SmaQ: Smart Quantization for DNN Training by Exploiting Value Clustering

Nima Shoghi, Andrei Bersatti, Moinuddin Qureshi, and Hyesoon Kim

Abstract—Advancements in modern deep learning have shown that deeper networks with larger datasets can achieve state of the art results in many different tasks. As networks become deeper, the memory requirement of neural network training proves to be the primary bottleneck of single-machine training. In this letter, we first study the characteristics of neural network weight, gradient, feature map, gradient map, and optimizer state distributions for some popular neural network architectures. Our investigation shows that the majority of the data structures used by neural networks can have their value distributions be approximated with normal distributions. We then introduce Smart Quantization (SmaQ), a quantization scheme that exploits this observed normal distribution to quantize the data structures. Our dynamic quantization method calculates the sampled mean and standard deviation of tensors and quantizes each tensor element to 6 or 8 bits based on the z-score of that value. Our scheme reduces the memory usage during training by up to 6.7x with minor losses in accuracy.

Index Terms—Approximate methods, machine learning, quantization

1 Introduction

Deep neural networks have been very successful at many different tasks, such as computer vision and natural language processing. The abundance of available training data, as well as the existence of ever more powerful processing units (GPGPUs and TPUs), have resulted in deeper models capable of learning more complex relationships. Recent deep learning research has shown that increasing the model depth and the dataset size leads to higher performance in most deep learning tasks. At the same time, this increase can be problematic, as the memory capacity of GPUs and TPUs is quite limited. Traditional data parallelism techniques will, thus, struggle to fit the architecture's weights, gradients, and feature maps onto memory, which has become a bottleneck. As a result, methods that decrease memory utilization during training can be very valuable, as they allow for deeper models to be trained at higher training batch sizes.

We study the value distribution of different neural network data structures and observe that these distributions have the majority of their data in clusters close to the mean and thus can be modeled with a normal distribution. We therefore introduce a quantization technique, *SmaQ*, that exploits this observed distribution and, thus, utilizes estimated statistical variables, mean and standard deviation, to convey the tensor information in a summarized, encoded form. We then evaluate our method, comparing our technique's memory usage and neural network inaccuracy as a product of data loss to other similar methods.

In this work, we aim to make progress towards a fully quantized training technique. While there is significant active research into reducing the memory footprint of DNN training, the existing solutions either are too complex, expensive, do not reduce the memory usage enough, or degrade training-accuracy.

In general, reduced precision datatypes and quantization have been the techniques most recently investigated in the pursuit of solutions. Reduced precision datatypes of 16 bits have been used before in practice. The IEEE 754-2019 standard defines the IEEE FP16 format as containing 5 exponent bits and 10 mantissa bits. Note that it has 3 less bits on the exponent than the IEEE FP32 format, thus reducing its dynamic range. To increase the dynamic range while still using 16 bits only (including the sign bit), the BFloat16 (BF16) [6] format uses 8 bits for the exponent and 7 for the mantissa.

Even lower bit-width precision datatypes and/or quantizations that are suitable for DNN training have been difficult to be achieve. These have resulted in a degree of information loss and numerical instability issues such as swamping [11], which is a characteristic of floating point addition and it arises from the truncation that occurs whenever a large number is added to a small number [5]. Wang *et al.* [11] proposed an 8-bit floating point format, with a 5-bit exponent and a 2-bit mantissa, but which requires chunk-based accumulation, stochastic rounding, and maintaining the first and the last layers in 16-bit precision. Park *et al.* proposed V-Quant[8], which is a quantization technique similar to ours in philosophy in that it attempts to quantize each element based on its value, relative to its distribution. V-Quant uses full precision values for elements with large values — identified through sorting the input data, and proposes compressing such outliers with conventional sparse data representation such as CSR. Finally, V-Quant only quantizes feature maps in most experiments. For its LSTM language model experiments, V-Quant only quantizes the weights, noting that LSTM's feature map distribution does not follow the assumption of the paper and thus is not well-suited for V-Quant. V-Quant's need to sort the input data and usage of the CSR format make it suboptimal to use in large scales. In order to address the existing challenges with the current methods, Cambier *et al.* [2] proposed *Shifted and Squeezed* 8-bit floating point (*S2FP8*), which is the most closely related work to our proposed technique in that it uses mean and standard deviation of the data to relay information. S2FP8 attempts to address the precision challenge of 8-bit training without requiring chunking, stochastic rounding, loss scaling, or maintaining some full or half precision layers. It involves encoding values into transformed values where the transformations shift and squeeze the values in order to be able to cover the wide dynamic range with the reduced bit-width. It uses 32-bit floating point number statistics (the shift and the squeeze statistics) to transform the values and requires expensive exponentiation in calculating its squeeze parameter.

2 Background and Motivation

2.1 Quantizing Neural Network Data Structures

The different data structures that we can attempt to quantize during neural network training include weights, feature maps (forward pass activation values), gradients, gradient maps (intermediate partial derivatives calculated in the backward pass and immediately used in the next layer), and optimizer-specific data structures such as momentum vectors. Different works in prior art have attempted to quantize or compress different subsets of these data structures (e.g., weights only) or treat different data structures differently, but our method quantizes all these data structures and applies the same technique for all, reducing complexity.

2.2 Characteristics of Neural Network Memory

Fig. 1 shows histograms for the weight, feature map, gradient, and gradient map values values of one of the convolution layers in ResNet 34 [4]. Most importantly, these distributions are consistently observed across most evaluated cases. We notice that the majority of

• *The authors are with the Georgia Institute of Technology, Atlanta, GA 30332 USA. E-mail: nimashoghi@gmail.com, {abersatti, moin}@gatech.edu, hyesoon@cc.gatech.edu.*

Manuscript received 25 June 2021; revised 4 Aug. 2021; accepted 23 Aug. 2021. Date of publication 30 Aug. 2021; date of current version 24 Sept. 2021.
(Corresponding author: Nima Shoghi.)
Digital Object Identifier no. 10.1109/LCA.2021.3108505

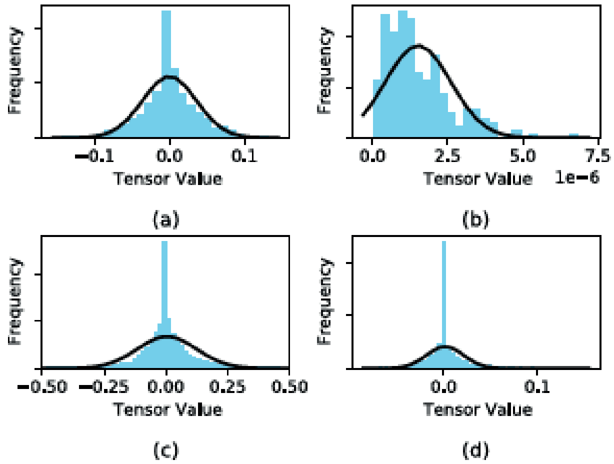

Fig. 1. (a) and (b) show the weight distribution for a convolution and downsample layer in ResNet 34, respectively, while (c) and (d) show the feature map and gradient values for the same convolution layer. Normal distributions, using the mean and standard deviation of the data, are overlayed on the histograms.

the values within these distributions are contained in a very close bounding box around the mean and that these distributions are even more tightly clustered around the mean than the normal distribution. We believe that an encoding mechanism that exploits this heavily concentrated nature of neural network data structure distributions can potentially introduce some major memory savings for both neural network training and inference.

3 Algorithm

The primary motivation behind our scheme is that, in the observed distributions of neural network data structure values, the majority of the data is contained within the cluster closest to the mean. We will refer to these values as *inliers*, and the rest of the data as *outliers*. Our mechanism exploits this property to use a lower-bit representation for inliers and a higher-bit representation for outliers. To encode, we first normalize the input to its z-score form, which has a sensible range. We store the mean and standard deviation as metadata. Then, we quantize the normalized tensor. To decode, we dequantize the quantized tensor into the normalized format. Then, we retrieve the tensor's metadata and use it to undo the normalization process. Fig. 2 shows an example of this encoding scheme in effect.

3.1 Normalizing the Data

One major challenge of using quantization is that we need to find sensible lower and upper bounds for the input data. Quantization methods for inference usually employ observers during the training process to find these bounds for weights. This is much harder to do for fully quantized training, however, and involves tedious hyperparameter adjustment. Our method does not require this kind of fine-tuning.

Therefore, to encode some input, we first *normalize* it to its z-score format, which has a sensible range of $(\mu - 2\sigma, \mu + 2\sigma)$. As such, we define our normalization and de-normalization functions, λ and λ^{-1}, as

$$\lambda(\mathbf{X} \sim (\mu, \sigma)) = \frac{\mathbf{X} - \mu}{\sigma} \tag{1}$$

$$\lambda^{-1}(\mathbf{X}_{normalized}, \mu, \sigma) = \mathbf{X} \cdot \sigma + \mu. \tag{2}$$

3.2 Quantizing the Normalized Data

In uniform quantization to N bits, the real value input space is uniformly divided into 2^N discrete buckets, and the bucket index is

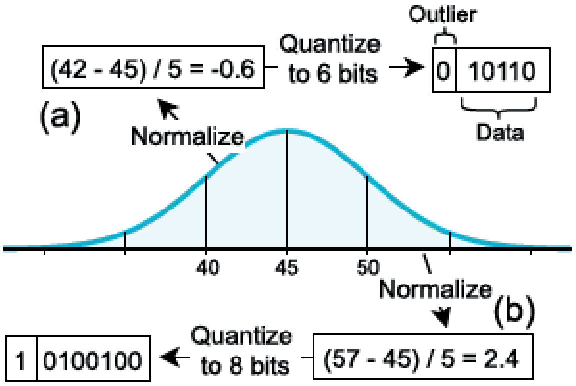

Fig. 2. (a) shows an example of encoding a value within our bounding box, which is quantized to 6 bits (outlier bit = 0). (b) shows an example of encoding an outlier value, which is quantized to 8 bits (outlier bit = 1).

used as the quantized representation. Our smart quantization method builds on top of uniform quantization, but the number of bits of each quantized element depends on whether it is an inlier or an outlier.

After normalizing the data, we create a bounding-box of 1 standard deviation[1] around the mean. We treat the values within the bounding box as inliers and those outside the bounding box as outliers. We then uniformly quantize the inliers — which have a range of $[\mu - \sigma, \mu + \sigma]$ — to $L_{inlier} - 1$ bits and the outliers — which have a range of $[\mu - 2\sigma, \mu - \sigma) \cup (\mu + \sigma, \mu + 2\sigma]$ — to $L_{outlier} - 1$ bits. Each input element's final representation will contain 1 metadata bit which indicates whether that element is an inlier or an outlier. For example, SmaQ (6, 8) will encode inliers to 6 bits (5 data bits and 1 outlier metadata bit) and outliers to 8 bits (7 data bits and 1 outlier metadata bit).

Our uniform quantization function takes the real-valued input and discretizes it by linearly interpolating the z-score value onto the range of the desired integer. After the interpolation, we use stochastic rounding to truncate the value into an integer.

4 Evaluation

4.1 Methodology

In this section, we evaluate our quantization method by encoding some very popular neural network architectures during training. This allows us to measure the accuracy loss and storage benefit as a result of different methods. We compare our results to FP32, FP16, BF16, FP8, and S2FP8 [2]. To simulate S2FP8, we use the following truncation equation[2]:

$$\mathbf{X}_{S2FP8} = [2^{-\beta}\{\text{truncate}_{FP8}(2^{\beta}|\mathbf{X}|^{\alpha})\}]^{1/\alpha}. \tag{3}$$

Any usable encoding method must be generally applicable to most classes of popular neural network architectures. In our experiments, we evaluate this quality by running our tests on the following architectures: Inception Networks (InceptionNet) [9], Residual Networks (ResNet) [4], and Bidirectional Encoder Representations from Transformers (BERT) networks [3]. InceptionNets and ResNets are two of the most popular neural network architectures for computer vision applications such as image classification and semantic segmentation. BERT is one of the most widely used networks for natural language processing tasks like machine translation and semantic similarity. We train ResNet and InceptionNet on the CIFAR10 image classification task and BERT on the Semantic Textual Similarity (STS) task from the GLUE dataset [10].

[1] The size of the bounding box is an adjustable hyperparameter.
[2] This is taken from Equation (5) of the S2FP8 paper.

TABLE 1
Experimental Results

	InceptionNet		ResNet 34		BERT	
Encoding	Accuracy	Storage Benefit	Accuracy	Storage Benefit	Accuracy	Storage Benefit
SmaQ (6,8)	92.22%	**4.93**	84.74%	**4.92**	**0.794**	**4.95**
S2FP8	25.13%	3.87	20.32%	3.87	N/A	N/A
FP8	15.99%	4.00	18.80%	4.00	0.766	4.00
FP16	18.24%	2.00	22.38%	2.00	0.793	2.00
BF16	**92.49%**	2.00	89.00%	2.00	0.791	2.00
FP32	92.14%	1.00	**89.20%**	1.00	0.793	1.00

The networks are trained using the PyTorch framework. For ResNet, we use the stochastic gradient descent optimizer with a learning rate of 0.003, momentum of 0.9, and weight decay of 0, and we train for 100 epochs. For BERT, we use the AdamW optimizer [7] with a learning rate of 2e-5 and weight decay of 0.01, and we train for 6 epochs.

We simulate the inaccuracy of encoding and decoding during training for every data structure by encoding and immediately decoding the data structure (we refer to this as *applying inaccuracy*). This inaccuracy is applied at the following steps:

- After the forward pass of each computation graph node, we apply inaccuracy to the *feature maps*.
- After the backward pass of each computation graph node, we apply inaccuracy to the *gradient maps*.
- Before PyTorch's parameter update stage (i.e., optimizer step), we apply inaccuracy to the *gradients*.
- After PyTorch's parameter update stage, we apply inaccuracy to *weights* and *optimizer state*.

For each network, we evaluate the *accuracy* and *storage benefit* of training the network with each encoding mechanism. The accuracy metric depends on the network evaluated: For InceptionNet and ResNet 34, the accuracy is the top-1 validation accuracy. For BERT, the accuracy is the mean of the Pearson and Spearman coefficients. The storage benefit refers to the average memory savings — relative to FP32 — of all encoded tensors over the entire training run. Its equation can be found in Equation (4). The additional 64 bits in the numerator are added to account for the mean and standard deviation storage cost

$$Storage\ Benefit = E\left(\frac{\sum[64 + (6 \cdot N_{inlier} + 8 \cdot N_{outlier})]}{\sum[32 \cdot (N_{inlier} + N_{outlier})]}\right). \tag{4}$$

We measure the overhead of our software emulation framework by measuring the training time of the network with and without our instrumentation. Our profiling results show that the framework increases training time by adding a factor of 2.5x (e.g., if the baseline trains in 1 second, SmaQ trains in 3.5 seconds). This limitation makes evaluating very large datasets time-consuming. It's important to note, however, that this overhead is a result of software emulation. With proper hardware and software optimization effort, a performance-focused implementation of SmaQ would have a much more manageable overhead.

4.2 Results

Table 1 shows the accuracy and storage benefit for the InceptionNet, ResNet, and BERT experiments. The takeaways from our experimental results are explained in this section.

4.2.1 Accuracy Loss and Storage Benefit

SmaQ attains a top-1 accuracy of 92.22% on InceptionNet, 84.74% on ResNet, and a Pearson and Spearman coefficient mean of 0.794 on BERT. S2FP8, on the other hand, attained 25.13% on InceptionNet, 20.32% on ResNet, and was not even able to complete training on BERT due to numerical instability issues leading to NaN and infinity float values being produced. FP8, similarly, produced lower accuracy values across the board: 15.99% on InceptionNet, 18.80% on ResNet, and 0.766 on BERT. This data shows that SmaQ heavily outperforms other encoding mechanisms that produce competitive Storage Benefit (S2FP8 and FP8). FP16 shows similar results to FP8, producing 18.24% on InceptionNet, 22.38% on ResNet, and 0.793 on BERT, all of which are lower than SmaQ's.

SmaQ's accuracy values are even competitive with FP32 and BF16, which set the current industry standard for neural network floating point value encoding. FP32 and BF16, respectively, produce accuracy values of 92.14% and 92.49% on InceptionNet, 89.20% and 89.00% on ResNet, and 0.793 and 0.791 on BERT. SmaQ's accuracy results outperform FP32 and BF16 in BERT and are within 5 percentage points in ResNet34. Most importantly, SmaQ exhibits these competitive accuracy values while using the least amount of memory, with a Storage Benefit of 4.93x, 4.92x, 4.93x for Inception, ResNet, and BERT, respectively.

4.2.2 Gradient Map and Optimizer State Encoding

Our results show S2FP8 is not able to perform well in any of the evaluated tasks. Further investigation shows that S2FP8 struggles with gradient map and optimizer state encoding. The original authors of the S2FP8 paper did not explicitly intend for S2FP8 to be used for gradient map or optimizer state encoding. For gradient maps, this is because they are usually immediately consumed after allocation, making them a suboptimal target for memory compression. We, however, believe that it is important for an encoding mechanism to be robust to all kinds of data distributions observed during DNN training, and that robustness of an encoding mechanism on different types of data distributions observed during DNN training can increase the reliability of the encoding mechanism as a whole. Our experimental results in Table 1 show that S2FP8 lacks robustness and is suboptimal for encoding gradient maps and optimizer state. For ResNet, S2FP8 produced 20.32% accuracy and for BERT, the training loop stopped due to detected NaN and infinite values for the loss and weights.

We re-ran the BERT and ResNet experiments with gradient map and optimizer state encoding disabled. This allows us to compare our algorithm's performance to S2FP8 under the same conditions (i.e., with the same data structures being encoded) that the S2FP8 authors evaluated. Table 2 shows these results. While S2FP8

TABLE 2
No Gradient Map and Optimizer State Encoding

	ResNet 34		BERT	
Encoding	Accuracy	Storage Benefit	Accuracy	Storage Benefit
SmaQ	86.96%	**4.88x**	0.795	**4.95x**
S2FP8	86.46%	3.87x	N/A	N/A
FP8	85.22%	4x	0.766	4x
FP16	89.03%	2x	**0.796**	2x
BF16	88.22%	2x	0.792	2x
FP32	**89.20%**	1x	0.793	1x

performs much better with a ResNet accuracy of 86.49%, SmaQ is still outperforms S2FP8 with a ResNet accuracy of 86.86%. SmaQ does this while using less memory than S2FP8, with SmaQ exhibiting a storage benefit of 4.88x, compared to S2FP8's 3.87x. S2FP8 still shows numerical instability problems for BERT.

5 Conclusion

We present *Smart Quantization (SmaQ)*, an encoding scheme that exploits the observed normal distributions to quantize neural network data structures. SmaQ is an appealing option for neural network training as it can be applied to weights, feature maps, gradients and optimizer state. SmaQ (6, 8), on average, used 1.5 less bits than FP8 or S2FP8, all the while exhibiting higher accuracy values than S2FP8, FP8, and even FP16. SmaQ's accuracy is competitive with FP32 and BF16 SmaQ achieves these results while maintaining a simple implementation at its core. Given the mean and standard deviation, SmaQ only requires N multiplications and N additions for encoding and decoding of an entire tensor of size N, and one additional division than that for encoding. Our future work includes exploring implementation optimizations, such as the ones below:

- We can partition the encoded tensor into chunks and decode the chunks in parallel. Outlier bits can be stored in a structure of arrays format as a single bitmask, reducing the need to scan data values. Padding bytes can be used to deal with alignment issues.
- Using reservoir sampling algorithms, we can take a sample from the input tensor and calculate the mean and standard deviation of that sample.
- The standard deviation can be estimated using a method similar to range batch-normalization [1] through the usage of the product of a scale adjustment variable and the range of the input sample.
- Calculated statistics can be re-used per iteration, removing the need for further calculation of mean and standard deviation beyond the initial computation.

Acknowledgments

The authors would like to thank the reviewers for their valuable suggestions. This work was supported in part by the NSF CSR under Grant 1815047 and in part by Facebook. Nima Shoghi and Andrei Bersatti are equal contributors.

References

[1] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," in *Proc. 32nd Int. Conf. Neural Inform. Process. Syst.*, 2018, pp. 5151–5159.

[2] L. Cambier, A. Bhiwandiwalla, T. Gong, O. H. Elibol, M. Nekuii, and H. Tang, "Shifted and squeezed 8-bit floating point format for low-precision training of deep neural networks," in *Proc. 8th Int. Conf. Learn. Representations,* 2020. [Online]. Available: https://openreview.net/forum?id=Bkxe2AVtPS

[3] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics Human Lang. Technol.*, 2019, pp. 4171–4186.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Computer Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[5] N. J. Higham, "The accuracy of floating point summation," *SIAM J. Sci. Comput.*, vol. 14, pp. 783–799, 1993.

[6] D. D. Kalamkar *et al.*, "A study of BFLOAT16 for deep learning training," *CoRR*, vol. abs/1905.12322, 2019.

[7] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *Proc. Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7

[8] E. Park, S. Yoo, and P. Vajda, "Value-aware quantization for training and inference of neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 608–624.

[9] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.

[10] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proc. Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rJ4km2R5t7

[11] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 7686–7695.