TRAINING AND INFERENCE WITH INTEGERS IN DEEP NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Researches on deep neural networks with discrete parameters and their deployments in embedded systems have been active and promising topics. Although previous works have successfully constrained precision in inference, transferring both training and inference processes to low-bitwidth integers has not been demonstrated simultaneously. In this work, we develop a new method termed as "WAGE" to achieve both discrete training and inference, where weights (W), activations (A), gradients (G) and errors (E) among layers are shifted and linearly constrained to low-bitwidth integers. To perform pure discrete dataflow for fixed-point devices, we further replace batch normalization by a constant scaling layer and simplify other components that are arduous for integer implementation. Improved accuracies can be obtained on multiple datasets, indicating a new type of regularization. Empirically, WAGE demonstrate the potential to deploy discrete training on hardware systems such as integer-based deep learning accelerators and neuromorphic chips with comparable accuracy and higher energy efficiency, which is crucial for future AI applications in variable scenarios with transfer and continual learning demands.

1 INTRODUCTION

Recently deep neural networks (DNNs) are being widely used for numerous AI applications (Krizhevsky et al., 2012; Hinton et al., 2012; Silver et al., 2016). Depending on the massive tunable parameters, DNNs are considered to have powerful multi-level feature extraction and representation abilities. However, training DNNs almost needs energy-intensive devices such as GPU and CPU with high precision (float32) processing units and enough memory capacity, which has greatly challenged their extensive applications in portable devices. In addition, a state-of-art network often has far more weights and effective capacity to shatter all training samples (Zhang et al., 2016), which leads to overfitting easily.

As a result, there is much interest in reducing the size of network during inference, as well as dedicated hardware for commercial solutions (Jouppi et al., 2017; Chen et al., 2017; Shi et al., 2015). Due to the accumulation in stochastic gradient descent (SGD) optimization, the precision demand for training is usually higher than inference (Hubara et al., 2016). Therefore, most of existing techniques only focus on the deployment of a well-trained compressed network, while still keeping high precision and computational complexity during training. In this work, we address the problem as how to process both training and inference with low-bitwidth integers, which is essential for implementing DNNs on dedicated hardware. To this end, two fundamental issues are addressed for discretely training DNNs: i) how to quantize all the operands and operations, and ii) how many bits or states are needed for SGD computation and accumulation.

With respect to the issues, we propose a framework termed as "WAGE" that constrains weights (W), activations (A), gradients (G) and errors (E) among all layers to low-bitwidth integers in both training and inference. Firstly, for operands, shift-based linear mapping and orientation-preserved shifting operations are applied to achieve ternary weights, 8 or less bits integers for activations and gradients accumulation. Secondly, for operations, batch normalization is replaced (Ioffe & Szegedy, 2015) by a constant scaling factor. Other techniques for fine-tuning such as SGD optimizer with momentum and L2 regularization are simplified or abandoned with little performance degradation. Considering the full bidirectional propagation, we completely streamline inference into accumulate-compare cycles, as well as training into low-bitwidth integer multiply-accumulate (MAC) cycles with some alignment operations.

We heuristically explore the bitwidth requirement of integers for error computation and gradient accumulation during training, which have rarely been discussed in previous works. Experiment indicates that it is the relative values (orientations) rather than absolute values (orders of magnitude) in error that leads previous layers to converge. Moreover, small values in errors have negligible effects on previous orientations though propagated layer by layer, which are partially discarded in quantization. We leverage these phenomena and use an orientation-preserved shifting operator to constrain errors. As for the gradient accumulation, though weights are quantized to ternary values in inference, a relatively higher bitwidth is indispensable to store and accumulate gradient updates during training.

The proposed framework is evaluated on MNIST, CIFAR10, SVHN, ImageNet datasets. We find that discretization of backpropagation somehow acts as a type of regularization. Comparing to those who only discretize weights and activations at inference time, it has better convergence accuracy and can further alleviate overfitting. To the best of our knowledge, WAGE is the first work to produce pure bidirectional low-precision integer dataflow for DNNs, which can be applied for training and inference in dedicated hardware neatly. We publish the code on GitHub[1].

2 Related Work

We mainly focus on reducing precision of operations and operands in both training and inference. Orthogonal and complementary techniques for reducing complexity like network compression, pruning (Han et al., 2015) and compact architectures (Howard et al., 2017) are impressively efficient but outside the scope this paper.

Weight and activation Courbariaux et al. (2015); Hubara et al. (2016) propose methods to train DNNs with binary weights (BC) and activations (BNN) successively. They add noises to weights and activations as a form of regularization but real-valued gradients are accumulated in real-valued variables, suggesting that high precision accumulation is likely required for SGD optimizer. XNOR-Nets (Rastegari et al., 2016) have a filter-wise scaling factor for weights to improve the performance. Convolutions in XNOR-Nets can be implemented efficiently using XNOR logical units and bit-count operations. However, these factors are calculated simultaneously during training with float32, which generally aggravates the training effort. In TWN (Li et al., 2016) and TTQ (Zhu et al., 2016) two symmetric thresholds are introduced to constrain the weights to be ternary-valued: $\{+1, 0, -1\}$. They claimed a tradeoff between model complexity and expressive ability.

Gradient computation and accumulation DoReFa-Net (Zhou et al., 2016) succeed in quantizing gradients to low-bitwidth floating-point numbers with discrete states in the backward pass. TernGrad (Wen et al., 2017) quantizes gradients to ternary values to reduce the overhead of gradient synchronization in distributed training. Nevertheless, weights in DoReFa-Net and TernGrad are still stored and updated with float32 like previous works. Besides, the quantization of batch normalization and its derivative are ignored. Thus, the overall computation graph for the training process is still with float32 and more complex with external quantization. Generally, it is difficult to apply DoReFa-Net training in an integer-based hardware directly, but it shows potential of exploring high-dimensional discrete spaces with discrete gradient descent directions.

3 WAGE Quantization

The main idea of WAGE quantization is to constrain four operands to low-bitwidth integers: weight $\boldsymbol{W}$ and activation $\boldsymbol{a}$ in inference, error $\boldsymbol{e}$ and gradient $\boldsymbol{g}$ in backpropagation training, see Figure 1. We extend the original definition of errors to multi-layer: error $\boldsymbol{e}$ is exactly the gradient of activation $\boldsymbol{a}$ for the perspective of each convolution or fully-connected layer, while gradient $\boldsymbol{g}$ particularly refers to the gradient accumulation of weight $\boldsymbol{W}$. Consider the i-th layer of a feed-forward network, we have:

$$\boldsymbol{e}^i = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^i},\ \boldsymbol{g}^i = \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^i} \tag{1}$$

[1]https://github.com/boluoweifenda/WAGE

where $\mathcal{L}$ is the loss function. We separate these two terms that are mixed up in most existing schemes. The gradient of weight $\boldsymbol{g}$ and the gradient of activation $\boldsymbol{e}$ will flow to different paths in each layer, which is a fork both in inference and in backward training and generally acts as node of MAC operations.

For the forward propagation in the i-th layer, assuming that weights are stored and accumulated with k_G-bits integers, then numerous works strive for a better quantization function $Q_W(\cdot)$ that maps higher precision weights to their k_W-bits reflections, for example $[-0.9, 0.1, 0.7]$ to $[-1, 0, 1]$. Although weights are accumulated in high precision like float32, deployments of the reflections in dedicated hardware are much more memory efficient after training. Activations are quantized with function $Q_A(\cdot)$ to k_A bits to align the increased bitwidth caused by MACs. Weights and activations are discretized to even binary values in previous works, then MACs degrade into logical and bit-count operations that are extremely efficient (Rastegari et al., 2016).

For the backward propagation in the i-th layer, the gradients of activations and weights are calculated by the derivatives of MACs that are generally considered to be in 16-bit floating-point precision at least. As illustrated in Figure 1, after the MACs between k_A-bits inputs and k_W-bits weights are performed, the bitwidth of outputs will increase to $[k_A + k_W - 1]$ in signed integer representation, the similar broadening happens to errors $\boldsymbol{e}$. In consideration of training with only low-bitwidth integers, we propose additional functions $Q_E(\cdot)$ and $Q_G(\cdot)$ to constrain bitwidth of $\boldsymbol{e}$ and $\boldsymbol{g}$ to k_E bits and k_G bits, respectively. In general, where there is a MAC operation, there are quantization operators named $Q_W(\cdot)$, $Q_A(\cdot)$, $Q_G(\cdot)$ and $Q_E(\cdot)$ in inference and backpropagation.

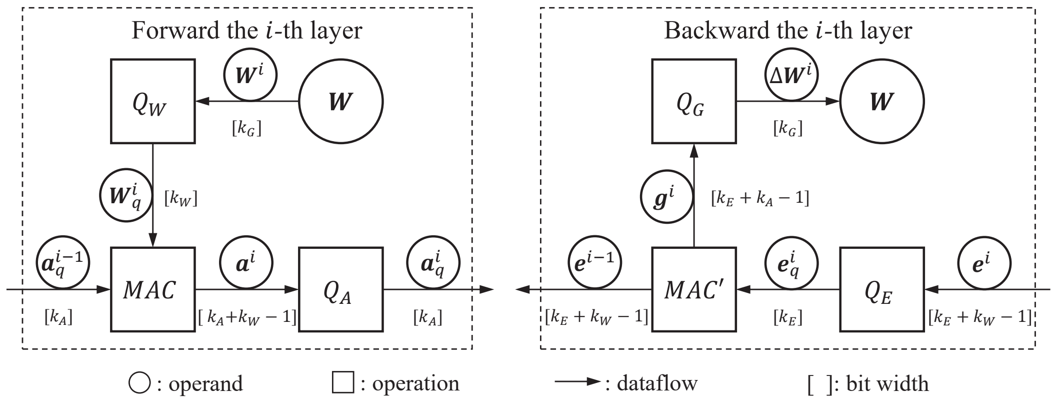

Figure 1: Four operators $Q_W(\cdot)$, $Q_A(\cdot)$, $Q_G(\cdot)$, $Q_E(\cdot)$ added in WAGE computation dataflow to restrain precision, bitwidth of signed integers are below or on the right of arrows, activations are included in MACs for concision.

3.1 Shift-based Linear Mapping and Stochastic Rounding

In WAGE quantization, we adopt a linear mapping with k-bit integers for simplicity, where continuous and unbounded values are discretized with uniform distance σ:

$$\sigma(k) = 2^{1-k}, k \in \mathbb{N}_+ \tag{2}$$

Then the basic quantization function that converts a floating-point number x to its k-bitwidth signed integer representation can be formulated as:

$$Q(x, k) = Clip\left\{ \sigma(k) \cdot round\left[\frac{x}{\sigma(k)}\right], -1 + \sigma(k), 1 - \sigma(k) \right\} \tag{3}$$

where $round$ approximates continuous values to their nearest discrete states. $Clip$ is saturation function that clips unbounded values to $[-1 + \sigma, 1 - \sigma]$, where the negative maximum value -1 is removed to maintain symmetry. For example, $Q(x, 2)$ will quantize $\{-1, 0.2, 0.6\}$ to $\{-0.5, 0, 0.5\}$. Equation 3 is only for simulation in floating-point hardware like GPU, whereas in a fully fixed-point device, quantization is satisfied automatically.

Before applying linear mapping in some operands (e.g., errors), we introduce an additional monolithic scaling factor for shifting values distribution to an appropriate order of magnitude, otherwise values will be all saturated or cleared by Equation 3. The scaling factor is calculated by $Shift$ function and then divided in later steps:

$$Shift(x) = 2^{round(\log_2 x)} \tag{4}$$

Finally, we propose stochastic rounding to substitute small and real-valued updates for gradient accumulation in training. Section 3.3.4 will detail the implementation of operator $Q_G(\cdot)$, where high bitwidth gradients are constrained to k_G-bit integers stochastically by a 16-bit random number generator. Figure 2 summarizes quantization methods used in WAGE.

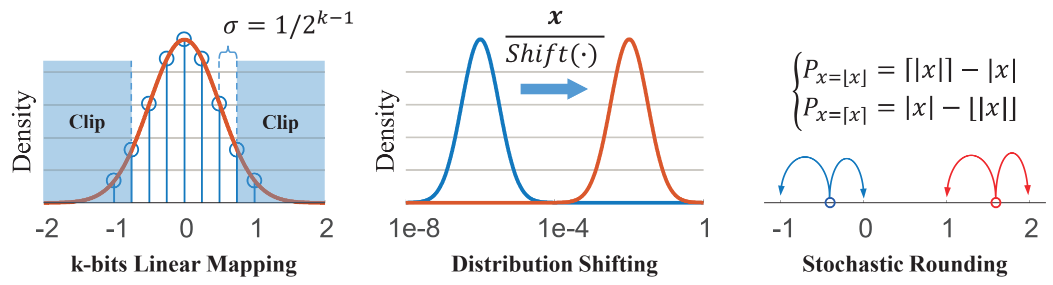
$\sigma = 1/2^{k-1}$
Density
Clip
Clip
-2
-1
0
1
2
k-bits Linear Mapping
$\frac{\boldsymbol{x}}{Shift(\cdot)}$
Density
1e-8
1e-4
1
Distribution Shifting
$\begin{cases} P_{x=\lfloor x \rfloor} = \lceil \lvert x \rvert \rceil - \lvert x \rvert \\ P_{x=\lceil x \rceil} = \lvert x \rvert - \lfloor \lvert x \rvert \rfloor \end{cases}$
-1
0
1
2
Stochastic Rounding

Figure 2: Quantization methods used in WAGE. The notation P, $\boldsymbol{x}$, $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ denotes probability, vector, $floor$ and $ceil$ respectively. $Shift(\cdot)$ refers to Equation 4 with a certain argument.

3.2 Weight Initialization

In previous works, weights are binarized directly by sgn function or ternarized by threshold parameters calculated during training. However, BNN fails to converge when batch normalization is removed because weight values ± 1 are rather big for a typical DNN. Batch normalization efficiently avoids the problem of exploding and vanishing gradients, as well as alleviates the demand for proper initialization. However, normalizing output data for each layer and computing their gradients are quite complex without floating point unit (FPU). Besides, the moving averages of batch outputs occupy external memory. BNN shows a shift-based variation of batch normalization but it is hard to transform all of the elements to the fixed-point representations. As a result, weights should be cautiously initialized in this work where batch normalization is simplified to a constant scaling layer. A modified initialization method based on MSRA (He et al., 2015) can be formulated as:

$$\boldsymbol{W} \sim U(-L, +L), L = max\{\sqrt{6/n_{in}}, L_{min}\}, L_{min} = \beta\sigma \tag{5}$$

where n_{in} is the layer fan-in number, and the original limit $\sqrt{6/n_{in}}$ in MSRA is calculated to keep same variance between inputs and outputs of one layer theoretically. The additional limit L_{min} is a minimum value the uniform distribution U should reach, and β is a constant greater than 1 to create overlaps between minimum step size σ and maximum value L. In case of k_W-bit linear mapping, if weights $\boldsymbol{W}$ are quantized directly with original limits, we will get all-zero tensors when bitwidth k_W is small enough, e.g., 4, or fan-in n_{in} is wide enough, where initialized weights can never reach the minimum step σ presented by fixed-point integers. So L_{min} ensures that weights can go beyond σ and quantized to non-zero values after $Q_W(\cdot)$ when initialized randomly.

3.3 Quantization Details

3.3.1 Weight $Q_W(\cdot)$

The modified initialization in Equation 5 will amplify weights holistically and guarantee their proper distribution, then $\boldsymbol{W}$ is quantized directly with Equation 3:

$$\boldsymbol{W}_q = Q_W(\boldsymbol{W}) = Q(\boldsymbol{W}, k_W) \tag{6}$$

It should be noted that the variance of weights now are scaled compared to the original limit, which will cause exploding of network's outputs. To alleviate the amplification effect, XNOR-Net proposed a filter-wise scaling factor calculated continuously with full precision. In consideration of integer implementation, we introduce a *layer-wise shift-based* scaling factor α to attenuate the amplification effect:

$$\alpha = max\{Shift(L_{min}/L), 1\} \tag{7}$$

where α is a pre-defined constant for each layer determined by the network structure configuration. The modified initialization and attenuation factor α together approximate floating-point weights to their integer representations, except that α takes effect after activations to maintain precision of weights presented by k_W bits integers.

3.3.2 Activation $Q_A(\cdot)$

As stated above, the bitwidth of operands will increase after MACs. Then a typical CNN is usually followed with pooling, normalization and activation function. Average pooling is avoided because mean operations will increase precision demand. Besides, we hypothesize that batch outputs of each hidden layer approximately have zero-mean, then batch normalization degenerates into to a scaling layer where trainable and batch-calculated scaling parameters are replaced by α mentioned in Equation 7. If activations are presented in k_A bits, the overall quantization of activations can be formulated as:

$$\boldsymbol{a}_q = Q_A(\boldsymbol{a}) = Q(\boldsymbol{a}/\alpha, k_A) \tag{8}$$

3.3.3 Error $Q_E(\cdot)$

Errors $\boldsymbol{e}$ are calculated layer by layer using the chain rule during training. Although computation graph of backpropagation is similar to the inference, the inputs are the gradients of $\mathcal{L}$, which are relatively small compared to actual inputs for networks. More importantly, the errors are unbounded and might have significantly larger ranges than that of activations, e.g., $[10^{-9}, 10^{-4}]$. DoReFa-Net first applies an affine transform on $\boldsymbol{e}$ to map them into $[-1, 1]$, and then inverts the transform after quantization. Thus, the quantized $\boldsymbol{e}$ are still presented as float32 numbers with discrete states and mostly small values.

However, experiment uncovers that it is the orientation rather than order of magnitude in errors that leads previous layers to converge, then the inverse transformation after quantization in DoReFa-Net is no longer needed. The orientation-only preservation prompts us to propagate errors with integer thoroughly, where error distribution is firstly scaled into $[-\sqrt{2}, +\sqrt{2}]$ by dividing a shift factor as shown in Figure 2 and then quantized by $Q(\boldsymbol{e}, k_E)$:

$$\boldsymbol{e}_q = Q_E(\boldsymbol{e}) = Q(\boldsymbol{e}/Shift(max\{|\boldsymbol{e}|\}), k_E) \tag{9}$$

where $max|\boldsymbol{e}|$ extracts the layer-wise maximum absolute value among all elements in error $\boldsymbol{e}$, multi-channel for convolution and multi-sample for batch training. The quantization of error will discard large proportion of values smaller than σ, we will discuss the influence on accuracy later.

3.3.4 Gradient $Q_G(\cdot)$

Since we only preserve relative values of error after shifting, the gradient updates $\boldsymbol{g}$ derived from MAC operations between backward errors $\boldsymbol{e}$ and forward activations $\boldsymbol{a}$ are shifted consequently. We first rescale gradients $\boldsymbol{g}$ with another scaling factor and then bring in shift-based learning rate η:

$$\boldsymbol{g}_s = \eta \cdot \boldsymbol{g}/Shift(max\{|\boldsymbol{g}|\}) \tag{10}$$

where η is an integer power of 2. The shifted gradients $\boldsymbol{g}_s$ represent for minimum step numbers and directions when updating weights. If weights are stored with k_G-bit numbers, the minimum step of modification will be ± 1 for integers and $\pm\sigma(k_G)$ for floating-point values, respectively. The implement of learning rate η here is quite different from that in training a vanilla DNN based on float32. In WAGE, there only remains directions for weights to change and the step sizes are integer multiples of minimum step σ. Shifted gradients $\boldsymbol{g}_s$ may get greater than 1 if η is 2 or bigger to accelerate training at the beginning, or smaller than 0.5 during latter half of training when learning rate decay is usually applied. As illustrated in Figure 2, to substitute accumulation of small gradients in latter case, we separate $\boldsymbol{g}_s$ into integer part and decimal part, then use a 16-bit random number generator to constrain high bitwidth $\boldsymbol{g}_s$ to k_G-bit integers stochastically:

$$\Delta \boldsymbol{W} = Q_G(\boldsymbol{g}) = \sigma(k_G) \cdot sgn(\boldsymbol{g}_s) \cdot \Big\{ \lfloor|\boldsymbol{g}_s|\rfloor + Bernoulli(\ |\boldsymbol{g}_s| - \lfloor|\boldsymbol{g}_s|\rfloor\ ) \Big\} \tag{11}$$

where $Bernoulli$ (Zhou et al., 2016) stochastically samples decimal part to either 0 or 1. With proper setting of k_G, quantization of gradients will restrict the minimum step size to update, which may avoid local minimum and overfitting. Furthermore, the gradients will be ternary values when η is not greater than 1, which reduces communication cost for distributed training (Wen et al., 2017). At last, weights $\boldsymbol{W}$ might exceed the range $[-1+\sigma, 1-\sigma]$ presented by k_G-bit integers after updating with discrete increments $\Delta\boldsymbol{W}$. So $Clip$ function is indispensable to saturate and make sure there are only $2^{k_G-1}-1$ states for weights accumulation. In case of t-th iteration, we have:

$$\boldsymbol{W}_{t+1} = Clip\left\{\boldsymbol{W}_t - \Delta\boldsymbol{W}_t, -1+\sigma(k_G), 1-\sigma(k_G)\right\} \tag{12}$$

3.4 Miscellaneous

From the above, we have illustrated our quantization methods for weights, activations, gradients and errors. See Algorithm 1 for the detailed computation graph. There remains some issues to specify in an overall training with integers.

Gradient descent optimizer like Momentum, RMSProp and Adam contains at least one copy of gradients updates $\Delta\boldsymbol{W}$ or their moving average, doubling memory consumption for weights during training, which is partially equivalent to use bigger k_G. Since the weight updates $\Delta\boldsymbol{W}$ are quantized to integer multiple of σ and scaled by η, we adopt pure batch SGD without any form of momentum or adaptive learning rate to show the potential of reducing storage demands.

Although L2 regularization works quite well for many large-scale DNNs where overfitting occurs commonly, WAGE removes small values by Equation 3 and introduces randomness in Equation 11, acting as certain types of regularization and can get comparable accuracy in later experiments. Thus, we leave out L2, $dropout$ or other regularization methods.

The $Softmax$ layer and cross-entropy criterion are widely adopted in classification tasks but the calculation of e^x can hardly be applied in low-bitwidth linear mapping occasions. For tasks with small number of categories, we avoid $Softmax$ layer and use mean-square-error but omit mean operation to form a sum-square-error (SSE) criterion since shifted errors will get the same values according to Equation 9.

4 Experiments

In this section, we set W-A-G-E bits to 2-8-8-8 as default for all layers in a CNN or MLP. The bitwidth k_W is 2 for ternary weights, which implies that there are no multiplication during inference. Constant parameter β is 1.5 to make equal probabilities for ternary weights when initialized randomly. Activations and errors should be of the same bitwidth since computation graph of backpropagation is similar to inference and might be applied in the same partition of hardware or memristor array (Sheridan et al., 2017). Although XNOR and DoReFa achieve 1 bit or 2 bits activations, reducing errors to bitwidth 4 or less will dramatically degenerate accuracies in our tests, so bitwidth k_A and k_E are increased to 8 simultaneously. Weights are stored in 8-bit integers during training and ternarized by two constant symmetrical thresholds during inference. Last but not the least, we make no exception for the first or the last layer, whereas previous works mostly compromise to the last layer for fear of severe accuracy drop caused by large variation range of outputs fed into the $Softmax$ function (Tang et al., 2017). We first build the computation graph for a vanilla network, then insert quantization nodes in forward propagation and override gradients in backward propagation for each layer on Tensorflow (Abadi et al., 2016). Our method is evaluated on MNIST, SVHN, CIFAR10 and ILSVRC12 (Russakovsky et al., 2015) and Table 1 shows the comparison results.

4.1 Implement Details

MNIST A variation of LeNet-5 (LeCun et al., 1998) with 32C5-MP2-64C5-MP2-512FC-10SSE is adopted. The input grayscale images are regarded as activations and quantized with Equation 8 where α equals to 1. The learning rate η in WAGE remains as 1 for the whole 100 epochs. We report average accuracy of 10 runs on test set.

[1]BWN is the counterpart of XNOR that only quantizes weights

[2]Only for worker-to-server communication in distributed training

Table 1: Test or validation error rates (%) for previous works and WAGE on multiple datasets. Opt denotes gradient descent optimizer and withM means SGD with momentum, BN represents for batch normalization and 32-bitwidth refer to float32, ImageNet top-k format: top1/top5.

Method	k_W	k_A	k_G	k_E	Opt	BN	MNIST	SVHN	CIFAR10	ImageNet
BC	1	32	32	32	Adam	✓	1.29	2.30	9.90	64.6/39.0
BNN	1	1	32	32	Adam	✓	0.96	2.53	10.15	-
BWN[1]	1	32	32	32	withM	✓	-	-	-	43.2/20.6
XNOR	1	1	32	32	Adam	✓	-	-	-	55.8/30.8
TWN	2	32	32	32	withM	✓	0.65	-	7.44	34.7/13.8
TTQ	2	32	32	32	Adam	✓	-	-	6.44	42.5/20.3
DoReFa	1-8	1-8	32	2-8	Adam	✓	-	2.50	-	47.0/ -
TernGrad[2]	32	32	2	32	Adam	✓	-	-	14.36	42.4/19.5
WAGE	2	8	8-12	8-12	SGD	✗	**0.40**	**1.93**	6.78	53.5/28.5

SVHN & CIFAR10 We use a VGG-like network (Simonyan & Zisserman, 2014) with 2×(128C3)-MP2-2×(256C3)-MP2-2×(512C3)-MP2-1024FC-10SSE. For CIFAR10 dataset, we follow the data augmentation in Lee et al. (2015) for training: 4 pixels are padded on each side, and a 32×32 patch is randomly cropped from the padded image or its horizontal flip. For testing, only single view of the original 32×32 image is evaluated. The model are trained with mini-batch size of 128 and totally 300 epochs. Learning rate η is set to 8 and divided by 8 at epoch 200 and epoch 250. The original images are scaled and biased to the range of $[-1, +1]$ for 8 bits integer activation representation. As for SVHN dataset, we leave out randomly flip augmentation and reduce training epochs to 40 since it is a rather big dataset. The error rate is evaluated in the same way as MNIST.

ImageNet WAGE framework is evaluated on ILSVRC12 dataset with AlexNet (Krizhevsky et al., 2012) model but removes dropout and local response normalization layers. Images are firstly resized to 256×256 then randomly cropped to 224×224 and horizontally flipped, followed by bias subtraction as CIFAR10. For testing, the single center crop in validation set is evaluated. Since ImageNet task is much difficult than CIFAR10 and has 1000 categories, it is hard to converge when applying SSE or hinge loss criterion in WAGE, so we compromise to $Softmax$ and remove $Q_A(\cdot)$ in the last layer. Besides, k_G and k_E are increased to 12 to form 28CC pattern for stable convergence. The model are trained with mini-batch size of 256 and totally 80 epochs. Learning rate η is set to 32 and divided by 8 at epoch 40 and epoch 60.

4.2 Training Curves and Regularization

We further compare WAGE with two variations and a vanilla CNN on CIFAR10. The vanilla CNN has the same VGG-like architecture described above except that none quantization of any operand or operation is applied. We add batch normalization in each layer and $Softmax$ for the last layer, replace SSE with cross-entropy criterion then use a $L2$ weight decay of 1e-4 and momentum of 0.9 for training. The learning rate is set to 0.1 and divided by 10 at epoch 200 and epoch 250. For variations of WAGE, pattern 28ff has no quantization nodes in backpropagation and pattern 28ff-BN has batch normalization. Figure 3 shows the training curves of four counterparts. It can be seen that the 2888 pattern has comparable convergence rate to the vanilla CNN, better accuracy than those who only discretize weights and activations in inference time, though slightly more volatile. The discretization of backpropagation somehow acts as another type of regularization and have significant error rate drop when decreasing learning rate η. Moreover, comparison of 28ff and 28ff-BN curves reveals that the scaling factor α introduced in Equation 7 reduces the demand for batch normalization.

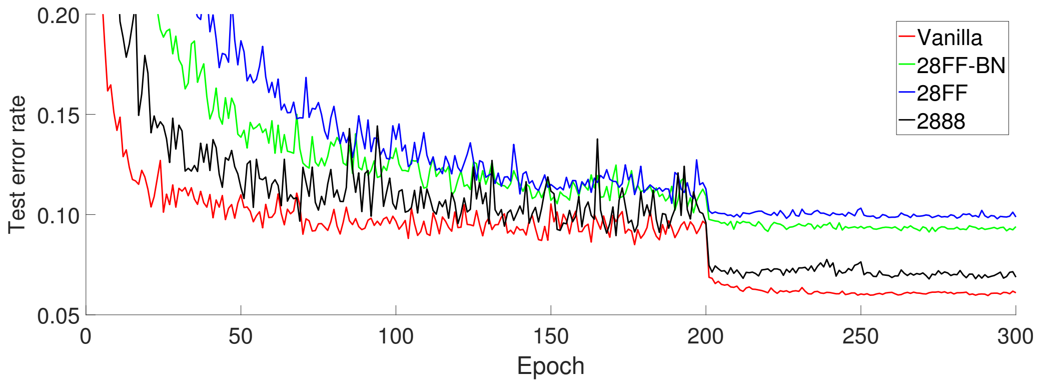

Figure 3: Training curves of WAGE variations and a vanilla CNN on CIFAR10.

4.3 Bitwidth of Errors

The bitwidth k_E is set to 8 as default in previous experiments. To further explore a proper bitwidth and its truncated boundary, we firstly export errors from vanilla CNN for CIFAR10 after 100 training epochs. The histogram of errors in the last convolution layer among 128 mini-batch data is shown in Figure 4. It is obvious that errors approximately obey logarithmic normal distribution where values are relatively small and have significantly large range. When quantized with k_E-bit integers, a proper window function should be chosen to truncate the distribution while retaining the approximate orientations for backpropagation.

Firstly, the upper (right) boundary is immobilized to the maximum absolute value among all elements in errors as described in Equation 9. Then the left boundary will be based on the bitwidth k_E. We conduct a series of experiments for k_E ranging from 4 to 15. The boxplot in Figure 4 indicates that 4-8 bits of errors represented by integers are enough for CIFAR10 classification task. Bitwidth 8 is chosen as default to match the 8-bit image color levels and most operands in the micro control unit (MCU). The histogram of errors in the same layer of WAGE-2888 shows that after being shifted and quantized layer by layer, the distribution of errors reshapes and mostly aggregates into truncated window. Thus, most information of orientations are retained. Besides, the smaller values in errors have negligible effects on previous orientations though accumulated layer by layer, which are partially discarded in quantization.

Since the width of the window has been optimized, we left-shift the window with factor γ to explore its horizontal position. The right boundary can be formulated as $max\{|\mathbf{e}|\}/\gamma$. Table 2 shows the effect of shifting errors: although large values are in the minority, they play critical roles for backpropagation training while the majority with small values actually act as noises.

Table 2: Test error rates (%) on CIFAR10 when left-shift upper boundary with factor γ.

γ	1	2	4	8
error	6.78	7.31	8.08	16.92

4.4 Bitwidth of Gradients

The bitwidth of gradient is set to 8 as default in previous experiments. Although weights are propagated with ternary values in inference and achieve $16\times$ compression rate than float32 weights, they are saved and accumulated in a relatively higher bitwidth (8 bits) for backpropagation training. Therefore, the overall compression rate is only $4\times$. The inconsistent bitwidth between weight updates k_G and their effects in inference k_W provides indispensable buffer space. Otherwise, there might be too many weights changing their ternary values in each iteration, making training very slow and unstable. To further explore a proper bitwidth for gradients, we use WAGE 2-8-8-8 in CIFAR10

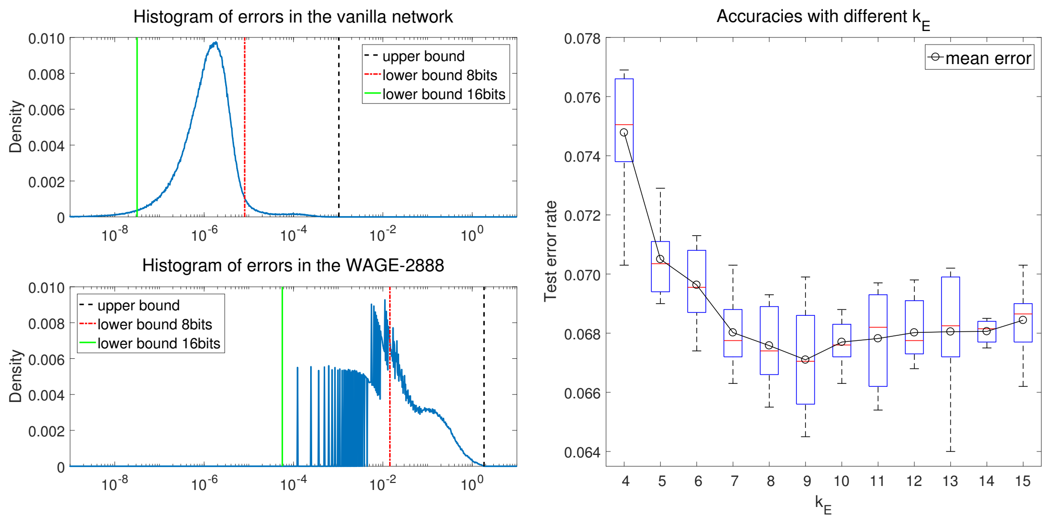

Figure 4: Left are histograms of errors $\boldsymbol{e}$ for same layer in vanilla network and WAGE-2888 network. Upper boundaries are the $max\{|\boldsymbol{e}|\}$ while lower boundaries are determined by the bitwidth k_E. The 10 run accuracies of different k_E are shown on the right.

as baseline and range k_G from 2 to 12, the learning rate η is divided by 2 every time the k_G decreases 1 bit to keep approximately equal weights accumulation in large number of iterations. Results in Table 3 show the effect of k_G and indicate the similar bitwidth requirement as previous experiments on k_E. For ImageNet implementation, we conduct six patterns to show bitwidth requirement: 28CC

Table 3: Test error rates (%) on CIFAR10 with different k_G.

k_G	2	3	4	5	6	7	8	9	10	11	12
error	54.22	51.57	28.22	18.01	11.48	7.61	6.78	6.63	6.43	6.55	6.57

from Table 1, 288C for smaller buffer space, 28Cf for no quantization of errors and 28fC for no quantization of gradients, 28ff for errors and gradients in float32 as unlimited case and its BN counterpart. The accuracy of original Alexnet reproduction is reported as baseline. Learning rate η is set to 4 and divided by 8 in 288C pattern, 0.1 and divided by 10 in 28ff counterparts and vanilla Alexnet. Table 4 indicates higher bits errors and gradients are preferred for ImageNet classification task. Besides, the comparison between pattern 28Cf and 28fC reveal that it might be more important to keep high-resolution orientation k_E than to make more buffer space k_G for gradient accumulation.

To avoid external memory consumption for full-precision hidden weights in the training, Deng et al. (2017) achieved overall 1-bit weights representation in both training and inference. They use a much larger mini-batch size of 1000 and float32 backpropagation dataflow to accumulate more precise weight updates, equally compensating the buffer space in WAGE provided by external bits of k_G. However, large batch size will dramatically increase total training time, counteracting the speed benefits brought by integer arithmetic units. Besides, intermediate variables like feature maps in hidden layer often consume much more memory than weights and linearly correlated with mini-batch size. Therefore, we compromise to bigger inconsistent k_G for better convergence rate, accuracy and smaller mini-batch size.

Table 4: Top-5 error rates (%) on ImageNet with different k_G and k_E.

Pattern	Vanilla	28ff-BN	28ff	28Cf	28fC	28CC	288f	288C
error	19.29	23.37	25.24	26.60	27.47	28.56	31.50	33.95

5 Discussion and Future Work

The goal of this work is to demonstrate potentials of applying training and inference with fixed-point integers in DNNs. We mainly reduce bitwidths of operations and operands for an overall integer dataflow and introduce many techniques to simplify the training process. However, there are still some points not involved in this work but yet to be improved or solved in future algorithm developments and hardware deployments.

MAC Operations: WAGE framework is mainly tested with 2-8-8-8 bitwidth configuration, which means that though there are no multiplications during inference with ternary weights, MAC operations are still needed to calculate $\boldsymbol{g}$ in training. Possible solution is 2-2-8-8 pattern if we do not consider the matching of bitwidths between $\boldsymbol{a}$ and $\boldsymbol{e}$. However, ternary $\boldsymbol{a}$ will dramatically slow down convergence and hurt accuracy since $Q(x, 2)$ has two relatively high thresholds and restrain most outputs of each layer to 0 at the beginning of training, this phenomenon is also observed in our BNN replication.

Non-linear quantization: The linear mapping with uniform distance is adopted in WAGE for its simplicity. However, non-linear quantization method like logarithmic representation might be more efficient because the weights and activations in a trained network naturally have logarithmic normal distributions as shown in Figure 4. Besides, values in logarithmic representation have much larger range in fewer bits than fixed-point representation and are naturally encoded in digital hardware. It is promising to training DNNs with integers encoded with logarithmic representation.

Normalization: Normalization layers like $Softmax$ and batch normalization are avoided or removed in some WAGE demonstrations. We think normalizations are essential for end-to-end multi-channel perception where sensors with different modalities have different input distributions, as well as cross-model features encoding and cognition where information from different branches gather to form higher-level representations. Therefore, a better way to perform discrete normalization is of great interest in further studies.

6 Conclusion

WAGE empowers pure low-bitwidth integer dataflow in DNNs for both training and inference. We introduce a new initialization method and a layer-wise constant scaling factor to replace batch normalization, which is a pain spot for network quantization. Many other components for training are also considered or simplified by alternative solutions. In addition, the bitwidth requirement for error computation and gradient accumulation are explored. Experiments reveal that we can quantized relative values of gradients, as well as discard the majority of small values and their orders of magnitude in backpropagation. Although the accumulation for weights updates are indispensable for stable convergence and final accuracy, there are still some space for compression and memory consumption can be further reduced in training. WAGE achieves state-of-art accuracies on multiple datasets with 2-8-8-8 bitwidth configuration. It is promising for incremental works via fine-tuning, more efficient mapping methods, quantization of batch normalization, etc. Overall, we introduce a framework without floating-point representation and demonstrate the potential to implement both discrete training and inference on integer-based lightweight ASIC or FPGA with on-site learning capability.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pp. 3123–3131, 2015.

Lei Deng, Peng Jiao, Jing Pei, Zhenzhi Wu, and Guoqi Li. Gated xnor networks: Deep neural networks with ternary weights and activations under a unified discretization framework. *arXiv preprint arXiv:1705.09283*, 2017.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems*, pp. 4107–4115, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. *arXiv preprint arXiv:1704.04760*, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial Intelligence and Statistics*, pp. 562–570, 2015.

Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pp. 525–542. Springer, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Patrick M Sheridan, Fuxi Cai, Chao Du, Wen Ma, Zhengya Zhang, and Wei D Lu. Sparse coding with memristor networks. *Nature nanotechnology*, 12(8):784, 2017.

Luping Shi, Jing Pei, Ning Deng, Dong Wang, Lei Deng, Yu Wang, Youhui Zhang, Feng Chen, Mingguo Zhao, Sen Song, et al. Development of a neuromorphic computing system. In *Electron Devices Meeting (IEDM), 2015 IEEE International*, pp. 4–3. IEEE, 2015.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Wei Tang, Gang Hua, and Liang Wang. How to train a compact binary neural network with high accuracy? In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. *arXiv preprint arXiv:1705.07878*, 2017.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

A ALGORITHM

We assume that network structures are defined and initialized with Equation 5. The annotations after pseudo codes are potential corresponding operations for implementations in a fixed-point dataflow.

Algorithm 1 Training an I-layer net with WAGE method on floating-point-based or integer-based device. Weights, activations, gradients and errors are quantized according to Equations 6 - 12.

```
Require: a mini-batch of inputs and targets $(\boldsymbol{a}_q^0, \boldsymbol{a}^*)$ which are quantized to $k_A$ bits integers, shift-
    based $\alpha$ for each layer, learning rate scheduler $\eta$, previous weight $\boldsymbol{W}$ saved in $k_G$ bits.
Ensure: updated weights $\boldsymbol{W}_{t+1}$
    1. Forward propagation:
 1: for $i = 1$ to $I$ do
 2:     $\boldsymbol{W}_q^i \leftarrow Q_W(\boldsymbol{W}^i)$                                #Clip
 3:     $\boldsymbol{a}^i \leftarrow ReLU(\boldsymbol{a}_q^{i-1}\boldsymbol{W}_q^i)$                       #MAC, Clip
 4:     $\boldsymbol{a}_q^i \leftarrow Q_A(\boldsymbol{a}^i)$                                 #Shift, Clip
 5: end for
    2. Back propagation:
    Compute $\boldsymbol{e}^I \leftarrow \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^I}$ knowing $\boldsymbol{a}^I$ and $\boldsymbol{a}^*$           #Substrate
 6: for $i = I$ to $1$ do
 7:     $\boldsymbol{e}_q^i \leftarrow Q_E(\boldsymbol{e}^i)$                                 #Max, Shift, Clip
 8:     $\boldsymbol{e}^{i-1} \leftarrow \boldsymbol{e}_q^i \boldsymbol{W}_q^i$                               #MAC, Clip
 9:     $\boldsymbol{g}^i \leftarrow {\boldsymbol{e}_q^i}^{\mathrm{T}} \boldsymbol{a}_q^{i-1}$                              #MAC, Clip
10:     $\Delta \boldsymbol{W}^i \leftarrow Q_G(\boldsymbol{g}^i)$                             #Max, Shift, Random, Clip
11:     Update and Clip $\boldsymbol{W}^i$ according to Equation 12       #Update, Clip
12: end for
```