Scalable Methods for 8-bit Training of Neural Networks

Ron Banner[1]*, **Itay Hubara**[2]*, **Elad Hoffer**[2]*, **Daniel Soudry**[2]
{itayhubara, elad.hoffer, daniel.soudry}@gmail.com
{ron.banner}@intel.com

(1) Intel - Artificial Intelligence Products Group (AIPG)
(2) Technion - Israel Institute of Technology, Haifa, Israel

Abstract

Quantized Neural Networks (QNNs) are often used to improve network efficiency during the inference phase, i.e. after the network has been trained. Extensive research in the field suggests many different quantization schemes. Still, the number of bits required, as well as the best quantization scheme, are yet unknown. Our theoretical analysis suggests that most of the training process is robust to substantial precision reduction, and points to only a few specific operations that require higher precision. Armed with this knowledge, we quantize the model parameters, activations and layer gradients to 8-bit, leaving at a higher precision only the final step in the computation of the weight gradients. Additionally, as QNNs require batch-normalization to be trained at high precision, we introduce Range Batch-Normalization (BN) which has significantly higher tolerance to quantization noise and improved computational complexity. Our simulations show that Range BN is equivalent to the traditional batch norm if a precise scale adjustment, which can be approximated analytically, is applied. To the best of the authors' knowledge, this work is the first to quantize the weights, activations, as well as a substantial volume of the gradients stream, in all layers (including batch normalization) to 8-bit while showing state-of-the-art results over the ImageNet-1K dataset.

1 Introduction

Deep Neural Networks (DNNs) achieved remarkable results in many fields making them the most common off-the-shelf approach for a wide variety of machine learning applications. However, as networks get deeper, using neural network (NN) algorithms and training them on conventional general-purpose digital hardware is highly inefficient. The main computational effort is due to massive amounts of multiply-accumulate operations (MACs) required to compute the weighted sums of the neurons' inputs and the parameters' gradients.

Much work has been done to reduce the size of networks. The conventional approach is to compress a trained (full precision) network [5, 24, 15] using weights sharing, low rank approximation, quantization, pruning or some combination thereof. For example, Han et al., 2015 [9] successfully pruned several state-of-the-art large-scale networks and showed that the number of parameters can be reduced by an order of magnitude.

Since training neural networks requires approximately three times more computation power than just evaluating them, quantizing the gradients is a critical step towards faster training machines. Previous work demonstrated that by quantizing network parameters and intermediate activations during the training phase more computationally efficient DNNs could be constructed.

*Equal contribution

Researchers [8, 6] have shown that 16-bit is sufficient precision for most network training but further quantization (i.e., 8-bit) results with severe degradation. Our work is the first to almost exclusively train at 8-bit without harming classification accuracy. This is addressed by overcoming two main obstacles known to hamper numerical stability: batch normalization and gradient computations.

The traditional batch normalization [14] implementation requires the computation of the sum of squares, square-root and reciprocal operations; these require high precision (to avoid zero variance) and a large dynamic range. It should come as no surprise that previous attempts to use low precision networks did not use batch normalization layers [26] or kept them in full precision [29]. This work replaces the batch norm operation with range batch-norm (range BN) that normalizes inputs by the range of the input distribution (i.e., $\max(x) - \min(x)$). This measure is more suitable for low-precision implementations. Range BN is shown analytically to approximate the original batch normalization by multiplying this range with a scale adjustment that depends on the size of the batch and equals to $(2 \cdot \ln(n))^{-0.5}$. Experiments on ImageNet with Res18 and Res50 showed no distinguishable difference between accuracy of Range BN and traditional BN.

The second obstacle is related to the gradients quantization. Given an upstream gradient g_l from layer l, layer $l-1$ needs to apply two different matrix multiplications: one for the layer gradient g_{l-1} and the other for the weight gradient g_W which are needed for the update rule. Our analysis indicates that the statistics of the gradient g_l violates the assumptions at the crux of common quantization schemes. As such, quantizing these gradients constitutes the main cause of degradation in performance through training. Accordingly, we suggest to use two versions of layer gradients g_l, one with low-precision (8-bit) and another with higher-precision (16-bit). The idea is to keep all calculations with g_l that does not involve a performance bottleneck at 16 bits, while the rest at 8 bits. As the gradients g_W are required only for the weight update, they are computed using the 16 bits copy of g_l. On the other hand, the gradient g_{l-1} is required for the entire backwards stream and as such it is computed using the corresponding 8-bit version of g_l. In most layers of the DNN these computations can be performed in parallel. Hence g_W can be computed at high precision in parallel with g_{l-1}, without interrupting the propagation of g_l to lower layers. We denote the use of two different arithmetic precision operations in the differentiation process as "Gradients Bifurcation".

2 Previous Work

While several works [8, 6] have shown that training at 16-bit is sufficient for most networks, more aggressive quantization schemes were also suggested [29, 21, 19, 13]. In the extreme case, the quantization process used only one bit which resulted in binarized neural networks (BNNs) [12] where both weights and activations were constrained to -1 and 1. However, for more complex models and challenging datasets, the extreme compression rate resulted in a loss of accuracy. Recently, Mishra et al. [20] showed that this accuracy loss can be prevented by merely increasing the number of filter maps in each layer, thus suggesting that quantized neural networks (QNNs) do not possess an inherent convergence problem. Nevertheless, increasing the number of filter maps enlarge quadratically the number of parameters, which raises questions about the efficiency of this approach.

In addition to the quantization of the forward pass, a growing interest is directed towards the quantization of the gradient propagation in neural networks. A fully quantized method, allowing both forward and backward low-precision operations will enable the use of dedicated hardware, with considerable computational, memory, and power benefits. Previous attempts to discretize the gradients managed to either reduce them to 16-bit without loss of accuracy [6] or apply a more aggressive approach and reduce the precision to 6-8 bit [29, 13] with a noticeable degradation. Batch normalization is mentioned by [26] as a bottleneck for network quantization and is either replaced by a constant scaling layer kept in full precision, or avoided altogether; this clearly has some impact on performance (e.g., AlexNet trained over ImageNet resulted with top-1 error of 51.6%, where the state of the art is near 42%) and better ways to quantize normalization are explicitly called for. Recently L1 batch norm with only linear operations in both forward and backward propagation was suggested by [27, 11] with improved numerical stability. Yet, our experiments show that with 8-bit training even L1 batch norm is prone to overflows when summing over many large positive values. Finally, Wen et al. [25] focused on quantizing the gradient updates to ternary values to reduce the communication bandwidth in distributed systems.

We claim that although more aggressive quantization methods exist, 8-bit precision may prove to have a "sweet-spot" quality to it, by enabling training with no loss of accuracy and without modifying the original architecture. Moreover, we note that 8-bit quantization is better suited for future and even current hardware, many of which can already benefit from 8-bit operations [22]. So far, to the best of our knowledge, no work has succeeded to quantize the activations, weights, and gradient of all layers (including batch normalization) to 8-bit without any degradation.

3 Range Batch-Normalization

For a layer with $n \times d-$dimensional input $x = (x^{(1)}, x^{(2)}, ..., x^{(d)})$, traditional batch norm normalizes each dimension

$$\hat{x}^{(d)} = \frac{x^{(d)} - \mu^d}{\sqrt{\text{Var}[x^{(d)}]}}, \tag{1}$$

where μ^d is the expectation over $x^{(d)}$, n is the batch size and $\text{Var}[x^{(d)}] = \frac{1}{n}||x^{(d)} - \mu^d||_2^2$. The term $\sqrt{\text{Var}[x^{(d)}]}$ involves sums of squares that can lead to numerical instability as well as to arithmetic overflows when dealing with large values. The Range BN method replaces the above term by normalizing according to the range of the input distribution (i.e., $\max(\cdot) - \min(\cdot)$), making it more tolerant to quantization. For a layer with $d-$dimensional input $x = (x^{(1)}, x^{(2)}, ..., x^{(d)})$, Range BN normalizes each dimension

$$\hat{x}^{(d)} = \frac{x^{(d)} - \mu^d}{C(n) \cdot \text{range}(x^{(d)} - \mu^d)}, \tag{2}$$

where μ^d is the expectation over $x^{(d)}$, n is the batch size, $C(n) = \frac{1}{\sqrt{2 \cdot \ln(n)}}$ is a scale adjustment term, and $\text{range}(x) = \max(x) - \min(x)$.

The main idea behind Range BN is to use the scale adjustment $C(n)$ to approximate the standard deviation σ (traditionally being used in vanilla batch norm) by multiplying it with the range of the input values. Assuming the input follows a Gaussian distribution, the range (spread) of the input is highly correlated with the standard deviation magnitude. Therefore by normalizing the range by $C(n)$ we can estimate σ. Note that the Gaussian assumption is a common approximation (e.g., Soudry et al. [23]), based on the fact that the neural input $x^{(d)}$ is a sum of many inputs, so we expect it to be approximately Gaussian from the central limit theorem.

We now turn to derive the normalization term $C(n)$. The expectation of maximum of Gaussian random variables are bounded as follows [16]:

$$0.23\sigma \cdot \sqrt{\ln(n)} \leq E[\max(x^{(d)} - \mu^d)] \leq \sqrt{2}\sigma\sqrt{\ln(n)}. \tag{3}$$

Since $x^{(d)} - \mu^d$ is symmetrical with respect to zero (centred at zero and assumed gaussian), it holds that $E[\max(\cdot)] = -E[\min(\cdot)]$; hence,

$$0.23\sigma \cdot \sqrt{\ln(n)} \leq -E[\min(x^{(d)} - \mu^d)] \leq \sqrt{2}\sigma\sqrt{\ln(n)}. \tag{4}$$

Therefore, by summing Equations 3 and 4 and multiplying the three parts of the inequality by the normalization term $C(n)$, Range BN in Eq. 2 approximates the original standard deviation measure σ as follows:

$$0.325\sigma \leq C(n) \cdot \text{range}(x^{(d)} - \mu^d) \leq 2 \cdot \sigma$$

Importantly, the scale adjustment term $C(n)$ plays a major role in RangeBN success. The performance was degraded in simulations when $C(n)$ was not used or modified to nearby values.

4 Quantized Back-Propagation

Quantization methods: Following [28] we used the GEMMLOWP quantization scheme as decribed in Google's open source library [2]. A detailed explanation of this approach is given in

Appendix B. While GEMMLOWP is widely used for deployment, to the best of the authors knowledge this is the first time GEMMLOWP quantization is applied for training. Note that the activations maximum and minimum values were computed by the range BN operator, thus finding the normalization scale as defined in Appendix B does not require additional $O(n)$ operations.

Finally we note that a good convergence was achieved only by using stochastic rounding [8] for the gradient quantization. This behaviour is not surprising as the gradients will serve eventually for the weight update thus unbiased quantization scheme is required to avoid noise accumulation.

Gradients Bifurcation: In the back-propagation algorithm we recursively calculate the gradients of the loss function $\mathcal{L}$ with respect to I_ℓ, the input of the ℓ neural layer,

$$g_\ell = \frac{\partial \mathcal{L}}{\partial I_\ell}, \tag{5}$$

starting from the last layer. Each layer needs to derive two sets of gradients to perform the recursive update. The layer activation gradients:

$$g_{\ell-1} = g_\ell W_\ell^T, \tag{6}$$

served for the Back-Propagation (BP) phase thus passed to the next layer,and the weights gradients

$$g_{W_\ell} = g_\ell I_{\ell-1}^T, \tag{7}$$

used to updated the weights in layer ℓ. Since the backward pass requires twice the amount of multiplications compared to the forward pass, quantizing the gradients is a crucial step towards faster training machines. Since g_ℓ, the gradients streaming from layer ℓ, are required to compute $g_{\ell-1}$, it is important to expedite the matrix multiplication described in Eq.6. The second set of gradient derive in Eq.7 is not required for this sequential process and thus we choose to keep this matrix multiplication in full precision. We argue that the extra time required for this matrix multiplication is comparably small to the time required to communicate the gradients g_ℓ. Thus, in this work the gradients used for the weight gradients derivation are still in float. In section 6, we show empirically that bifurcation of the gradients is crucial for high accuracy results.

Straight-Through Estimator: Similar to previous work [12, 20], we used the straight-through estimator (STE) approach to approximate differentiation through discrete variables. This is the most simple and hardware friendly approach to deal with the fact that the exact derivative of discrete variables is zero almost everywhere.

5 When is quantization of neural networks possible?

This section provides some of the foundations needed for understanding the internal representation of quantized neural networks. It is well known that when batch norm is applied after a convolution layer, the output is invariant to the norm of the weight on the proceeding layer [14] i.e., $BN(C \cdot W \cdot x) = BN(W \cdot x)$ for any given constant C. This quantity is often described geometrically as the norm of the weight tensor, and in the presence of this invariance, the only measure that needs to be preserved upon quantization is the *directionality* of the weight tensor. In the following we show that quantization preserves the direction (angle) of high-dimensional vectors when W follows a Gaussian distribution.

More specifically, for networks with M-bit fixed point representation, the angle is preserved when the number of quantization levels 2^M is much larger than $\sqrt{2\ln(N)}$, where N is the size of quantized vector. This shows that significant quantization is possible on practical settings. Taking for example the dimensionality of the joint product in a batch with 1024 examples corresponding to the last layer of ResNet-50, we need no more than 8-bit of precision to preserve the angle well (i.e., $\sqrt{2\ln(3 \cdot 3 \cdot 2048 \cdot 1024)} = 5.7 << 2^8$). We stress that this result heavily relays on values being distributed according to a Gaussian distribution, and suggests why some vectors are robust to quantization (e.g., weights and activations) while others are more fragile (e.g., gradients).

5.1 Problem Statement

Given a vector of weights $W = (w_0, w_1, ..., w_{N-1})$, where the weights follow a Gaussian distribution $W \sim N(0, \sigma)$, we would like to measure the cosine similarity (i.e., cosine of the angle) between W and $Q(W)$, where $Q(\cdot)$ is a quantization function. More formally, we are interested in estimating the following geometric measure:

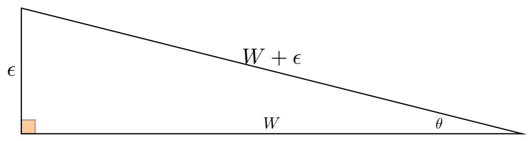

Figure 1: Graphic illustration of the angle between full precision vector W and its low precision counterpart which we model as $W + \epsilon$ where $\epsilon \sim \mathbb{U}(-\Delta/2, \Delta/2)$

$$\cos(\theta) = \frac{W \cdot Q(W)}{||W||_2 \cdot ||Q(W)||_2} \tag{8}$$

We next define the quantization function $Q(\cdot)$ using a fixed quantization step between adjacent quantified levels as follows:

$$Q(x) = \Delta \cdot \left(\left\lfloor \frac{x}{\Delta} \right\rfloor + \frac{1}{2} \right), \text{ where } \Delta = \frac{\max(|W|)}{2^M} \tag{9}$$

We consider the case where quantization step Δ is much smaller than mean$(|W|)$. Under this assumption correlation between W and quantization noise $W - Q(W) = (\epsilon_0, \epsilon_1, ..., \epsilon_{N-1})$ is negligible, and can be approximated as an additive noise. Our model assumes an additive quantization noise $\bar{\epsilon}$ with a uniform distribution i.e., $\epsilon_i \sim \mathcal{U}[-\Delta/2, \Delta/2]$ for each index i. Our goal is to estimate the angle between W and $W + \bar{\epsilon}$ for high dimensions (i.e., $N \to \infty$).

5.2 Angle preservation during quantization

In order to estimate the angle between W and $W + \epsilon$, we first estimate the angle between W and ϵ. It is well known that if ϵ and W are independent, then at high dimension the angle between W and ϵ tends to $\frac{\pi}{2}$ [3] i.e., we get a right angle triangle with W and ϵ as the legs, while $W + \epsilon$ is the hypotenuse as illustrated in Figure 1-right. The cosine of the angle θ in that triangle can be approximated as follows:

$$\cos(\theta) = \frac{||W||}{||W + \epsilon||} \geq \frac{||W||}{||W|| + ||\epsilon||} \tag{10}$$

Since W is Gaussian, we have that $E(||W||) \cong \sqrt{N}\sigma$ in high dimensions [4]. Additionally, in Appendix A we show that $E(||\bar{\epsilon}||) \leq \sqrt{N/12} \cdot \Delta$. Moreover, at high dimensions, the relative error made as considering $E||X||$ instead of the random variable $||X||$ becomes asymptotically negligible [3]. Therefore, the following holds in high dimensions:

$$\cos(\theta) \geq \frac{\sigma}{\sigma + E(\Delta)/\sqrt{12}} = \frac{2^M \cdot \sigma}{2^M \cdot \sigma + E(\max(|W|))/\sqrt{12}} \tag{11}$$

Finally, $E(\max(W)) \leq \sqrt{2}\sigma\sqrt{\ln(N)}$ when W follows a Gaussian distribution [16], establishing the following:

$$\cos(\theta) \geq \frac{2^M}{2^M + \sqrt{\ln N}/\sqrt{6}} \tag{12}$$

Eq. 12 establishes that when $2^M >> \sqrt{\ln(N)}$ the angle is preserved during quantization. It is easy to see that in most practical settings this condition holds even for challenging quantizations. Moreover, this results highly depends on the assumption made about the Gaussian distribution of W (transition from equation 11 to equation 12).

6 Experiments

We evaluated the ideas of Range Batch-Norm and Quantized Back-Propagation on multiple different models and datasets. The code to replicate all of our experiments is available on-line [2].

[2]https://github.com/eladhoffer/quantized.pytorch

6.1 Experiment results on cifar-10 dataset

To validate our assumption that the cosine similarity is a good measure for the quality of the quantization, we ran a set of experiments on Cifar-10 dataset, each with a different number of bits, and then plotted the average angle and the final accuracy. As can be seen in Figure 2 there is a high correlation between the two. Taking a closer look the following additional observations can be made: (1) During quantization the direction of vectors is better preserved with the forward pass compared to the backward pass; (2) validation accuracy follows tightly the cosine of the angle in the backward pass, indicating gradient quantization as the primary bottleneck; (3) as expected, the bound on $E(\cos(\theta))$ in Eq. 12 holds in the forward pass, but less so in the backward pass, where the Gaussian assumption tends to break. The histograms in Figure 2 further confirms that the layer gradients g_l do not follow Gaussian distribution. These are the values that are bifurcated into low and high precision copies to reduce noise accumulation.

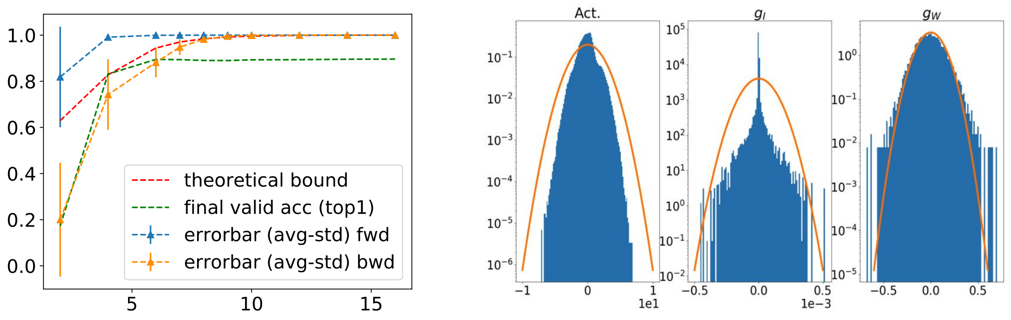

Figure 2: Left: empirical and theoretical analysis of cosine similarity, $\cos(\theta)$, with respect the number of bits used for quantization. Right: Histograms of activations, layer gradients g_l and weight gradients g_W. To emphasize that g_l do not follow a Gaussian distribution, the histograms were plotted in a log-scale (ResNet-18, Cifar-10).

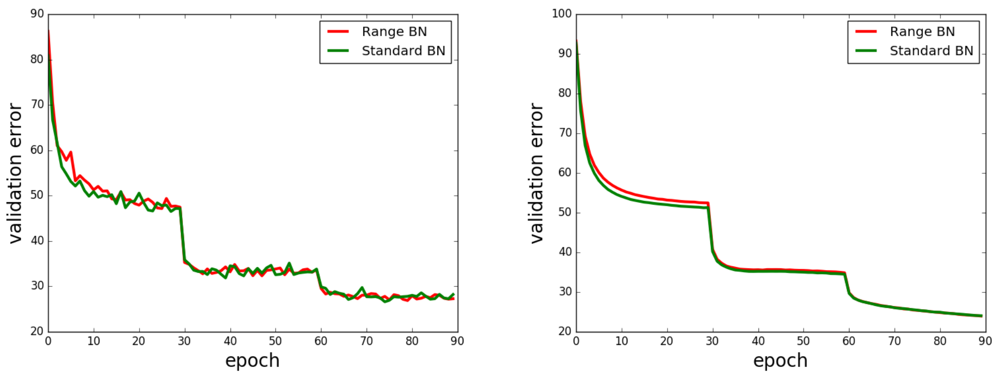

Figure 3: Equivalent accuracy with standard and range batch-norm (ResNet-50, ImageNet).

6.2 Experiment results on ImageNet dataset: Range Batch-Normalization

We ran experiments with Res50 on ImageNet dataset showing the equivalence between the standard batch-norm and Range BN in terms of accuracy. The only difference between the experiments was the use of Range BN instead of the traditional batch-norm. Figure 3 compares between the two. It shows equivalence when models are trained at high precision. We also ran simulations on other datasets and models. When examining the final results, both were equivalent i.e., 32.5% vs 32.4% for ResNet-18 on ImageNet and 10.5% vs 10.7% for ResNet-56 on Cifar10. To conclude, these simulations prove that we can replace standard batch-norm with Range BN while keeping accuracy unchanged. Replacing the sum of squares and square root operations in standard batch-norm by a few maximum and minimum operations has a major benefit in low-precision implementations.

6.3 Experiment results on ImageNet dataset: Putting it all together

We conducted experiments using RangeBN together with Quantized Back-Propagation. To validate this low precision scheme, we were quantizing the vast majority of operations to 8-bit. The only operations left at higher precising were the updates (float32) needed to accumulate small changes from stochastic gradient descent, and a copy of the layer gradients at 16 bits needed to compute g_W. Note that the float32 updates are done once per minibatch while the propagations are done for each example (e.g., for a minibatch of 256 examples the updates constitute less than 0.4% of the training effort). Figure 4 presents the result of this experiment on ImageNet dataset using ResNet18 and ResNet50. We provide additional results using more aggressive quantizations in Appedix F.

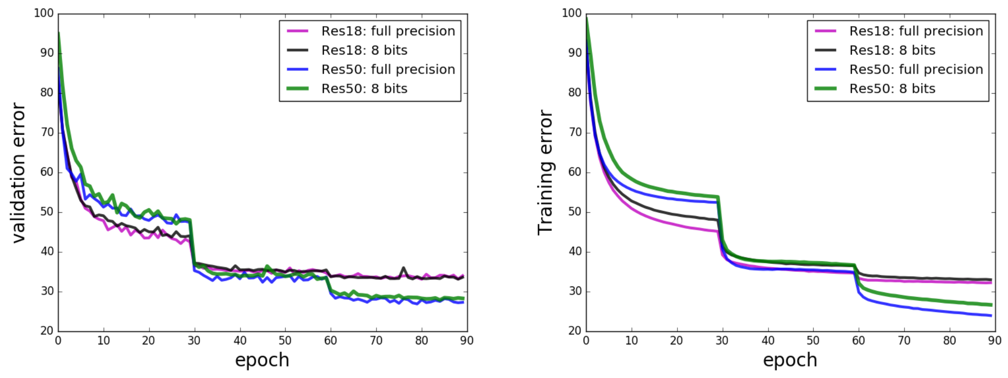

Figure 4: Comparing a full precision run against 8-bit run with Qunatized Back-Propogarion and Range BN (ResNet-18 and ResNet-50 trained on ImageNet).

7 Discussion

In this study, we investigate the internal representation of low precision neural networks and present guidelines for their quantization. Considering the preservation of direction during quantization, we analytically show that significant quantization is possible for vectors with a Gaussian distribution. On the forward pass the inputs to each layer are known to be distributed according to a Gaussian distribution, but on the backward pass we observe that the layer gradients g_l do not follow this distribution. Our experiments further assess that angle is not well preserved on the backward pass, and moreover final validation accuracy tightly follows that angle. Accordingly, we bifurcate the layer gradients g_l and use it at 16-bits for the computation of the weight gradient g_W while keeping the computation of next layer gradient g_{l-1} at 8-bit. This enables the (slower) 16-bits computation of g_W to be be done in parallel with g_{l-1}, without interrupting the propagation the layer gradients.

We further show that Range BN is comparable to the traditional batch norm in terms of accuracy and convergence rate. This makes it a viable alternative for low precision training. During the forward-propagation phase computation of the square and square root operations are avoided and replaced by $\max(\cdot)$ and $\min(\cdot)$ operations. During the back-propagation phase, the derivative of $\max(\cdot)$ or $\min(\cdot)$ is set to one where the coordinates for which the maximal or minimal values are attained, and is set to zero otherwise.

Finally, we combine the two novelties into a single training scheme and demonstrate, for the first time, that 8-bit training on a large scale dataset does not harm accuracy. Our quantization approach has major performance benefits in terms of speed, memory, and energy. By replacing float32 with int8, multiplications become 16 times faster and at least 15 times more energy efficient [13]. This impact is attained for 2/3 of all the multiplications, namely the forward pass and the calculations of the layer gradients g_l. The weight gradients g_W are computed as a product of 8-bit precision (layer input) with a 16-bit precision (unquantized version of g_l), resulting with a speedup of x8 for the rest of multiplications and at least x2 power savings. Although previous works considered an even lower precision quantization (up-to 1-bit), we claim that 8-bit quantization may prove to be more of an interest. Furthermore, 8-bit matrix multiplication is available as an off-the-shelf operation in existing hardware and can be easily adopted and used with our methods.

References

[1] Anderson, A. G. and Berg, C. P. The high-dimensional geometry of binary neural networks. *arXiv preprint arXiv:1705.07199*, 2017.

[2] Benoit, J., Pete, W., Miao, W., et al. gemmlowp: a small self-contained low-precision gemm library, 2017. https://github.com/google/gemmlowp.

[3] Biau, G. and Mason, D. M. High-dimensional p p-norms. In *Mathematical Statistics and Limit Theorems*, pp. 21–40. Springer, 2015.

[4] Chandrasekaran, V., Recht, B., Parrilo, P. A., and Willsky, A. S. The convex geometry of linear inverse problems. *Foundations of Computational mathematics*, 12(6):805–849, 2012.

[5] Chen, W., Wilson, J., Tyree, S., Weinberger, K., and Chen, Y. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pp. 2285–2294, 2015.

[6] Das, D., Mellempudi, N., Mudigere, D., et al. Mixed precision training of convolutional neural networks using integer operations. 2018.

[7] Deng, J., Dong, W., Socher, R., et al. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

[8] Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1737–1746, 2015.

[9] Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[10] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[11] Hoffer, E., Banner, R., Golan, I., and Soudry, D. Norm matters: efficient and accurate normalization schemes in deep networks. *arXiv preprint arXiv:1803.01814*, 2018.

[12] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. In *Advances in Neural Information Processing Systems 29 (NIPS'16)*, 2016.

[13] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*, 2016.

[14] Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[15] Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[16] Kamath, G. Bounds on the expectation of the maximum of samples from a gaussian. *URL http://www. gautamkamath. com/writings/gaussian max. pdf*, 2015.

[17] Krizhevsky, A. Learning multiple layers of features from tiny images. 2009.

[18] Krizhevsky, A. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[19] Lin, X., Zhao, C., and Pan, W. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, pp. 344–352, 2017.

[20] Mishra, A., Nurvitadhi, E., Cook, J. J., and Marr, D. Wrpn: Wide reduced-precision networks. *arXiv preprint arXiv:1709.01134*, 2017.

[21] Miyashita, D., Lee, E. H., and Murmann, B. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016.

[22] Rodriguez, A., Segal, E., Meiri, E., et al. Lower numerical precision deep learning inference and training. *https://software.intel.com/en-us/articles/lower-numerical-precision-deep-learning-inference-and-training*, 2018.

[23] Soudry, D., Hubara, I., and Meir, R. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems*, pp. 963–971, 2014.

[24] Ullrich, K., Meeds, E., and Welling, M. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

[25] Wen, W., Xu, C., Yan, F., et al. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in Neural Information Processing Systems*, pp. 1508–1518, 2017.

[26] Wu, S., Li, G., Chen, F., and Shi, L. Training and inference with integers in deep neural networks. *International Conference on Learning Representations (ICLR)*, 2018.

[27] Wu, S., Li, G., Deng, L., et al. L1-norm batch normalization for efficient training of deep neural networks. *arXiv preprint arXiv:1802.09769*, 2018.

[28] Wu, Y., Schuster, M., Chen, Z., et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[29] Zhou, S., Ni, Z., Zhou, X., et al. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

Appendix

A Bound on the expected norm of ϵ

By Jensen inequality the following holds true:

$$E(||\epsilon||) = E\left[\sqrt{\sum_{i=0}^{N-1} \epsilon_i^2}\right] \leq \sqrt{E\left[\sum_{i=0}^{N-1} \epsilon_i^2\right]} = \sqrt{\sum_{i=0}^{N-1} E[\epsilon_i^2]} \tag{13}$$

Given a uniform random variable in $\epsilon_i \sim U[-\Delta/2, \Delta/2]$, we next derive the expected value of its square as follows:

$$E[\epsilon_i^2] = \int_{-\Delta/2}^{\Delta/2} x^2 \cdot \frac{1}{\Delta} \cdot dx = \frac{\Delta^2}{12} \tag{14}$$

Note that Δ is a random variable. Hence, we substitute Eq 14 into Eq 13 using the following conditional expectation:

$$E(||\epsilon|| \text{ given } \Delta) \leq \sqrt{\frac{N}{12}} \cdot \Delta \tag{15}$$

We can now establish $E(||\epsilon||)$ as follows:

$$E(||\epsilon||) \leq E(E(\ ||\epsilon||\ |\Delta))|\Delta)) = \sqrt{\frac{N}{12}} \cdot E(\Delta) \tag{16}$$

B Quantization methods

Following [28] we choose to use the GMMLOWP quantization scheme as decribed in Google's open source library [2]. Given an input tensor x, clamping values $[v_{\min}, v_{\max}]$,and number of bits M we set the output to be:

$$\begin{aligned} \text{scale} &= (v_{\max} - v_{\min})/2^M \\ \text{zero} - \text{point} &= \text{round}(\min(\max(-v_{\min}/\text{scale}, 0), 2^M)) \\ \text{output} &= \text{round}(x/\text{scale} + \text{zero} - \text{point}) \end{aligned}$$

The clamping values for the weights and activations were defined as the input's absolute maximum and minimum. Since the activations can have a high dynamic range which can be aggressively clamped as shown by [28] we defined its clamping values to be the average of absolute maximum and minimum values of K chunks. This reduces the dynamic range Variance and allows smaller quantization steps.

it is important to note that a good convergence was achieved only by using stochastic rounding [8]. This behaviour is not surprisings as the gradients serves eventually for the weight update thus unbias quantization scheme is required to avoid quantization noise accumulation.

C Additional Experiments

In this section we present our more aggressive quantization experiments of the Quantized Back-Propagation scheme. In the extreme case, QBP ternarizes the gradients and uses only 1-bit for the weights and activations. In this case, we refer to QBP networks as Ternarized Back-Propagation (TBP), in which all forward MACs operations can be replaced with XNOR and population count (*i.e.,* counting the number of ones in the binary number) operations. To avoid significant degradation in test accuracy, we apply stochastic ternarization and increase the number of filter maps in a each convolution layer.

C.1 CIFAR10

A well studied dataset is the CIFAR10 image classification benchmark first introduced by Krizhevsky [17]. CIFAR10 consists of a training set of size 50K, and a test set of size 10K color images. Here, each images represents one of the following categories: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships and trucks.
We trained a VGG-like network similar to the one suggested by Hubara et al. [12] on the CIFAR10 dataset, with the same hyper-parameters used in the original work. We compared two variants: the original model BNN model and the BNN model trained with TBP (this paper), where ternarized gradients are used.
The results shown in Table (2) demonstrate that if we inflate the convolutional filters by 3 we can achieve similar results as the BNN and full precision models achieved. This is in accordance with previous finding [20], that found that widening the network can mitigate accuracy drop inflicted by low precision training. To make sure this is not a unique case for BNN we also applied TBP on ResNet with depth of 18. As can be seen from Table (2), as before, inflating the network improves performance, until it is only 1% from the original performance, after inflating it by 5.

C.2 ImageNet

Next, we applied TBP to the more challenging ImageNet classification task introduced by Deng et al. [7]. It consists of a training set of size 1.2M samples and a test set of size 50K. Each instance is labeled with one of 1000 categories including objects, animals, scenes, and even some abstract shapes. We report two error rates for this dataset: top-1 and top-5, as is typical done. Top-k error rate represent the fraction of test images for which the correct label is not among the k most probable labels predicted by the model.

We run several experiments on AlexNet inflated by 3. Similarly to previous work and to ease the comparison we kept first and last layer in full precision. With binarized weights, 4-bit activations and gradients, TBP converged to 53.3% top-1 accuracy and 75.84% top-5 accuracy. By using only 2bit activation TBP reached 49.6% top-1 accuracy and 73.1% top-5 accuracy. We are currently working on more advanced typologies such as ResNet-50 model [10]. Results are summarized in Table (1).

Table 1: Classification top-1 validation error rates of TBP BNNs trained on ImageNet with AlexNet topology. I, A, W, G stands for Inflation, Activation bits, Weights bits and Gradient bits respectively

MODEL	I	A	W	G	ERROR
TBP (OURS)	3	2	1	4	50.43%
TBP (OURS)	3	4	1	4	46.7%
DOREFA (ZHOU ET AL.)	1	2	1	6	53.9%
DOREFA (ZHOU ET AL.)	1	4	1	6	51.8%
WRPN (MISHRA ET AL.)	2	1	1	32	48.3%
WRPN (MISHRA ET AL.)	2	2	2	32	55.8%
SINGLE PRECISION -NO QUANTIZTION					
ALEXNET (KRIZHEVSKY)	1	32	32	32	43.5%

C.3 Additional experiments

To shed light on why TBP works, and on what is yet to be solved, we conducted additional set of experiments on the CIFAR-10 dataset with both ResNet-18 and the BNN like typologies. The Back Propagation algorithm includes three phases. The forward phase in which we calculate the activations values for each layer. The Backward-Propagation (BP) phase in which the gradients from layer ℓ pass to layer $\ell - 1$ and the update phase in which the model parameters (*i.e.,* weights) are updated. Both stages require MAC operations as detailed in section 4. In this paper we focused on the BP stage. As oppose to the update stage that can be done in parallel, BP is a sequential stage. However, if the update phase uses full precision MAC operation the hardware need to support it. Moreover compressing the update gradients reduce the communication bandwidth in distributed systems. Thus, quantizing the weights gradients for the updates phase can also reduce power consummation and accelerate the training.

Table 2: Classification test error rates of TBP BNNs trained on CIFAR10

Model	Error Rate
Binarized activations, weights and tern gradients	
TBP BNN, inflated by 3	9.53%
TBP ResNet-18, inflated by 5	10.8%
TBP ResNet-18, inflated by 3	14.21%
TBP ResNet-18 [10]	18.5%
Binarized activations, weights	
BNN [12]	10.15%
Binarized ResNet-18, inflated by 5	10.7%
No binarization (standard results)	
BNN [12]	10.94%
ResNet-18	9.64%

Ternarizing both stages. Ternarizing both stages results with completely MAC free training. However, our results show that without enabling at least 3bit precision for the update stage the model reaches only approximately 80% accuracy. This indicates that the ternarization noise is too high, and thus distorts the update gradients direction. If we stop the gradients ternarization once the accuracy ceases to increase, the convergence continues and the accuracy increases to the same accuracy as TBP. Thus, ternarizing the update stage can be used to accelerate TBP training of BNN networks by first training it with ternarized weights gradients and then, for the last couple of epochs, continue training with full precision weights gradients.

Algorithm 1: Multiple stochastic ternarization sampling algorithm.

```
Require: gradients from previous layer $g_{s_k}$, binarized activation $a^b_{k-1}$ and number of samples $S$.
  for $k = 1$ to $S$ do
    $g^b_{s_k} \leftarrow \mathrm{StcTern}(g_{s_k})$
    $g_{W^b_k} \mathrel{+}= g^b_{s_k} a^b_{k-1}$
  end for
  Return $g_{W^b_k} \mathrel{/}= S$
```

Multiple stochastic ternarization sampling. To alleviate the need for float MAC operation in the update phase we suggest to use gated XNOR operation multiple times, each time with different stochastic sample of the ternarized tensor and average the results. The algorithm is detailed in Algorithm (1) and results are given in Table (3). As expected the accuracy improves with the amount of sampling. To find the number of samples needed for each layer we adopted a similar geometrical approach as suggested by Anderson & Berg [1] and measured the correlation coefficient (R) between

Data set	Error
TBP BNN, 1 sample	20.1%
TBP BNN, 5 samples	13.5%
TBP BNN, 10 samples	13%
TBP BNN, 20 samples	12%
TBP BNN, $R > 0.7$	12.5%
ResNet-18 10 samples	14%
ResNet-18 20 samples	12.7%

Table 3: Classification test error rates of TBP BNNs trained on CIFAR10 with Multiple Stochastic ternarization sampling for the update phase. ResNet-18 and BNN models were inflated by 5 and 3 respectively.

the update gradients received with and without ternarization. Our experiments indicate that more samples are required for the first two convolution layer (12 samples) while the rest of the layers need approximately 6 samples. Using this configuration keeps the correlation coefficient above 0.7 and results with 87.5% accuracy.