ICLR 2018 REPRODUCIBILITY CHALLENGE: TRAINING AND INFERENCE WITH INTEGERS IN DEEP NEURAL NETWORKS

Mohammad Hossein Askari Hemmat
Ecole Polytechnique Montréal
m.h.askari.hemmat@gmail.com

Isaac Sultan
McGill University
isaac.sultan@mail.mcgill.ca

Breandan Considine
University of Montréal
breandan.considine@gmail.com

ABSTRACT

We reproduce Wu et al.'s ICLR 2018 submission 'Training And Inference With Integers In Deep Neural Networks'. The proposed 'WAGE' model reduces floating-point precision with only a slight reduction in accuracy. The paper introduces two novel approaches which allow for the use of integer values by quantizing weights, activation, gradients and errors in both training and inference. We reproduce the WAGE model, trained on the CIFAR10 dataset. The methodology demonstrated in this paper has applications for use with Application Specific Integrated Circuit (ASICs). All source code for this re-implementation can be found on our GitHub repository[1] .

1 INTRODUCTION

Reproducibility is often described as one of the main principles of the scientific method. However, confidence in peer-reviewed scientific findings has been shaken by what is deemed a 'reproducibility crisis' Baker (2015). This issue can be seen, too, within the machine learning (ML) field Olorisade et al. (2017). Olorisade et al.'s paper notes that access to datasets and source code are key obstacles to the reproducibility of ML experiments. Our paper aims to reproduce Wu et al's ICLR 2018 submission 'Training And Inference With Integers In Deep Neural Networks', which was published in the ICLR 2018 conference. We show that the performance obtained by its authors can be re-computed - an important step in assessing the overall validity of this paper.

Previous work in quantization has demonstrated reduced precision is an effective method for reducing model size without compromising inference accuracy. However, there is no prior literature that on low-bitwidth integers both for training and inference of a single model. In *Training And Inference With Integers In Deep Neural Networks*Wu et al. (2018) paper, Wu et al. introduce an integer based model, and demonstrate that their framework can be used for both the inference and training phase of a Deep Neural Network. The methodology is named 'WAGE'. The weights *(W)*, activations *(A)*, gradients *(G)* and errors*(E)* are shifted and linearly constrained to low-bitwidth integers. To ensure compatibility with fixed point devices, batch normalization is replaced with a constant scaling layer, with other components simplified. Improved accuracies are demonstrated on multiple datasets.

There is great potential to implement WAGE-like models for training and inference on integer-based lightweight ASIC or Field Programmable Gate Arrays (FPGAs) with on-site learning capabilities. This may dramatically reduce size, power and processing time in comparison with the widely used General Purpose Graphics Processing Units (GPGPUs).

[1] https://github.com/hossein1387/IFT6135/tree/master/Project/src

Reduce Precision Method		bitwidth		Accuracy loss vs. 32-bit float (%)
		Weights	Activations	
Dynamic Fixed Point	**w/o fine-tuning**	8	10	0.4
	w/ fine-tuning	8	8	0.6
Reduce Weight	**BinaryConnect**	1	32 (float)	19.2
	Binary Weight Network (BWN)	1*	32 (float)	0.8
	Ternary Weight Networks (TWN)	2*	32 (float)	3.7
	Trained Ternary Quantization (TTQ)	2*	32 (float)	0.6
Reduce Weight and Activation	**XNOR-Net**	1*	1*	11
	Binarized Neural Networks (BNN)	1	1	29.8
	DoReFa-Net	1*	2*	7.63
	Quantized Neural Networks (QNN)	1	2*	6.5
	HWGQ-Net	1*	2*	5.2
Non-linear Quantization	**LogNet**	5 (conv), 4 (fc)	4	3.2
	Incremental Network Quantization (INQ)	5	32 (float)	-0.2
	Deep Compression	8 (conv), 4 (fc)	16	0
		4 (conv), 2 (fc)	16	2.6

Table 1: Methods to reduce numerical precision for AlexNet. Accuracy is measured for Top-5 error on ImageNet. *Not applied to first and/or last layers. Even a ternary weight network, with only 2-bit weights and 32-bit float for activations, produces a minor 4% reduction in accuracy. Sze et al. (2017)

1.1 Literature Review

Most neural-network computations are composed of an affine transformations followed by the application of a non-linear element-wise function. New specialised hardware platforms are able to take advantage of such computations to accelerate training and inference. For example, Google recently proposed new dedicated hardware named the Tensor Processing Unit (TPU) et al (2017). 24 % of the TPU's die floor plan is comprised of the *Matrix Multiply Unit*, used to compute the product of weights and activations, and the local unified buffer.

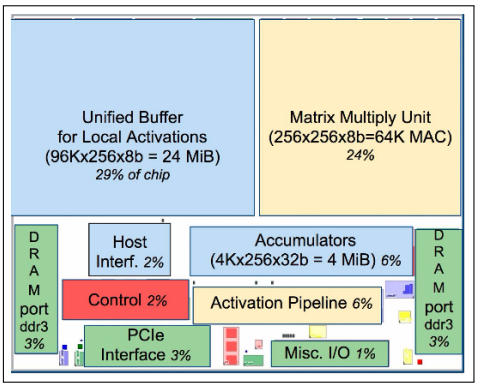

Figure 1: The TPU's floor plan, with the die area used by each block illustrated. The data buffers (blue) use 37% , the compute block (yellow) uses 30% , the I/O (green) uses 10% and the control (red) uses 2% of the die area. The *Matrix Multiply Unit* uses a significant portion of the die et al (2017).

Figure 1 illustrates that the *Matrix Multiply Unit* uses a significant portion of die area. It contains 256×256 Multiply and Accumulators (MACs) that perform 8-bit multiply-and-add on signed and unsigned integers. Google has stated that the success of the TPU is owed to its large *Matrix Multiply Unit*. Use of a large MAC unit comes at a cost of higher power consumption and production cost. It is also worth considering that the TPU is specifically engineered for applications within Google. The lack of formal documentation for the TPU's architecture hinders the ability of researchers to study its performance. Nonetheless, it is clear that a lower gate count results in lower power consumption and smaller die size. This is a reason why traditional 32 bit MAC units are replaced by smaller 8 bits MACs in recent neural network accelerators (as well as Google's TPU). The following table illustrates that even further bit reduction can produce a high level of accuracy. A recent paper on gated XNOR network provides a further improved result Rastegari et al. (2016).

By using precision reduction models, smaller, more power-efficient hardware can be used with minor reduction in accuracy. Previous works have successfully reduced floating-point precision in inference. However, prior to the publication of the paper that we produce, using low-bitwidth integers for both training and inference on the same model had not been demonstrated. In *Binary Neural Net* (BNN) Courbariaux & Bengio (2016), both weights and activations are binarized. The binarization of weights and activations is applied using the $sign$ function. Here, the binarization is applied only in inference. Figure 2 illustrates binarization in the forward path of a **BNN**. We later compare this data path to the WAGE data path.

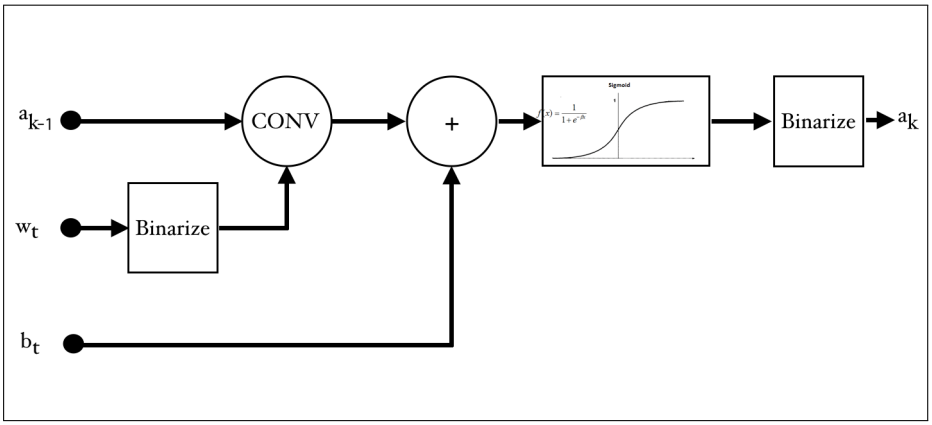

Figure 2: This figure shows the forward computation path for a BNN.The weights are binarized and then convolved with the input. After applying non-linearity binarization is again applied to produce a binarized output.

Figure 2 shows the forward computation path for a BNN. The weights are binarized and then convolved with the input. After applying non-linearity, binarization is again applied to produce a binarized output. In order to do training in BNN, we need to store both full precision values and binarized values. For back-propagation since an non-differentiable $sign$ function is used, we must also use a gradient approximation Courbariaux & Bengio (2016).

Other works build on the BNN idea of quantized parameters. For example, the XNOR-Net Rastegari et al. (2016) introduces a new quantization for weights to improve performance. Convolutions in XNOR-Net can be implemented efficiently using XNOR logical units and bit-count operations. However, these floating-point factors are calculated concurrently during training, which slows training. In TWN Li & Liu (2016) and TTQ Zhu et al. (2016) two symmetric thresholds are introduced to constrain the weights to be ternary-valued: f+1; 0;1g. This is a trade-off between model complexity and expressive ability.

2 METHODOLOGY

We reproduced the most important results for the reproducibility challenge. Wu et al. implemented WAGE with the Tensorflow (TF) deep-learning library for Python Abadi et al.. The authors provided to the implementation of WAGE trained on the CIFAR10 dataset [2]. This was a useful starting point to understand the details of implementation.

We implemented the WAGE, BNN and a simple CNN models in the PyTorch library Paszke et al. (2017). PyTorch has several notable advantages when compared with TF. Most notably, PyTorch uses a dynamic computational graph, which enables fast prototyping and greater ease in debugging. This makes is a suitable choice for research purposes. A disadvantage of TF's static computational graph is discussed in the Discussion.

We designed a VGG like network with custom CNN WAGE layers and Linear Wage layers. The same was done for the BNN. As mentioned in the paper, a full precision output layer was used for

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

WAGE. The BNN, on the other hand, did not require this - which we note as a drawback that the authors should have mentioned in comparison with the BNN methodology.

minibatch-size	num. epochs	learning rate	learning schedule	optimizer
127	300	8	divide by 8 at epoch 200 & epoch 250	SGD

Table 2: The hyper-parameters used when training WAGE

Unlike the BNN, which use both quantized value and high precision values during the training phase, in our WAGE implementation only integer values were used when quantizing weights, activation, gradients and errors. This was done in both forward and backward paths.

Batch normalization was also replaced with a shift and scale operator. As Wu et al. explained, this replacement can only be used under the correct parameter initialization. This improved upon the computationally expensive operation of classical batch normalization is a computationally expensive operation.

The WAGE quantization for weights, activation, gradients and errors are clearly illustrated in Figure 3

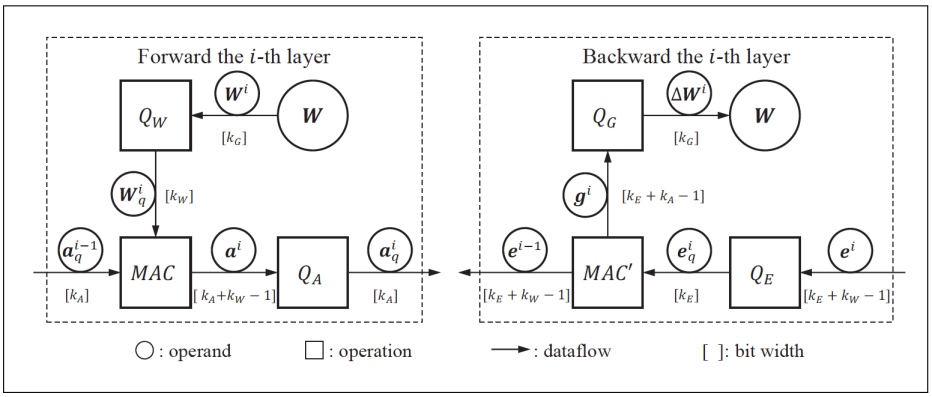

Figure 3: This figure illustrates the quantization method used in WAGE. The figure on the left shows quantization of weight and activation in forward path and figure on the right shows back propagation in which error and gradient are quantized.

Figure 2 and Figure 3 demonstrates the quantization in both paths in WAGE. Since we use a non-differentiable $sign$ function in forward propagation, we used approximate methods to compute gradient in back-propagation. The following method was used to quantize a parameter in WAGE:

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

Where:

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

The $round$ function approximates continuous values to their nearest discrete states. $Clip$ is the saturation function that clips unbounded values to $[1+\sigma, 1-\sigma]$. To prevent $Clip$ function from saturating, we used a scaling operator to shift the values distribution:

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

Small updates were substituted for gradient accumulation in training. The Table 2 summarizes all the quantization used in WAGE:

	Quantization Formula
Weight	$W_q = Q_W(W) = Q(W, k_W)$
Activation	$a_q = Q_A(a) = Q(a/\alpha, k_A)$
Gradient	$\Delta W = Q_G(g) = \sigma(k_G).sgn(g_s)\{\|[g_s]\| + Bernoulli(\|g_s\| - \|[g_s]\|)\}$
Error	$eq = Q_E(e) = Q(e/Shift(max\{\|e\|\}, kE)$

Table 3: Quantization methods used in WAGE. The table summarizes all quantization formulas for weight, activation, gradient and error.

3 RESULTS

Figure 4 shows the learning curves of a vanilla CNN and WAGE 2888 on CIFAR10 dataset. After 300 epochs on the CIFAR10 dataset, a vanilla CNN has approximately 2% error and WAGE 2888 has approximately 7% error. This figure corresponds to Figure 3 of WAGE paper.

Our reproduction verifies the original results. We were unable to find the precision used with the vanilla CNN. For our test, we used a 32 bit floating point to reproduce the learning curve. As shown, compared with vanilla CNN, WAGE 2888 demonstrates promising results with the important advantage of use of less precision.

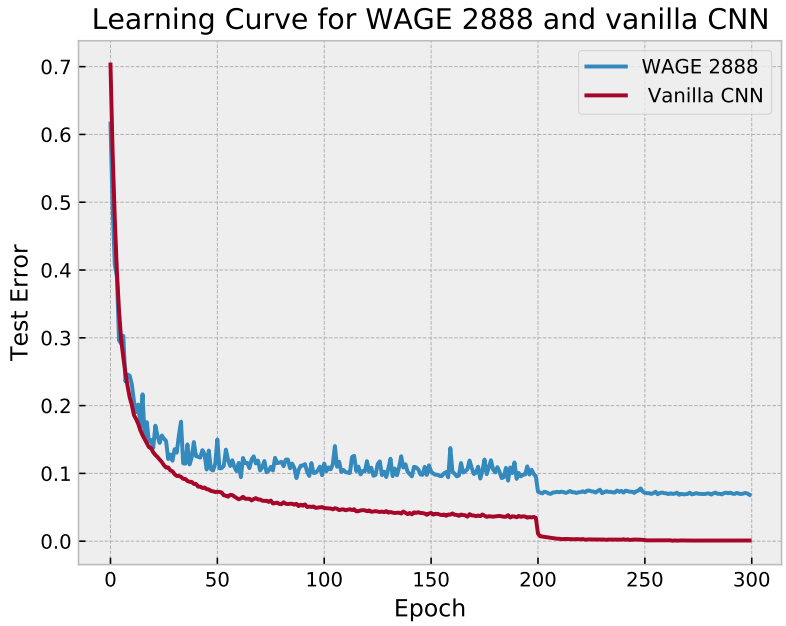

Figure 4: This figure illustrates the learning curves of a vanilla CNN and WAGE with a 2888 configuration. After 300 epochs on the CIFAR10 dataset, a vanilla CNN has approximately 2% error and WAGE 2888 has approximately 7% error.

We believe the Wu et al.'s paper lacked an experiment to determine which quantization parameters in WAGE were most correlated with accuracy. Therefore, we ran 20 different configurations of WAGE, in each reducing only the bit width of one parameter, i.e. for gradients, we varied gradient bit width as follows: 2, 4, 8, C, F. Figure 4 illustrates 20 different tests with different WAGE configuration.

Table 4 shows that the precision parameter for gradient and error has the most effect on the test accuracy. This is illustrated by the top right and bottom left images. Both gradient and errors are

used for back-propagation. Our result show that while weight bits and activation bits can be as low as 2 or 4 bits, gradients and activation required at least 8 bits to produce a satisfactory level of accuracy.

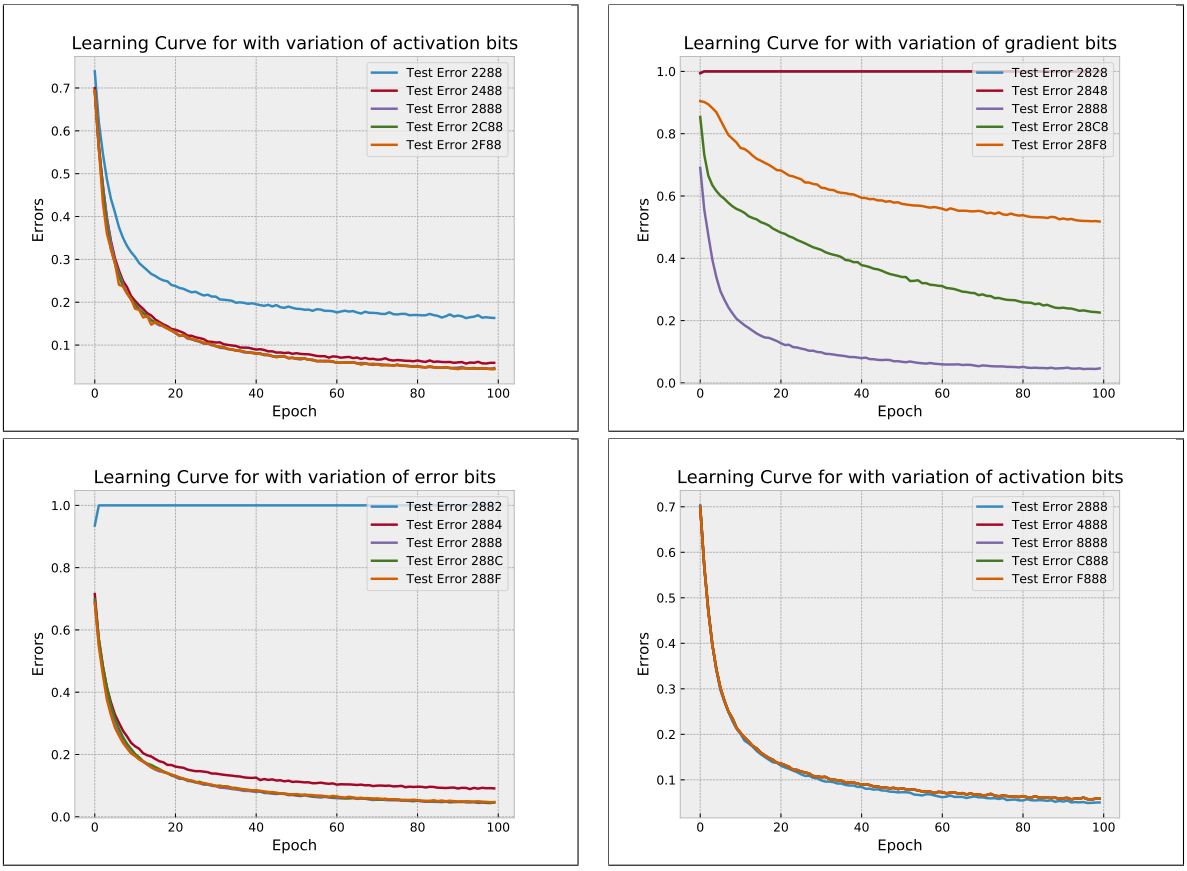

Table 4: This figure illustrates different WAGE setup learning curves. We varied activation, gradient, error and weight bits. Adjusting precision on errors and gradients (back prop) has the most effect on the test accuracy.

Figure 5 depicts test errors in the last convolution layer among 128 mini-batch data. While in the original paper, experiments are conducted for k_E in the range of 4 to 15, we tested fewer models with error bit-width of 2, 4, 6, 10, 12. Our reproduction of the box-plots confirms that with greater than 4 bits of errors represented by integers produce sufficient accuracy on a CIFAR10 classification task. Wu et al. noted that bit-width 8 is chosen as default. This matches the 8-bit image color levels and most operands in MCU.

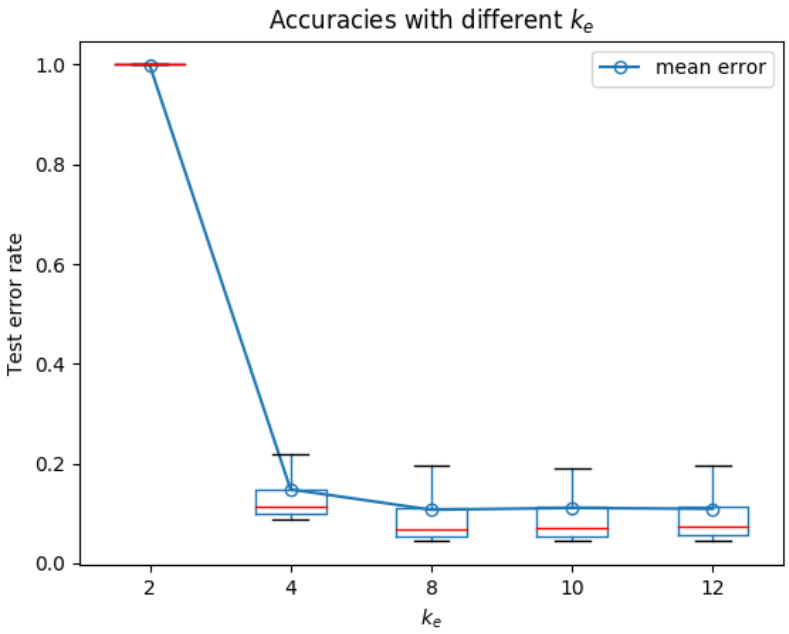

Figure 5: This figure illustrates the test errors of WAGE-2888 with different bit-width k_e

4 DISCUSSION

In the original paper, the authors used the following datasets to evaluate their methodology: MNIST, SVHN, CIFAR10 and ILSVRC12.

Dataset	URL	Size (GB)
MNIST	`yann.lecun.com/exdb/mnist/`	0.07
SVHN	`github.com/thomalm/svhn-multi-digit`	2
CIFAR10	`cs.toronto.edu/~kriz/cifar.html`	0.16
ILSVRC12	`image-net.org/download-images`	139

Table 5: The datasets used in the original paper

All of the datasets are readily available, although ILSVRC12 requires registration on the ImageNet website. We used CIFAR10 in our evaluation Krizhevsky (2009), a dataset of 60000 32x32 colour images in 10 classes, with 6000 images per class. This dataset is commonly used as a baseline for machine learning papers, as does not require as much computational power as larger datasets such as ImageNet, while still displaying natural images unlike MNIST.

Although Wu et al. do not mention how the CIFAR10 dataset was partitioned, we made an assumption that the 50000 training images and 10000 test images split presented in the University of Toronto repository was maintained. The authors also cite the data augmentation procedure followed, following the methodology of Lee et al. Lee et al. (2015). We did not increase the size of the training set in our reproduction, in order not to increase computational overhead.

In terms of processing power, the original paper did not provide their hardware requirements. We used a single NVIDIA Tesla V100, with a 16GB configuration. This is a very powerful GPU, designed specifically for deep learning applications. It boasts a 47X higher inference performance than a CPU unit [3]. An advantage of using such powerful hardware, is that time does not have to be spent parallelizing a model such as WAGE over multiple GPUs.

[3] `http://www.nvidia.com/content/PDF/Volta-Datasheet.pdf`

With the aforementioned hardware, WAGE could be trained on CIFAR10 in one hour. However, it should be noted that GPU used is prohibitively expensive, priced at $8,720.99 USD. More commonly, such as model would be trained on a cloud computation instance such as AWS or GCE. These services charge a lower rental fee for usage of a GPU, but it is likely that training could not be performed as quickly, as only less powerful GPU options are provided.

As previously mentioned in the Methodology, We et al. provided their source code for WAGE on a publicly available GitHub repository. While, the code and its requirements were outlined, some details were missing. In particular the version numbers of the packages used would be helpful. Since TF is implemented with a static computational graph, using the same version is essential for a identical replication.

Moreover, no details were given regarding the random seed used in their experiments. In order to create a fully reproducible TF model, the seed should be set a multiple layers of abstraction: the Python hash-seed, the numpy random seed, the python random seed and the TF random seed should all be manually set. Also, TF should forced to use a single thread, as multiple threads are a potential source of non-reproducible results.

On the other hand, the authors did mention the hyper-parameters used in training the models. We used these hyper-parameters in our training, as shown in Table 2. We were able to communicate with the WAGE authors at the ICLR 2018 conference, where they clarified on the procedure to quantize the forward path. This was implemented with an approximation to the sign function, which in its pure form is not differentiable. Otherwise, the WAGE model provided no notable issues in reimplementation.

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning.

Monya Baker. 1,500 scientists lift the lid on reproducibility. *Nature*, May 2015.

Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016. URL `http://arxiv.org/abs/1602.02830`.

Norman P. Jouppi et al. In-datacenter performance analysis of a tensor processing unit. *CoRR*, abs/1704.04760, 2017. URL `http://arxiv.org/abs/1704.04760`.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

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 and Bin Liu. Ternary weight networks. *CoRR*, abs/1605.04711, 2016. URL `http://arxiv.org/abs/1605.04711`.

Babatunde K Olorisade, Pearl Brereton, and Peter Andras. Reproducibility in machine learning-based studies: An example of text mining. 2017.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016. URL `http://arxiv.org/abs/1603.05279`.

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *CoRR*, abs/1703.09039, 2017. URL `http://arxiv.org/abs/1703.09039`.

Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. *CoRR*, abs/1802.04680, 2018. URL http://arxiv.org/abs/1802.04680.

Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. *CoRR*, abs/1612.01064, 2016. URL http://arxiv.org/abs/1612.01064.