gysel et al. successfully performed deep learning inference using the AlexNet architecture on the ImageNet dataset, with 8-bit dynamic fixed-point weights and 8-bit dynamic fixed-point data, resulting in less than 1% degradation of accuracy [16]. However, the networks needed to be retrained in order to attain this level of accuracy.

After the success of performing deep learning inference by using an 8-bit precision representation of weights and data, researchers have been further motivated to squeeze the representation to below 8-bits, in particular, the 1-bit (binarized representation) [17], [25] and 2-bit (ternarized representation) [18], [26]. Although, by using these representations, multiplication operations in a deep neural network are removed or converted to sign detection operations, the corresponding significant degree of degradation in accuracy overwhelms the computational advantage. Therefore, evaluating a deep learning inference model with 8 layers or more (e.g. AlexNet, GoogLeNet) on large datasets (e.g., ImageNet), with less than 8 bits to represent each of the weights and data values, without substantial accuracy degradation and/or retraining, is still an open question.

B. Posit number system

The posit number system is a type of tapered accuracy number system [27], which means that numbers with small exponents are more accurate than numbers with large exponents [19]. The challenges encountered by the floating-point number system such as manipulating overflow, underflow, double zero and exception are addressed in this number system. The posit number system format defined as $P_{(n,es)}$ where *n* refers to the total number of bits in this system and *es* indicates the number of exponent bits [19]. Each number in this system, as shown by Eq. 1 [19], is indicated by $useed$, $exponent$, r_{value} (responsible for finding the number regime) and $fraction$ (to indicate the precision).

$$X = (-1)^{sign} \times (useed)^{r_{value}} \times 2^{exponent} \times (1 + fraction) \quad (1)$$

For instance, 2.56 in the posit number system with $P_{(16,1)}$ format is represented by 4 as a $useed$, 1 as an $exponent$, 0 as an r_{value} and 0.280 as a $fraction$ as shown by Fig. 1. Note that the conversion from decimal floating point to posit numbers are explained in detail in section III.

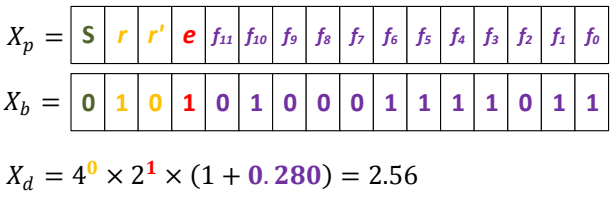


Figure 1. Representation of a number in the posit number system with $P_{(16,1)}$ format [19]

III. DCNNs with posit representation

In this paper, we explore the effects of using the posit number system (used to represent weights), on the accuracy and memory utilization of the DCNN during inference. To achieve this goal, the weights are converted from the original single floating number system to the new posit number system during memory read and write operations. On the other hand, the posit number system is converted back to a single floating point number system during computational operations as needed by standard computing architectures. The proposed DCNNs architecture is shown in Fig. 2. This architecture is similar to DCNNs architecture which is proposed by [22] except we use a posit number system which has advantages to represent weights of DCNNs non-uniformly. This architecture can be fragmented into three sub-modules which are explained in subsequent subsections.

A. Conversion from posit to floating point

The first step is to convert the posit number to a decimal floating point number and then convert to a binary floating point number. The conversion from a posit to a decimal floating point number is divided in four steps [19]: (i) extracting the sign bit; (ii) extracting the regime bit; (iii) extracting the exponent bit; (iv) extracting the fraction bit. The most significant bit in posit representation indicates the sign bit. The regime bit is represented by unary arithmetic [19]. Therefore, when extracting the regime value (r_{value}), the algorithm starts to count the number of consecutive one's or zero's after a sign bit until it reaches a bit of the opposite value (zero or one respectively). Then, the result is negated if the bits counted are zeros or is decremented by 1 if the bits counted are ones. The exponent bits are represented by an unsigned integer and are thus easily extracted from the posit number bit string. The rest of bits in the bit string are fraction bits. The decimal floating point number is converted into a binary floating point number by dividing or multiplying by 2 until the number is in the range of $[1, 2)$ [19].

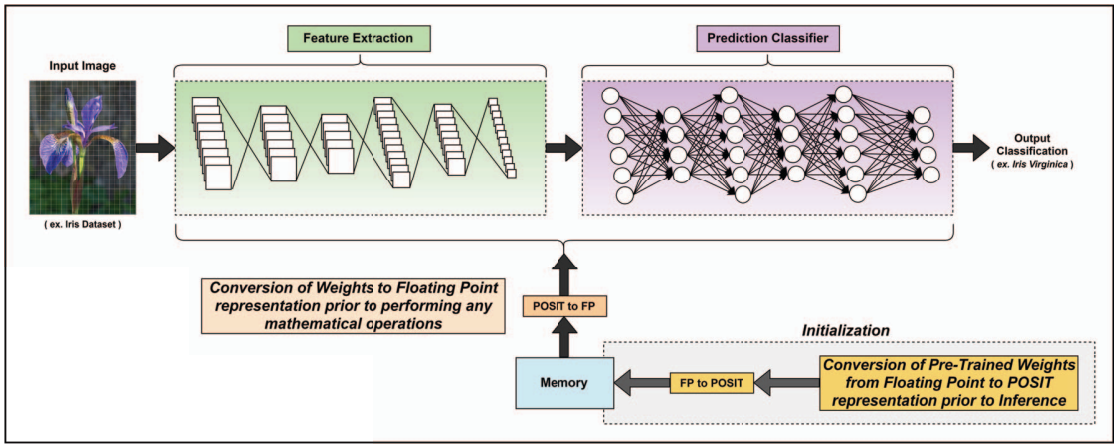


Figure 2. High level overview of the DCNN architecture implementation which uses the posit number system to represent the weights

B. DCNNs

In DCNNs, the features are extracted using convolutional layers. This feature is shown by vector $F_n = (f_1, f_2, ..., f_m)$, where n indicates the number of images in the dataset, and m shows the feature vector dimension. Then the features are classified by a network of fully connected layers. In the last layer, the softmax layer is used as a classifier to minimize $y - f^*(x, w)$; where y defines a label, x denotes an input to the softmax layer, w denotes the weights in softmax layer and f^* is the best approximation function [7].

C. Conversion from floating point to posit

This conversion consists of two steps [19]: (i) converting the binary floating point number system to the decimal floating point number system; (ii) converting the binary floating point number system to the posit number system. The first conversion is performed by multiplying the fraction by two raised to the power of the exponent. Then the decimal floating point number achieved in the first step is converted into a posit number by dividing or multiplying it by two until the number is in the range $[1, useed)$ in order to find the regime bit. Then, this process is continued until the number is in the range $[1, 2)$ in order to find the exponent. The remaining bits are the fraction [19].

IV. Evaluation

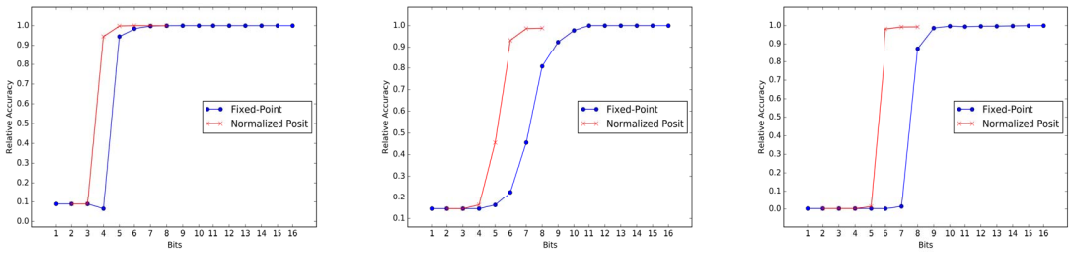
The new approach is evaluated on three datasets: (i) MNIST dataset; (ii) CIFAR-10 dataset; (iii) subset of the ImageNet dataset. The MNIST dataset (handwritten numerical digits dataset), and others datasets are collected for assessing the performance of new techniques on basic image recognition tasks. Different DCNNs are used for each dataset, and the single floating point number system is selected for a baseline implementation. The baseline is implemented using the Keras API [28] and the accuracy results are as shown in Table I.

Table I. Top-1 accuracy of 3 different neural networks

Task	Dataset	# inference set	Network	layers	Top-1 accuracy
Digit classification	MNIST	10000	LeNet	2 Conv and 2 FC	99.03%
Image classification	CIFAR-10	10000	Convnet	3 Conv and 2 FC	68.45%
Image classification	ImageNet	10000	AlexNet	5 Conv and 3 FC	55.45%

In this paper, the weights are represented by a variable length fixed-point number system (with a maximum bit length of 16 bits) and 8-bit posit number system. To represent weights in the variable fixed-point number system only one bit is considered for the integer part and the fractional part is varied in a range of [0,15] bits, as most of the weights in well explored DCNNs are in the $[-1, 1]$ interval. To represent weights with the posit number system, we selected the $P_{(i,0)}$ format where i is varied within the range [2,8]. Note that the exponent selected is zero. The reason behind this selection is because the dynamic range of the posit number system with a zero exponent is the closest approximation to the weights' dynamic range, as compared to other possible options for exponent value. Among these posit formats, $P_{(2,0)}$ has the smallest dynamic range ($[-1, 1]$), while other posit formats have a larger dynamic range. However, variable length fixed point number systems have the same dynamic range. Therefore, we are motivated to normalize all the formats in posit number system and call it the normalized posit number system. In this version of the posit number system, all of the formats have the same dynamic range of $[-1, 1]$. The relative accuracy results for different tasks are shown in Fig. 3.

The normalized posit number system outperformed other number systems in terms of accuracy with fewer bits. The results demonstrate that it is possible to perform LeNet, ConvNet, and AlexNet with 5 bits, 7 bits and 7 bits respectively, using the posit number system, with less than 1% accuracy degradation in comparison to performance of the same networks with 7 bits, 11 bits and 9 bits respectively while using the variable length fixed point number



(a) (b) (c)

Figure 3. Results showing the relative accuracy to the baseline for DCNN implementations on various datasets with representation of weights using variable length fixed point and normalized posit number systems. (a) Relative Accuracy results for LeNet on MNIST dataset. (b) Relative Accuracy results for ConvNet on Cifar-10 dataset. (c) Relative Accuracy results for AlexNet on ImageNet dataset.

system. This reduces memory utilization by 28.6%, 36.4% and 23% as compared to standard state of the art variable length fixed point implementations [22], [29], and can also significantly reduce the number of memory accesses through memory concatenation schemes. Note that this improvement is achieved without using quantization or retraining the DCNNs.

V. Conclusion

We explore using the posit number system to represent weights of three DCNNs on MNIST, Cifar-10, and ImageNet datasets. The normalized posit number system outperformed the fixed point number system in terms of accuracy, with fewer number of bits used to represent weights. By using a memory concatenation encoding scheme, the number of memory accesses required to transfer the weights reduces significantly, as well as the total energy consumption for the same task. For future work, we will explore the effect of low precision data representation using the posit number system and implement DCNNs using the posit representation for both storage and computation.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[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, no. 2, 2017, p. 3.

[3] 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.

[4] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International Conference on Machine Learning*, 2016, pp. 173–182.

[5] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3389–3396.

[6] 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.

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

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[9] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

[10] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[11] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[12] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-L. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. Mackean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick,

N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit TM," pp. 1–17, 2017.

[13] D. Li, X. Wang, and D. Kong, "Deeprebirth: Accelerating deep neural network execution on mobile devices," *arXiv preprint arXiv:1708.04728*, 2017.

[14] S. Kodali, P. Hansen, N. Mulholland, P. Whatmough, D. Brooks, and G.-Y. Wei, "Applications of deep neural networks for ultra low power iot," in *2017 IEEE 35th International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 589–592.

[15] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, N. E. Jerger, and A. Moshovos, "Proteus: Exploiting numerical precision variability in deep neural networks," in *Proceedings of the 2016 International Conference on Supercomputing*. ACM, 2016, p. 23.

[16] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented Approximation of Convolutional Neural Networks," *Iclr*, p. 8, 2016. [Online]. Available: http://arxiv.org/abs/1604.03168

[17] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *arXiv:1609.07061*, 2016. [Online]. Available: https://arxiv.org/abs/1609.07061

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

[19] 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.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] B. Graham, "Fractional max-pooling," *arXiv preprint arXiv:1412.6071*, 2014.

[22] P. Judd, J. Albericio, T. Hetherington, T. Aamodt, N. E. Jerger, R. Urtasun, and A. Moshovos, "Reduced-precision strategies for bounded memory in deep neural nets," *arXiv preprint arXiv:1511.05236*, 2015.

[23] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[24] Q. C. Zhaoxia Deng, Cong Xu and P. Faraboschi, "Reduced-precision memory value approximation for deep learning," *HP*, December 2015.

[25] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[26] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[27] R. Morris, "Tapered floating point: A new floating-point representation," *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1578–1579, 1971.

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

[29] P. Judd, J. Albericio, T. Hetherington, T. Aamodt, N. E. Jerger, R. Urtasun, and A. Moshovos, "Proteus: Exploiting precision variability in deep neural networks," *Parallel Computing*, 2017.