Training and Inference with Integers in Deep Neural Networks - reproducibility

Iban Harlouchet
iban.harlouchet@gmail.com

J.-Ph. Reid
JPh.Reid@gmail.com

Abstract

Wu et al. [2018] have recently developed a new method to discretize both training and inference of deep neural networks (DNNs). This method called 'WAGE' consist of constraining the precision of the weights $\mathbf{W}$, the activations $\mathbf{a}$, the gradients $\mathbf{g}$ and the errors $\mathbf{e}$ among layers thanks to a series of quantized operations. What makes this method interesting compared to others is that it does not require batch normalization for convergence. Despite the lack of precision throughout the layers, this method provides good accuracy on MNIST, CIFAR10 and SVHN datasets, which suggests that WAGE method acts as a type of regularization. The authors demonstrated the potential of WAGE method, as it provides comparable accuracy and higher energy efficiency, which is crucial to future AI applications to small electrical devices. Therefore, it comes with no surprise that this manuscript was accepted to the ICLR2018 as an oral presentation. We show in this report a summary of the paper and its motivation. We focus on the WAGE quantization method used by S. Wu *et al.*, and explore its limits. Further, we reproduce almost all results presented in this paper with comparable accuracy. The source code and the original figures are available on our GitHub [1].

1 Introduction

1.1 Motivation

One fundamental challenge to apply DNNs to portable devices is that it requires energy-intensive hard drives such as GPUs and abundant memory in order to train properly models for specific tasks. To tackle this challenge, it is not advised to reduce the size of training data and the model capacity as it would considerably affect the prediction accuracy. One possibility explored by many was to constrain the precision of the model weights. For example, Courbariaux et al. [2015], Hubara et al. [2016] and Rastegari et al. [2016] introduced a method called BinaryConnect which consists in training a DNN with binary weights (*i.e.*+1 or -1) for the forward and backward propagations. These authors showed that this technique not only increases the effectiveness of the model, but also acts as a regularizer. Along the same lines, Fengfu Li and Liu [2016] proposed ternarized weights (with +1, 0 and -1). This technique has stronger abilities than binary precision counterparts and is more effective. Both of these techniques, however, rely on batch normalization to constrain the variance throughout the neural network.

1.2 Proposed approach

An innovative approach proposed by S. Wu *et al.*(Wu et al. [2018]) consist in limiting the precision of not only the input data, but also each consecutive layer of the network to a specific bitwidth. More precisely, the authors design a model with training and inference with low-bitwidth integers. Two fundamental issues are therefore addressed: i) How to quantize all the operands and operations? ii) What bitwidth is required for training adequately the model?

[1]https://github.com/jphreid/CNN_Quantization

The authors propose a framework termed as ‘WAGE’ which constraints weights (W), activations (A), gradients (G) and errors (E) through all the layers to low-bitwidth integers in both training and inference. Concretely, the authors’ approach consists in reducing the real number line to a small range with a low precision that can be interpreted as a finite set of integers. The authors manipulate the model parameters with shifting, clipping, rounding and scaling operations for convergence, and do not need batch normalization to do so. The authors present in their study an exploration of the possible bitwidth values and their impact on the accuracy.

In this report, we suggest following the footpath of S.Wu *et al.*, *i.e.*

- Implement WAGE model for MNIST, CIFAR10 and SVHN;
- Explore the bitwidth dependence of the test error rate;
- Compare our results with those included in S. Wu *et al.*.

2 Theory

2.1 Elementary operations related to quantization

Quantization operation – The quantization operation converts floating-point input $\mathbf{x}$ to a $k-$bitwidth signed ’integer’, and is defined as

$$Q\left(\mathrm{x}, \mathrm{k}\right) = \mathrm{Clip}\left(\sigma(\mathrm{k}) \cdot \mathrm{round}\left[\frac{\mathrm{x}}{\sigma(\mathrm{k})}\right], -1+\sigma(\mathrm{k}), 1-\sigma(\mathrm{k})\right) \tag{2.1}$$

where $\sigma\left(k\right)$ corresponds to the minimal positive quantity

$$\sigma\left(\mathrm{k}\right) = 2^{1-\mathrm{k}}, \quad k \text{ integer } \geq 2. \tag{2.2}$$

Concretely, $Q\left(\mathbf{x}, \mathrm{k}\right)$ first rounds continuous value x to its nearest discrete state (or multiple of $\sigma(\mathrm{k})$) which is then clipped to the interval $[-1+\sigma(\mathrm{k}), 1-\sigma(\mathrm{k})]$. For instance, the 2-bitwidth quantization, for $\mathbf{x} \in \mathbb{R}$, corresponds to

$$Q\left(x, 2\right) = \begin{cases} -\frac{1}{2} & \text{if } x < -\frac{1}{4} \\ 0 & \text{if } x \in \left[-\frac{1}{4}, \frac{1}{4}\right] \\ \frac{1}{2} & \text{if } x > \frac{1}{4}. \end{cases} \tag{2.3}$$

More generally, k-bitwidth quantization $Q(x,k)$ consists of the $2^k - 1$ states uniformaly distributed in the interval

$$\left[-1 + \frac{1}{2^{k-1}}, 1 - \frac{1}{2^{k-1}}\right]. \tag{2.4}$$

Shifting operation – The authors use an additional transformation which shifts the values of a given distribution. It is built on the function

$$\mathrm{Shift}\left(\mathrm{x}\right) = 2^{\mathrm{round}(\log_2 \mathrm{x})}, \quad \mathrm{x} \in \mathbb{R}^+ \tag{2.5}$$

which associates the value $\mathrm{Shift}\left(\mathrm{x}\right) = 2^{\mathrm{k}}$ to x belonging to the interval $]2^{k-\frac{1}{2}}, 2^{k+\frac{1}{2}}[$, $k \in \mathbb{Z}$. It is then trivial to show that, for any $\gamma = 2^{\eta}, \eta \in \mathbb{Z}$,

$$\frac{\mathrm{x}}{\mathrm{Shift}\left(|\mathrm{x}|/\gamma\right)} \in \begin{cases} \left[\gamma/\sqrt{2}, \gamma\sqrt{2}\right] & \text{if } x > 0 \\ \left[-\gamma\sqrt{2}, -\gamma/\sqrt{2}\right] & \text{if } x < 0 \end{cases} \tag{2.6}$$

Will see later that the authors use $x/\mathrm{Shift}\left(|\mathrm{x}|\right)$ as a shifting transformation that ’preserves the orientation rather than orders of magnitude’ in errors and gradients.

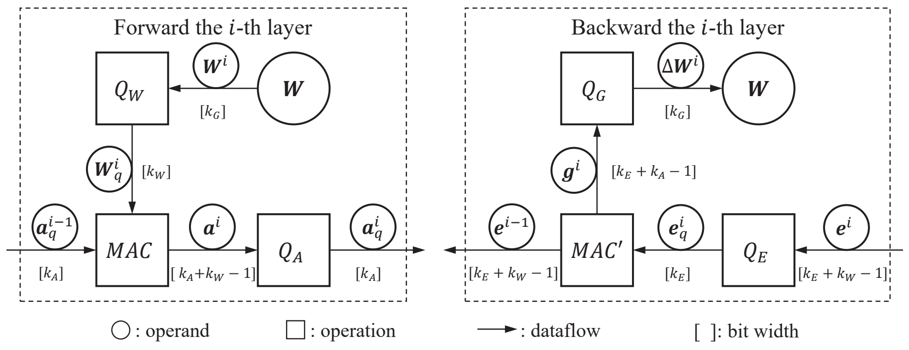

Figure 1: Forward (left) and backward (right) propagation diagrams for the WAGE model. The quantization operators $Q_{\mathbf{W}}(\cdot)$, $Q_{\mathbf{A}}(\cdot)$, $Q_{\mathbf{E}}(\cdot)$ and $Q_{\mathbf{G}}(\cdot)$ are defined in the text. Diagrams taken from S. Wu *et al.*.

Stochastic rounding – The stochastic rounding $\mathcal{S}_r$ of a real number x is defined as

$$\mathcal{S}_r(x) = \operatorname{sgn}(x)\left\{\lfloor|x|\rfloor + \mathcal{B}(|x| - \lfloor|x|\rfloor)\right\} \tag{2.7}$$

where the Bernoulli variable $\mathcal{B}(\cdot)$ samples decimal parts to either 0 or 1 [Zhou et al., 2016], and where $\operatorname{sgn}(x)$ and $\lfloor x \rfloor$ stands for the sign and the floor of x respectively. The function $\mathcal{S}_r(x)$ is defined as

$$\mathcal{S}_r(x) = \begin{cases} \operatorname{sgn}(x)\lfloor|x|\rfloor & \text{with probability } 1 - |x| + \lfloor|x|\rfloor, \\ \operatorname{sgn}(x)\lfloor|x|\rfloor) + 1 & \text{with probability } |x| - \lfloor|x|\rfloor. \end{cases} \tag{2.8}$$

It is straightforward to show that

$$\mathbb{E}\left[\mathcal{S}_r(x)\right] = x. \tag{2.9}$$

2.2 Weights initialization

S. Wu *et al.* implement a modified initialization method based on a previous study [He et al., 2015] where the weights are initialized as

$$\boldsymbol{W} \sim U(-L, +L) \tag{2.10}$$
$$L = \max\left[\sqrt{6/n_{in}}, L_{\min}\right], L_{\min} = \beta\sigma(k)$$

where n_{in} is the layer fan-in number, and the limit $\sqrt{6/n_{in}}$ constraints the weights so that the inputs and outputs of a specific layer keep the same variance [Glorot and Bengio, 2010]. The additional limit $L_{\min} = \beta\sigma(k)$ is a minimum value of the Uniform distribution, which ensures that weights can go beyond the minimal stepsize $\sigma(k)$ and quantized to non-zero values when $\beta > 1$.

2.3 WAGE quantization

The WAGE quantization is summarized in the Fig. 1, and consists of four operations: Weight and Activation quantization in forward propagation, Gradient and Error quantization in backward propagation. Hence the appellation WAGE.

The weights $\mathbf{W}$ represent either the convolutional kernels or the fully-connected layer weights. Activation functions follow, referred to as *multiply-accumulate cycle* (MAC) operations. Both the weights $\mathbf{W}$ and the activations $\mathbf{a}$ are quantized to specific bitwidths. This forward pass is represented in Fig. 1 (left).

The error $\mathbf{e}$ and the gradient $\mathbf{g}$ are the derivative of the loss with respect to the activation functions $\mathbf{a}$ and the weights $\mathbf{W}$ respectively. Specifically, we have

$$\mathbf{e}^i = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^i}, \tag{2.11}$$

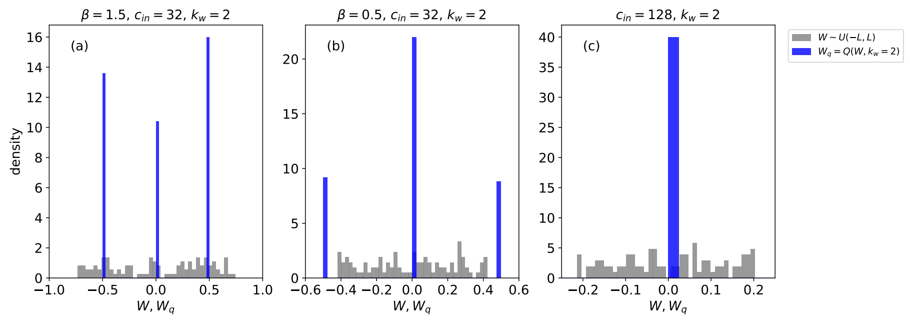

Figure 2: Weights distribution which follows a Uniform distribution, $\mathbf{W} \sim \mathcal{U}(-L, L)$, with L defined in Eq. 2.10. The blue histogram represents the weights quantized distribution, defined in Eq. 2.13. (a) $\beta = 1.5$, $c_{\text{in}} = 32$ and $k_{\text{w}} = 2$. (b) Same as (a), but for $\beta = 0.5$. (c) L_{min} is ignored, $c_{\text{in}} = 128$ and $k_{\text{w}} = 2$. General note: $c_{\text{in}} = 32$ and 128 represent the input images and the output of the first convolutional layer.

$$\mathbf{g}^i = \frac{\partial \mathcal{L}}{\partial \mathbf{W}^i}, \tag{2.12}$$

where $\mathcal{L}$ represents the loss function and i the layer number. The backward pass quantization is represented in Fig. 1 (right).

We will show in the subsequent sections that the key arithmetic operation of deep neural networks (DNNs) is the MAC operation. Thus, discretizing such DNNs boils down to discretization of all MAC operations, *i.e.*to the application of the quantization operators named $Q_{\mathbf{W}}(\cdot)$, $Q_{\mathbf{A}}(\cdot)$, $Q_{\mathbf{E}}(\cdot)$ and $Q_{\mathbf{G}}(\cdot)$ in forward and backward passes.

2.3.1 Weight $Q_W(\cdot)$

The weight values of the ith layer are quantized using Eq. 2.1, *i.e.*

$$\mathbf{W}_q^i \leftarrow Q_W\left(\mathbf{W}^i\right) = Q\left(\mathbf{W}^i, k_{\text{w}}\right) \tag{2.13}$$

We show in Fig. 2 the effect of 'W' quantization on the weight values. We first set $k_{\text{w}} = 2$, $\beta = 1.5$ and, by considering the first convolution layer, $c_{\text{in}} = 32$. Clearly, $\mathbf{W}_q$ quantized distribution overlaps with the Uniform distribution as expected (Fig. 2a). The quantized distribution no longer overlaps if we consider $\beta < 1$, *e.g.* $\beta = 0.5$ (Fig. 2b). Finally, if we ignore the limit L_{min} and if c_{in} is large enough, the quantized distribution collapses to zero, which disables the training of the model (Fig. 2c).

As depicted in Fig. 2(a), $Q_W(\cdot)$ generates a quantized distribution with a variance that scales compared to the original distribution. To alleviate this amplification effect, the authors use a *layer-wise shift based* scaling factor α, proposed by [Rastegari et al., 2016] and defined as

$$\alpha = \max\left[\text{Shift}\left(\text{L}_{\text{min}}/\text{L}\right), 1\right] \tag{2.14}$$

Hence, the modified weights of the i^{th} layer are scaled with a pre-defined factor α^i.

2.3.2 Activation $Q_A(\cdot)$

Although the input images and the kernel weights are quantized to a certain precision, the generated feature maps will have larger bitwidth due to multiply-accumulate cycle (MAC) operation. Specifically, for the WAGE model, the MAC operation has the form of

$$\mathbf{a}^i \leftarrow \text{RELU}\left(\mathbf{a}_{\text{q}}^{\text{i}-1}\mathbf{W}_{\text{q}}^{\text{i}}\right), \tag{2.15}$$

where $\mathbf{W}_q^i$ with k_W bits is computed from Eq. 2.13 and $\mathbf{a}_q^{i-1}$ with k_A bits represents the quantized activation of the previous layer. The output $\mathbf{a}^i$ will have a bitwidth of $k_A + k_W - 1$ in signed integer representation. It will then be reduced to k_A using Eq. 2.1,

$$\mathbf{a}_q^i \leftarrow Q_A\left(\mathbf{a}^i\right) = Q\left(\mathbf{a}^i, k_A\right). \tag{2.16}$$

Further, the authors hypothesize that batch outputs of each hidden layer have approximately zero-mean, and they replace the usual batch-normalization layer by the scaling factor α defined in Eq. 2.14.

2.3.3 Error $Q_E(\cdot)$

Errors $\mathbf{e}^i$ are computed using traditional stochastic gradient descent through each activation layer i (see Eq. 2.11). As the authors claim, experiments suggest that it is the orientations rather than order of magnitude in errors that dictates previous layers to converge. Consequently, they use an orientation-preserved shifting operation to constrain errors between a certain interval (see Eq. 2.6), and the quantized error $\mathbf{e}_q$ they use to update the model parameters is defined to be

$$\mathbf{e}_q^i \leftarrow Q_E\left(\mathbf{e}^i\right) = Q\left(\frac{\mathbf{e}^i}{\text{Shift}\left(\max \mid \mathbf{e}^{\text{i}} \mid\right)}, k_E\right) \tag{2.17}$$

where $\mathbf{e}^i$ is given by Eq. 2.11 and $\max \mid \mathbf{e}^i \mid$ extracts the layer-wise maximum absolute value among all elements in error $\mathbf{e}^i$, multi-channel for convolution and multi-sample for batch training.

From stochastic gradient descent, each error layer $\mathbf{e}^{i-1}$ will depend on the subsequent error layer $\mathbf{e}^i$. In fact, we have that

$$\begin{aligned}\mathbf{e}^{i-1} &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{i-1}} \\ &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}_q^i}\frac{\partial \mathbf{a}_q^i}{\partial \mathbf{a}^{i-1}} && \text{from Eq. 2.11 \& 2.16} \\ &= \mathbf{e}_q^i \mathbf{W}_q^i. \end{aligned} \tag{2.18}$$

As discussed above, the computation of the error $\mathbf{e}^i$ reduces to a MAC operation.

2.3.4 Error $Q_G(\cdot)$

The gradient $\mathbf{g}$ is first shifted to its orientation preserving representation using Eq. 2.6, and then multiplied by the learning rate η which is an integer power of 2,

$$\mathbf{g}_s = \eta \cdot \mathbf{g}/\text{Shift}(\max\{|\mathbf{g}|\}), \tag{2.19}$$

The authors call $\mathbf{g}_s$ the shifted gradient.

If weights are stored in k_G-bit numbers, it is possible to show that their updates, defined as

$$\boldsymbol{W}_{t+1} = Clip\left\{\boldsymbol{W}_t - \Delta \boldsymbol{W}_t, -1 + \sigma(k_G), 1 - \sigma(k_G)\right\}, \tag{2.20}$$

will remain in k_G-bit numbers if the gradient update $\Delta \boldsymbol{W}$ is given by

$$\Delta \boldsymbol{W} = Q_G(\boldsymbol{g}) = \sigma(k_G) \cdot \mathcal{S}_r\left(\mid \mathbf{g}_s \mid\right) \tag{2.21}$$

where $\mathcal{S}_r(\cdot)$ refers to the stochastic rounding (Eq. 2.8).

Again, the computation of the gradient $\mathbf{g}^i$ reduces to a MAC operation, that is

$$\begin{aligned}\mathbf{g}^{i} &= \frac{\partial L}{\partial \mathbf{W}^{i}} \\ &= \frac{\partial L}{\partial \mathbf{a}_q^i}\frac{\partial \mathbf{a}_q^i}{\partial \mathbf{W}^{i}} && \text{from Eq. 2.11 \& 2.16} \\ &= \mathbf{e}_q^{i\intercal} \mathbf{a}_q^{i-1}. \end{aligned} \tag{2.22}$$

2.4 WAGE Algorithm

We have listed in the previous section all the pieces of the WAGE algorithm shown below. The WAGE algorithm is twofolds : the forward and the backward propagations through I layers. Note that the inputs and targets of the network are quantized $k_{\mathbf{A}}$-bit integers.

Algorithm 1 The WAGE forward and backward propagation algorithms.

```
Require: a mini-batch of inputs and targets $(\mathbf{a}_q^0, \mathbf{a}^*)$ which are quantized to $k_A$-bit integers, shift-based $\alpha$ (see Eq. 2.14) for each layer, learning rate scheduler $\eta$ (see Eq. 2.19), previous weight $\mathbf{W}_t$ saved in $k_G$ bits.
Ensure: updated weights $\mathbf{W}_{t+1}$
 1: function WAGE FORWARD PROPAGATION($\mathbf{a}_q^0, \mathbf{a}^*$)
 2:     for $i = 1 : I$ do
 3:         $\mathbf{W}_q^i \leftarrow Q_{\mathbf{W}}\left(\mathbf{W}^i\right)$                    Eq. 2.13
 4:         $\mathbf{a}^i \leftarrow \mathrm{ReLU}\left(\mathbf{a}_q^{i-1}\mathbf{W}_q^i\right)$            Eq. 2.15
 5:         $\mathbf{a}_q^i \leftarrow Q_A\left(\mathbf{a}^i\right)$                       Eq. 2.16
 6:     end for
 7: end function

 8: function WAGE BACK PROPAGATION(($\mathbf{a}^I, \mathbf{a}^*$))
 9:     for $i = I : 1$ do
10:         $\mathbf{e}_q^i \leftarrow Q_E\left(\mathbf{e}^i\right)$                       Eq. 2.17
11:         $\mathbf{e}^{i-1} \leftarrow \mathbf{e}_q^i \mathbf{W}_q^i$                       Eq. 2.18
12:         $\mathbf{g}^i \leftarrow \mathbf{e}_q^{i\intercal} \mathbf{a}_q^{i-1}$                       Eq. 2.22
13:         $\Delta\mathbf{W}^i \leftarrow Q_G\left(\mathbf{g}^i\right)$                    Eq. 2.21
14:         Update and clip $\mathbf{W}^i$                     Eq. 2.20
15:     end for
16: end function
```

3 Experiments

3.1 Source code

The source code of S. Wu *et al.* is available online. It was originally ran on Python 2.7 and TensorFlow(GPU). In order to reproduce the results, we adapted the source code for Python 3.5. The source code available online was to reproduce the CIFAR10 results of the 2888 WAGE model only. We adapted this source code to include the CIFAR10 28ff and vanilla (no quantization), but also the 2888 on MNIST and SVHN datasets. Due to limited calculation time available for this project, we decided not to reproduce the ImageNet results.

3.2 Datasets

We describe below each dataset on which we tested the WAGE variation models.

- **MNIST**
  - $\mathbf{X}$: 28×28 greyscale images normalized in $[0, 1]$
  - $\mathbf{y}$: one-hot coding of lenght 10
  - train MNIST : 50000 inputs
  - test MNIST : 10000 inputs
- **CIFAR10**:
  - $\mathbf{X}$: 32×32 RGB images in $[0, 255]$
  - $\mathbf{y}$: one-hot coding of length 10
  - train CIFAR : 50000 inputs
  - test CIFAR : 10000 inputs
- **SVHN**:
  - $\mathbf{X}$: 32×32 RGB images in $[0, 255]$
  - $\mathbf{y}$: in [1, 10] (we encoded it in one-hot vector of length 10).

  - train SVHN : 604388 inputs

    Although the authors don't mention if they combined the train (73257 inputs) and extra (531131 inputs) SVHN datasets for a total of 604388 inputs, we suppose they did it (as it improves the prediction, and for comparison reasons).

  - test SVHN : 26032 inputs

We refer the reader to the `getData.py` in the MNIST, CIFAR10 and SVHN folders on our GitHub.

3.3 Neural network descriptions

We used the following CNN models to conduct our experiments. The model for the MNIST was not included in the original source code.

Convolutional layers below implement "same" convolution. Due to the fact that mean operations increase precision demand, max pooling is preferred to average pooling, with stride and receptive field equal to 2×2.

- **MNIST**
  - conv5-32-maxpool2-relu
  - conv5-64-maxpool2-relu
  - fc512-relu
  - fc10
- **CIFAR10** and **SVHN**: S. Wu *et al.* made a small mistake by referring to the following architecture as VGG7. It's actually a VGG8.
  - conv3-128-relu
  - conv3-128-maxpool2-relu
  - conv3-256-relu
  - conv3-256-maxpool2-relu
  - conv3-512-relu
  - conv3-512-maxpool2-relu
  - fc1024-relu
  - fc10

We refer the reader to the `NN.py` in the MNIST, CIFAR10 and SVHN folders on our GitHub.

The authors use the sum of squared errors SSE loss function for the WAGE variations (2888 and 28ff), and avoid the Softmax and the mean operations, since both alter the uniform length $\sigma(k)$ (see Eq. 2.2). Also, just as inputs are quantized by Q_A, outputs of WAGE neural networks are quantized by Q_E.

For the vanilla models (*i.e.*models without quantization), the authors use the Softmax and the cross entropy loss function. Besides, they add batch-normalization layers after each convolutional layer.

3.4 Hyperparameters

Here is the list of the hyperparameters for the WAGE algorithm:

1. Learning rates η followed a predefined schedule.

   MNIST

   - 2888:
     Epoch 0 to 100 : $\eta = 1$.

   CIFAR

- 2888:
  Epoch 0 to 200 : $\eta = 8$; Epoch 200 to 250 : $\eta = 1$; Epoch 250 to 300 : $\eta = 0.125$.
- Vanilla:
  Epoch 0 to 200 : $\eta = 0.1$; Epoch 200 to 250 : $\eta = 0.01$; Epoch 250 to 300 : $\eta = 0.001$.
- 28ff:
  As discussed (not in great details) in the original paper, we explored several learning rate schedules based on the vanilla model. Our best result was obtained with the vanilla learning rate schedule scaled by 2.

SVHN

- 2888:
  Epoch 0 to 25 : $\eta = 8$; Epoch 25 to 35 : $\eta = 1$; Epoch 35 to 40 : $\eta = 0.125$.

2. Batch size of 128 was used for all the training datasets.
3. Optimizer with Nesterov momentum 0.9 was used for Vanilla and 28ff models, and simplified to vanilla SGD for 2888 model.
4. L2 regularization with parameter 0.0001 was used for Vanilla model and abandoned for 28ff and 2888 models.

We refer the reader to the `Option.py` in the MNIST, CIFAR10 and SVHN folders on our GitHub.

3.5 Results

We show in Fig. 3 the test error rate with epoch of CIFAR10 for the WAGE variations (2888 and 28ff) and the vanilla models. The sharp steps in the test error rates at epochs 200 and 250 are signatures of the learning rate annealing method. In fact, the authors use a piecewise evolution of the learning rate with epoch. See section 3.4 for more details. We show in table 1 our best results for the three models along with those from S. Wu *et al.*. Note that the authors did not detail the learning schedule for the 28ff model. Since we used the same optimizer as that for the vanilla model, we also set the same learning rate schedule, but scaled by a factor 2.

We present in the table 2 the WAGE 2888 model on both MNIST and SVHN datasets, along with those of CIFAR10. The accuracy corresponds to an average over 10 experiments for MNIST and CIFAR10 and 3 experiments for SVHN respectively. The results for the SVHN are presented in annexe, see Fig. 7. We reach high reproducibility for the three models. Note that we had to adapt the source code available online for both MNIST and SVHN, but also to find the optimal hyperparameters to reach best accuracy.

We tested the effect of left-shifting the error $\mathbf{e}$ with a scaling factor γ, by replacing $\max \mid \mathbf{e} \mid$ by $\max \mid \mathbf{e} \mid /\gamma$ in Eq. 2.17. Our results are indicated in table 3 along with those of S. Wu *et al.*. Again, we reach high reproducibility.

We show in the Fig. 4 the error $\mathbf{e}$ density of the last convolutional neural network of the WAGE 2888 and vanilla models. We add the density delimitations defined by the k_E-bit integer and the $Q_E(\cdot)$ quantization.

Finally, we show in Fig 5 the test error rate distribution for the MNIST and CIFAR10 for the 288-k_E model with $k_E = 8$. These distributions are composed of 10 experiments conducted with the very same experiment parameters detailed in section 3.4.

As a general note, we would like to emphasize that we conducted approximately 50 experiments on the three datasets, gathering about 200 hours of experiments with the MILA laboratory's GPUs (hence the MILA name in the tables refering to us).

Table 1: Test error rates (%) on CIFAR10 for the WAGE models (2888 and 28ff) and the vanilla model. The stars ($\star$) indicate digitized test error rates, as the values were not included in the original manuscript. Results from S. Wu *et al.* are taken from table 1 of their manuscript. *et al.* .

	28ff	2888	Vanilla
MILA	8.32	6.86	5.81
S. Wu *et al.*	7.3*	6.78	5.7*

Table 2: Test error rates for the MNIST, SVHN and CIFAR for the WAGE 2888 model. Our test error rates for MNIST and CIFAR10 represent the average over 10 experiments. See Fig. 5 for more information. Results from S. Wu *et al.* are taken from table 1 of their manuscript.

	MNIST	SVHN	CIFAR10
MILA	0.51	1.99	6.86
S. Wu *et al.*	0.4	1.92	6.78

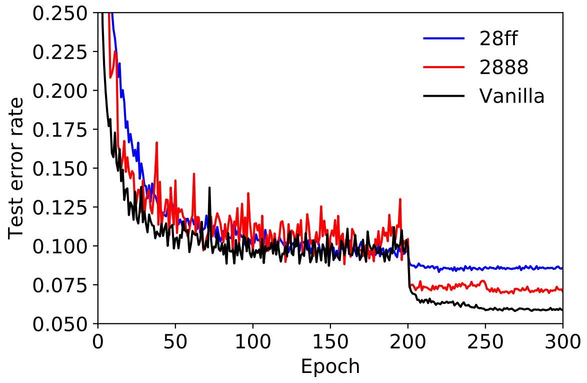

Figure 3: Training curves of WAGE variations and vanilla CNN on CIFAR10. These results replicate those presented in Fig 3 of S. Wu *et al.*.

Table 3: Test error rates (%) on CIFAR10, with left-shift of factor γ. Results from S. Wu *et al.* are taken from table 2 of their manuscript.

γ	1	2	4	8
MILA	6.94	-	-	16.29
S. Wu *et al.*	6.78	7.31	8.08	16.92

Table 4: Test error rates (%) on CIFAR10 with different k_G. In order to reduce the computational time, we only included values for $k_G = 2, 12$ and 8, *i.e.* the limits and the optimal value of k_G respectively. Results from S. Wu *et al.* are taken from table 3 of their manuscript.

k_G	2	3	4	5	6	7	8	9	10	11	12
MILA	54.51	-	-	-	-	-	6.94	-	-	-	6.64
S. Wu *et al.*	54.22	51.57	28.22	18.01	11.48	7.61	6.78	6.63	6.43	6.55	6.57

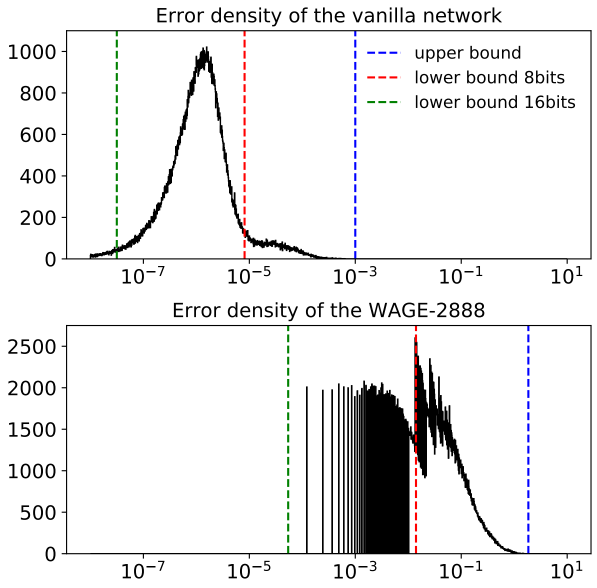

Figure 4: Histograms of errors $\mathbf{e}$ for the vanilla (top) and WAGE model (bottom). The blue, red and green vertical lines correspond to the upper, and lower bound 8bits and 16bits respectively. These results replicate those presented in Fig 4 of S. Wu *et al.*.

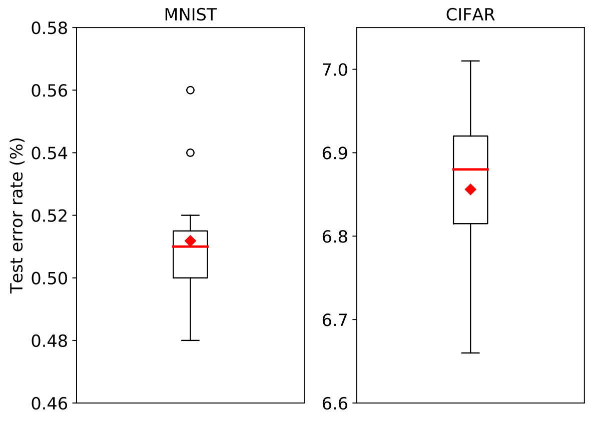

Figure 5: Boxplot for the ten experiments for the WAGE 2888 model. We show the result for the MNIST and CIFAR datasets. The red line and dots correspond the median and average of the datasets respectively. These results replicate partly those presented in Fig 4 of S. Wu *et al.*.

4 Discussion

We reproduced S.Wu *et al.* results for the CIFAR10 WAGE variations and vanilla models, but also on the MNIST and SVHN 2888 models (see tables 1 and 2 and the Fig. 3). We attribute the variation between our results and those reported in the original manuscript to potential difference in initializations of batches and weights. As expected, the accuracy for the vanilla model surpasses that of the WAGE variations. However, considering the differences between the two models, *i.e.* the quantized operations to reduce the precision of the model parameters and the lack of batch normalization, the result obtained from the WAGE models are quite competitive.

In Fig. 3), we observe that model 2888 outperforms model 28ff. One hypotheses to explain this phenomenon is that model 2888 might regularize better than model 28ff. As a matter of fact, the authors of the study claim that WAGE acts as a regularizer, removing small values thanks to Eq. 2.1 and introducing randomness thanks to Eq. 2.21.

We explored a variant of the WAGE 2888 model, by 'left-shifting' $\max \mid \mathbf{e} \mid$ with a factor γ. This consists in substituting $\max \mid \mathbf{e} \mid$ by $\max \mid \mathbf{e} \mid / \gamma$ in $Q_E\left(\mathbf{e}\right)$ (see Eq. 2.17). Intuitively, the 'left-shifting' operation will exclude larger values of errors $\mathbf{e}$. As shown in table 3, excluding such values has significant impact on the model accuracy, despite the fact that these values are in minority.

To test the limits of the WAGE algorithm, we explored different values of k_G-bit integers. As shown in table 4, the test error rate decreases monotonically with the number of k_G-bit values. This is to be expected, but it has a great cost: there is approximately an order of magnitude increase in the energy consumption and the integrated circuit surface area between $k_G = 8$ and 12 (see table 5 of S. Wu *et al.*).

As shown in the Fig. 4, the error $\mathbf{e}$ density after the last convolutional layer follows a logarithmic normal distribution. Unfortunately, the authors don't offer any explanation of why that is.

S. Wu *et al.* conducted a series of experiments for 288-k_E with k_E ranging from 4 to 15 (see Fig. 4. of the original manuscript). Although the minimum test error rate was obtained with $k_E = 9$, the authors chose $k_E = 8$ as default to match the *8-bit image color* levels.

5 Conclusion

In their article, the authors S.Wu *et al.* introduce the WAGE model, where they discretize both training and inference of DNNs. They explore bidwidth requirements for error computation and gradient accumulation, *i.e.* k_E and k_G, and observe that WAGE operates some kind of regularization. They also introduce a new initialization method and a layer-wise constant scaling factor that replaces batch normalization.

Compared to other models that reduce the precision of parameters and inputs, the WAGE framework achieves state of the art accuracy on MNIST, CIFAR10 and SVHN with 2888 configuration, and has comparable test errors to a vanilla CNN on CIFAR10.

The WAGE framework allows to perform pure discrete dataflow for fixed-point devices, thus pointing to potential developments and applications for portable devices.

Although the study of S. Wu *et al.* presents some missing information and some lack of explicit and less cluttered explanations for our full understanding, we didn't find in it any flaws.

References

M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3123–3131. Curran Associates, Inc., 2015.

B. Z. Fengfu Li and B. Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256. PMLR, 2010.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034. IEEE Computer Society, 2015.

I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. *Advances in Neural Information Processing Systems*, pages 4107–4115, 2016.

M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, -, pages 525–542. Springer International Publishing, 2016.

S. Wu, G. Li, F. Chen, and L. Shi. Training and inference with integers in deep neural networks. *CoRR*, abs/1802.04680, 2018.

S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. 2016.

A Appendice A : additional results

We show in Fig. 6 the layerwise histograms for the WAGE 2888 model. These results were obtained after 20 epochs and learning rate $\eta = 8$. The $\mathbf{W}$, and $\mathbf{g}$ yaxis are in linear scale, whereas the $\mathbf{a}$ and $\mathbf{e}$ are in logscale. These results replicate those presented in Fig 5 of S. Wu *et al.*. The reader should note the difference in the x-scale for the $\mathbf{e}$ compared to the original results. It is difficult to explain this difference as the authors do not mention in their manuscript in which epoch they recorded the histograms.

Finally, we show in Fig. 7 the training curves for SVHN and MNIST datasets. These results are not explicitly shown in the original manuscript.

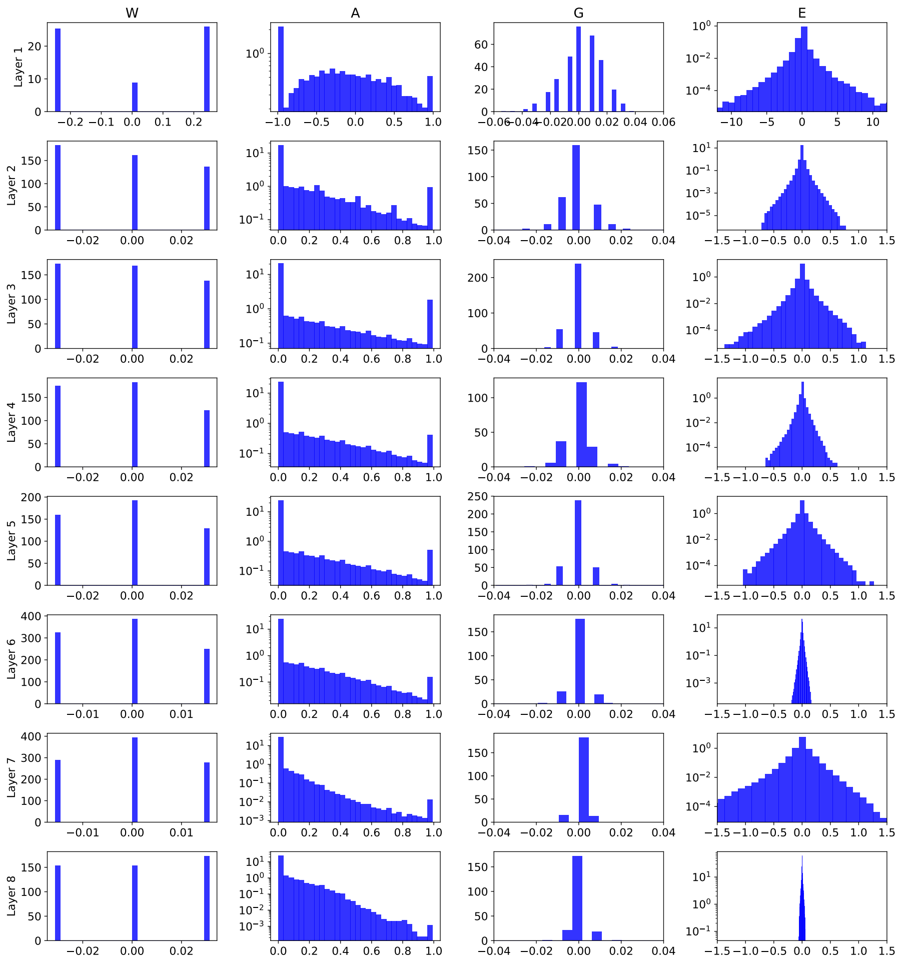

Figure 6: Layerwise histograms of the WAGE CNN model (2888).

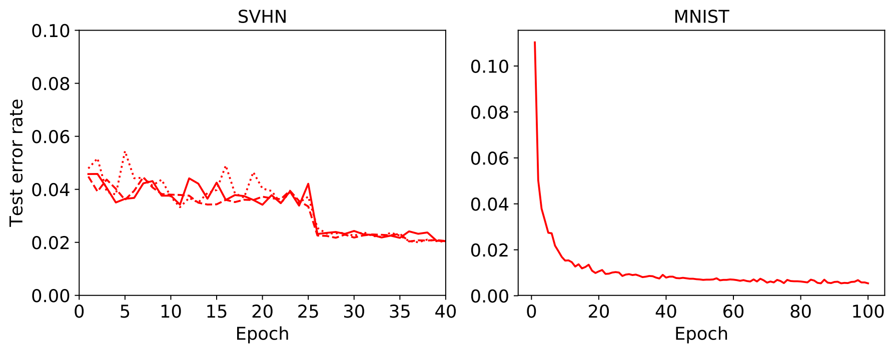

Figure 7: Training curves for the SVHN and MNIST datasets for the 2888 model. The best accuracy is 1.98% for SVHN and 0.51% for MNIST. Learning rate schedules are (SVHN) $\eta = 8$ for epochs $[0, 25[$, $\eta = 1$ for epochs $[25, 35[$ and $\eta = 1/8$ for epochs $[35, 40[$; (MNIST) $\eta = 1$ for epochs $[0, 100[$.