COMP6248 Reproducability Challenge Training and Inference with Intergers in Deep Neural Networks

Xiao Xu, Shuo An & Weicheng Pi
30578248, 30035333 & 30142164
Department of Electronics and Computer Science
University of Southampton
{xx2n18,sa2y18,wp1u18}@soton.ac.uk

Abstract

This project aims to reproduce the paper, ”Training and Inference with Integers in Deep Neural Networks”[1] from ICLR 2018. The authors of the target paper proposed a method termed as ”WAGE” to discretise the training and inference to low-bitwidth integers simultaneously. Based on the authors’ code, we implemented the experiments to test the performance of this framework and different bitwidth of the operators and operands. The details of the implementations are recorded in this report.

1 Introduction

Deep neural networks are widely adapted for plenty of applications currently. However, training DNN is not an easy task. Due to the massive amount of the parameters, training it needs enormous computation resources supplied by powerful CPUs and GPUs. Therefore, the neural networks can hardly be implemented on the embedded devices.

To compress the DNNs, the authors of the target paper proposed a framework named ”WAGE” to constrain weights(W), activations(A), gradients(G) and errors(E) among all layers to low-bitwidth integers in both training and inference. By doing so, the size of neural networks is reduced.

This project used the code (Wu et al., 2018) released by the authors as a reference to reproduce the target paper. The program is mostly built based on the Pytorch edition(Yang, 2018) from Github. Some modification and adaptions are applied in our experiments. The code is published on Github[2].

2 Paper Summary

In the target paper, the authors proposed a novel approach called WAGE to reduce the size of the neural networks so that they can be applied to dedicated hardware neatly. The core idea is to quantize the weights (W), activations (A), gradients (G), and errors (E) to low-bitwidth integers simultaneously. Different data sets were used to test the performance of this framework. Several experiments were implemented to find suitable bitwidth for the quantization.

Four operators $Q_w(\cdot)$, $Q_a(\cdot)$, $Q_g(\cdot)$ and $Q_e(\cdot)$ are developed to implement the ”WAGE” framework. These four operators are implemented to quantize the weights, activations, gradients and errors among all layers, respectively. Some basic functions are prepared to build these operators.

The quantization function $Q(x, k)$ is the main function proposed to transfer the float number x into k-bitwidth number. It is used for the simulation on the floating-point hardware like GPU, which can be expressed as:

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

[1]https://openreview.net/pdf?id=HJGXzmspb
[2]https://github.com/xxxxxxxiao/DL_WAGE_Pytorch

where the $round$ function rounds the value to the nearest discrete states and the $Clip$ function bounds the values from $-1+\sigma$ to $1-\sigma$. $\sigma(x)$ function is the linear mapping function to discretize the continuous values, which is written as

$$\sigma(k) = 2^{1-k} \tag{2}$$

To prevent the values from being saturated or cleared, the $shift$ function is applied in some operands(e.g. error). The scaling factor is calculated based on it and divided in the later steps:

$$Shift(x) = 2^{round(log_2 x)} \tag{3}$$

Apart from these functions, stochastic rounding is proposed to substitute the updates for gradient accumulation in training, which is implemented in the $Q_G(\cdot)$ function.

3 Experiments

3.1 WAGE Quantizations

The weight is initialized based on the method MSRA(He et al., 2015), which can be formulated as:

$$W \sim U(-L,+L), L = max\{\sqrt{6/n_{in}}, L_{min}\}, L_{min} = \beta\sigma \tag{4}$$

After the initialization, the $\boldsymbol{W}$ is quantized by the Equation 1 directly. The functions are defined in the module of weight initialization in this project. During the experiments, the constant β is set to 1.5, which is the same as the target paper. The parameter k passed into the $Q_W(\cdot)$ defines the bitwidth of the weights. The quantizer takes effect in the training epochs.

The activation and error are all quantized by Equation 1 by passing the k parameters. A hypothesis that batch outputs of each hidden layer approximately have zero-mean is proposed in the activation quantization. The activation operand $\boldsymbol{a}$ is divided by a scale factor α and then quantized by $Q_A(\cdot)$. Which can be expressed by:

$$\boldsymbol{a}_q = Q_A(\boldsymbol{a}) = Q(\boldsymbol{a}/\alpha, k_A), \alpha = max\{Shift(L_{min}/L), 1\} \tag{5}$$

For simplicity, the scale factor is applied in the weight quantization phase $Q_W(\cdot)$ rather than the $Q_A(\cdot)$ function in the Python program. The authors find that the orientations rather than the orders of magnitude in errors that lead to the convergence(Wu et al., 2018). Therefore, the error distribution is firstly scaled into $[-\sqrt{2}, +\sqrt{2}]$ by dividing shift factor defined by Equation 3 and then quantized.

As for the gradients, the scaling procedure is firstly applied since the errors are already shifted and reserved relative values only. Then the stochastic rounding shifts gradients to obtain the $\triangle\boldsymbol{W}$.

$$\boldsymbol{g}_s = \eta \cdot \boldsymbol{g}/Shift(max\{|\boldsymbol{g}|\}) \tag{6}$$

$$\triangle\boldsymbol{W} = Q_G(\boldsymbol{g}) = \sigma(k_G) \cdot sgn(\boldsymbol{g}_s) \cdot \{\lfloor|\boldsymbol{g}_s|\rfloor + Bernoulli(|\boldsymbol{g}_s| - \lfloor|\boldsymbol{g}_s|\rfloor)\} \tag{7}$$

After that, the updated weight can be derived as:

$$\boldsymbol{W}_{t+1} = Clip\{\boldsymbol{W} - \triangle\boldsymbol{W}_t, -1+\sigma(k_G), 1-\sigma(k_G)\} \tag{8}$$

These functions are all defined in the $WAGE\_initializer$ module in this project. In our experiments, the bitwidths of WAGE are set to 2-8-8-8 as the baseline.

3.2 WAGE Performance

First, we implemented the WAGE models on several datasets like the target paper to test the performance. The bitwidths were all set 2-8-8-8 for WAGE respectively.

For the MNIST data set, a variation of LeNet-5(LeCun et al., 1998) with 32C5-MP2-64C5-MP2-512FC-10SSE was defined, in which the SSE represents sum-square-error criterion. The learning rate remains 1 for the whole 100 epochs. The parameters in the wage.sh file were modified to implement this network.

For SVHN and CIFAR10, a VGG-like network was applied as the paper suggested. Before the training, the images of CIFAR10 were padded and randomly cropped. The network was trained

Table 1: Test error rates(%) comparison between reproduction results and target paper

Datasets	**Reproduction results**	**Results in target paper**
MNIST	0.58	0.40
SVHN	2.96	1.92
CIFAR10	7.20	1.78

with a mini-batch size of 128 for 300 epochs. The learning rate η was changed as the training epochs increased, which was realized by the customed $schedule$ function defined in the train.py file. As for SVHN, flip augmentation was left out and the number of epochs was 40 as the target paper suggested. The error rate was evaluated in the same way as MNIST. Table 1 shows the results.

From the table, it illustrated that the WAGE framework reproduced by our experiments behaved closely to the target paper, except the error rate of CIFAR10 did not achieve the expected value. Compared with other related works which are stated in the target paper, the WAGE shows promising performance on the datasets of MNIST, SVHN and CIFAR10. Since the ImageNet is more massive and complicated than the other datasets, we did not evaluate the WAGE framework on it in this reproduced task.

3.3 Training Curves

Followed the target paper, we implemented the WAGE variations and vanilla CNN on CIFAR10 and trained for 300 epochs. Three training curves, pattern 2-8-8-8, 2-8-f-f and vanilla were plotted in Figure 1. The curves of the vanilla CNN and the 2-8-f-f are not depicted as shown in the target paper and the reason has not been detected yet. Overall, the green curve(2-8-8-8) still illustrates that the 2-8-8-8 pattern converges well and the test error rate drops quickly when reaching 200 epochs since the learning rate is divided by 10 at epoch 200.

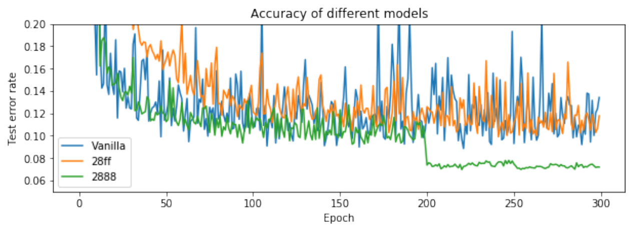

Figure 1: Training curves of WAGE variations and vanilla CNN on CIFAR10

3.4 Bitwidth with Errors

In the target paper, the authors tested k_E with different bitwidth on CIFAR10 and ran 10 times for each k_E with 300 training epochs. The boxplot was created based on the 10 results of each bitwidth.

We tested each k_E once, each training for 300 epochs, due to the long training time and huge computation resource. A line chart is plotted based on the final accuracy of each run in Figure 2. It can be seen that the test errors of our experiments are generally a little higher than what the target paper presents. Although there are some differences between the trend lines of these two plots, both of them illustrate that the test error rate will reduce with the increasing of k_E, whereas it will remain in the same level when k_E is greater than 6. Therefore, a bitwidth of errors around 7 or 8 is applicable and the paper also mentions that a bitwidth of 8 can match the 8-bit image color levels as well as most operands in the micro control unit(MCU).

3.5 Bitwidth with Gradients

Similar to the previous work on the bitwidth of errors, we tuned the value of k_G to find the suitable bitwidth for gradient. The $wlgrad$ value in the wage.sh file was modified. Then we depicted the

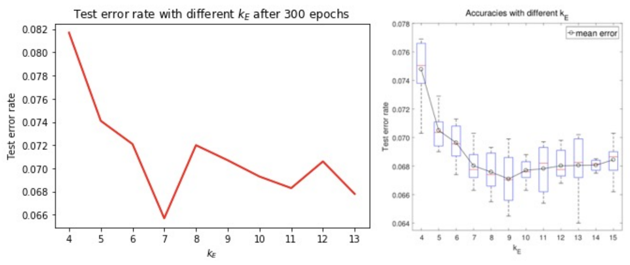

Figure 2: Different bitwidth of k_E and the test accuracy on the CIFAR10 data set. Left is the curve of reproduced results and the right is the original in target paper.

Table 2: Test error rates(%) comparison between reproduction results and target paper

k_G	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Reproduced	57.04	50.25	30.52	19.42	11.22	7.93	7.2	6.91	6.4	6.98	7.24
Target paper	54.22	51.57	28.22	18.01	11.48	7.61	6.78	6.63	6.43	6.55	6.57

outcomes and compared them with the results from the paper, which could be discovered in Table 2. Not surprisingly, the overall error rates generated by our reproducing experiment are higher than the results reported by the paper. However, we could still observe that the error rate decreases as the k_G raises but the decreasing speed slows down when k_G is greater than 7. It is clear that the reproduced outcome is nearly closed to the one presented in the target paper, which indicates that a bitwidth of 8 may be a reasonable choice since the error rate barely changes after k_G gets bigger than 7.

4 Conclusion

This report introduces the concept of WAGE framework and then conducts a series of experiments followed by the target paper to check the reproducibility of this method. By applying WAGE with 2-8-8-8 pattern to LENET-5 and VGG-like architectures and testing on the datasets of MNIST, SVHN and CIFAR10, the experiments deliver promising outcomes compared with the previous works but not as superior as the paper reveals.

The training curves Figure 1 in experiment show a good performance of the WAGE framework. We then further explored proper bitwidths of errors and gradients by testing different values one by one, and the reproduced results basically agreed with that the 2-8-8-8 pattern is the acceptable default setting for the WAGE framework.

References

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.

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

Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HJGXzmspb.

Guandao Yang. Wage pytorch. https://github.com/stevenygd/WAGE.pytorch, 2018.