REVISITING BFLOAT16 TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

State-of-the-art generic low-precision training algorithms use a mix of 16-bit and 32-bit precision, creating the folklore that 16-bit precision alone is not enough to maximize model accuracy. As a result, deep learning accelerators are forced to support both 16-bit and 32-bit compute units which is more costly than only using 16-bit units for hardware design. We ask *can we do pure 16-bit training which requires only 16-bit compute units, while still matching the model accuracy attained by 32-bit training*. Towards this end, we study pure 16-bit training algorithms on the widely adopted BFloat16 compute unit. While these units conventionally use *nearest rounding* to cast output to 16-bit precision, we show that nearest rounding for model weight updates can often cancel small updates, which degrades the convergence and model accuracy. Motivated by this, we identify two simple existing techniques, stochastic rounding and Kahan summation, to remedy the model accuracy degradation in pure 16-bit training. We empirically show that these two techniques can enable up to 7% absolute validation accuracy gain in pure 16-bit training. This leads to 0.1% lower to 0.2% higher matching validation accuracy compared to 32-bit precision training across seven deep learning applications.

1 INTRODUCTION

Recently there has been an explosion in the compute resources required for training deep learning models (Shoeybi et al., 2019; Rajbhandari et al., 2019; Real et al., 2019). As a result, there has been broad interest in leveraging low-precision ($<$ 32-bit) training algorithms to reduce the required compute resources (De Sa et al., 2017; Hubara et al., 2017; Gupta et al., 2015). Among these algorithms, mixed-precision training—in which model activations and gradients are stored using 16-bit floats while model weights and optimizer states use 32-bit floats—is commonly used when training generic deep learning models (Micikevicius et al., 2017; Kalamkar et al., 2019). While there is a wide body of literature showing that low-precision training can minimally impact accuracy on specific models (Wang et al., 2018b; De Sa et al., 2015; Zhang et al., 2017), conventional wisdom suggests that at least some 32-bit computation is required as a fail-safe in generic deep learning training. As such, new accelerator architectures for deep learning are forced to support both 32-bit and 16-bit compute units. This is much more costly in terms of area, power, and speed when compared to hardware with only 16-bit compute units (Horowitz, 2014).

In this paper we question if 32-bit compute units are truly needed for new deep learning hardware accelerators. Namely, can we match the model accuracy of 32-bit-precision algorithms while leveraging *only* 16-bit compute units? To answer this question, we study *pure 16-bit training* algorithms, ones which use only 16-bit compute units and which store activations, gradients, model weights, and optimizer states all in 16-bit precision. Specifically, we focus on training with the BFloat16 compute unit which is widely adopted in modern deep learning accelerators (Jouppi et al., 2017; Burgess et al., 2019). Such units take 16-bit inputs, perform computation, and then round the results to a 16-bit output. BFloat16 compute units can provide $3\times$ higher power efficiency, $1.5\times$ lower latency, and $1.5\times$ less chip area than 32-bit units (Horowitz, 2014; Galal et al., 2013). In addition, pure 16-bit training algorithms reduce the memory footprint and bandwidth consumption of model weights and optimizers by $2\times$ compared to mixed precision or 32-bit precision training, especially for large models with billions of weights (Shoeybi et al., 2019; Rajbhandari et al., 2019). Developing reliable pure 16-bit training algorithms will enable hardware designers to realize these advantages.

The simplest approach to pure 16-bit training is to take a 32-bit baseline and "make it low-precision" by replacing all the 32-bit numbers with 16-bit numbers and replacing each 32-bit floating-point op-

eration with its 16-bit analog, using nearest rounding[1] to quantize as necessary: we call this approach the *standard* algorithm. *Unfortunately, we show empirically that standard pure 16-bit training does not match 32-bit training on model accuracy across deep learning models.* For example, the standard pure 16-bit training algorithm one would run on conventional hardware attains 16% and 7% lower training and validation accuracies than a 32-bit baseline. Motivated by this observation, we start by analyzing what factors limit the model accuracy of this standard pure 16-bit algorithm.

In our analysis in Section 3, we derive insights from a simple least-squares regression model, which inspires a clean, minimal set of techniques that allow pure 16-bit training to attain high model accuracy on deep learning models. Using this least-squares regression model, we reveal that nearest rounding of compute unit outputs causes significant convergence degradation and consequent model accuracy loss. More concretely, we show a key theoretical insight hidden in existing work: when running stochastic gradient descent on a least-squares regression model, nearest rounding while updating model weights ignores small updates. This phenomenon significantly degrades the convergence of stochastic gradient descent when model updates become small relative to model weights, which is also what we observe when training deep learning models. In comparison, nearest rounding in the forward and backward pass of backpropagation has a negligible impact on convergence. These insights lead us to consider two simple existing techniques to achieve high-accuracy pure 16-bit training. First, we can use *stochastic rounding* instead of nearest rounding for the model weight updates. Here, the rounded weights become an unbiased estimate of the precise weights without rounding: thus, regardless of the magnitude of updates, the expectation of rounded weights converges at the same speed as the precise weights. Second, we can use the well-known Kahan summation algorithm (Kahan, 1965) to accumulate model updates while still keeping nearest rounding for all operations. This method tracks and compensates weight rounding errors across iterations with auxiliary 16-bit values, which avoids catastrophic cancellation of many small model updates.

Empirically, in Section 4 we first validate that, as suggested by our theory, nearest rounding for model weight updates is the sole bottleneck for convergence and model accuracy on several deep learning models. We then demonstrate that pure 16-bit training using stochastic rounding or Kahan summation on model weight updates can match 32-bit training in model accuracy across a wide range of applications (He et al., 2016; Amodei et al., 2016; Devlin et al., 2018; Naumov et al., 2019). To validate that nearest rounding for model weight updates is the cause of the error, we show that if we store model weights in 32-bit precision without rounding during weight updates, and we keep using 16-bits and nearest rounding for all other operations, then the attained model accuracy matches full 32-bit precision training. Next, we demonstrate that 16-bit training with stochastic rounding for weight updates attains model accuracy matching 32-bit training for five out of seven applications in our study. Note that while it works most of the time, this is not a silver bullet, as using stochastic rounding alone could not fully match 32-bit training on all models. To address this, we show that Kahan summation for model weight updates closes remaining gaps on all the models we consider; this Kahan summation comes with a trade off, as it requires $2\times$ weight memory, but achieves up to 0.2% higher validation accuracy than stochastic rounding. *Our results suggest that deep learning accelerators using only 16-bit compute units are feasible if stochastic rounding and Kahan summation are supported respectively by the hardware and the software stack.*

2 Preliminary

In this section we establish the background and notation for our study and present the preliminary observations that motivate our work. We focus on the case of stochastic gradient descent (SGD), which is the primary workhorse used to train deep learning models. SGD computes gradients from a subset of training samples, and uses them to update the model weights so as to decrease the loss in expectation. In the classic supervised learning setting, let $(\boldsymbol{X}, \boldsymbol{y})$ be a dataset where $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n] \in \mathbb{R}^{n \times d}$ and $\boldsymbol{y} = (y_1, y_2, ..., y_n) \in \mathbb{R}^n$. On this dataset, we use stochastic gradient descent to optimize a loss function $f(\boldsymbol{w}) = 1/n \sum_{i=1}^{n} f_i(\boldsymbol{w}, \boldsymbol{x}_i, y_i)$ defined by the model. At the t-th iteration, we sample an index subset $\sigma(t) \subset \{1, 2, .., n\}$ and compute a sample gradient $\nabla f_{\sigma(t)}(\boldsymbol{w}_t)$ as an unbiased estimate of the full gradient $\nabla f(\boldsymbol{w})$. In deep learning, model training can be described as a compute graph where the compute graph operators such as addition and matrix multiplication are the nodes. For example, the model weight update operator is defined as the subtraction in the operation $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \nabla f_{\sigma(t)}(\boldsymbol{w}_t)$ which updates the model weight $\boldsymbol{w}$.

[1] This nearest rounding is the standard rounding mode for compute unit output commonly supported across hardware platforms (Intel, 2018; Nvidia, 2020).

16-bit Compute Units On modern hardware, numerical operators in the compute graph are supported by fused multiply-and-accumulation (FMAC) compute units. These units work by computing $a \leftarrow a + (x \times y)$, where x and y are input floating point numbers, and a is an accumulator that is part of the FMAC unit. Importantly, for a 16-bit FMAC unit, the accumulator a has higher-than-16-bit precision. This higher precision accumulator is standard in modern hardwares because it ensures the precision of complex operators like matrix multiplication (Henry et al., 2019; Markidis et al., 2018). The result in the accumulator then needs to be *rounded* to 16-bits before it is output from the FMAC unit (e.g. before writing to memory). FMAC units use the same hardware implementation to support all operators from simple additions to computationally intensive convolutions, so this output-rounding step happens for all the operators in a compute graph.

Nearest Rounding FMAC output rounding is widely implemented with *nearest rounding* as the standard mode, due to its efficient support across hardware platforms. "Nearest rounding" means rounding a higher precision floating point number to the closest low-precision representable value. *Given that the add step already uses accurate higher preicision accumulators, this nearest rounding is the primary source of numerical errors for training using 16-bit FMAC units.* In this context of 16 bit FMAC units and nearest rounding, we discuss the following training-precision approaches.

- In *32-bit precision training*, compute graph operators read and write memory using 32-bit format. These operators require 32-bit compute units, which constrains compute and memory efficiency.
- In *mixed precision training*, model weights are stored in 32-bit precision while activations and gradients use 16-bit precision. Thus, new accelerators customized to maximize efficiency for mixed precision training still require 32-bit compute units for operators involving 32-bit weights as the input; this has lower efficiency in power, speed and chip area than only using 16-bit units.
- In *pure 16-bit training*, activation, gradients and model weights are all stored in 16-bit precision. All operators use pure 16-bit input and write out 16-bit output after rounding. Thus, aside from just saving memory, pure 16-bit training can eliminate the requirement for 32-bit compute units. This opens the possibility for highly-efficient accelerators using only 16-bit compute units. In spite of this favorable efficiency, we now show that it can be surprisingly challenging for standard pure 16-bit training to match 32-bit training on model accuracy.

Motivating Observations Although recent works have shown that certain models are robust to numerical error during training (Wang et al., 2018b; De Sa et al., 2015; Zhang et al., 2017), surprisingly, we observe that it is challenging for pure 16-bit training to attain the same accuracy as 32-bit precision training on several state-of-the-art deep learning models. To demonstrate this, we compare 32-bit precision training and standard pure 16-bit training (using nearest rounding for all its operator outputs). For example, Figure 1 illustrates that for a BERT-Base model for natural language inference, the standard pure 16-bit training algorithm demonstrates 16% and 7% lower training and validation accuracy than 32-bit precision training. This gap suggests that nearest rounding is the primary source of numerical error in pure 16-bit algorithms, significantly degrading the convergence and model accuracy. To alleviate this problem, in Section 3, we study how nearest rounding impacts convergence, and we expose insights which lead to simple techniques to improve the model accuracy in pure 16-bit training.

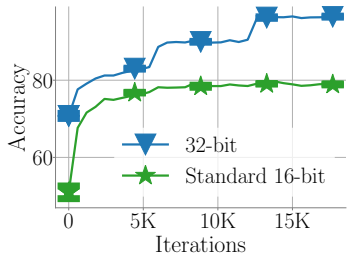

Figure 1: standard pure 16-bit training shows lower training accuracy compared to 32-bit training on a BERT model.

3 Precise Model Weight Updates Are All You Need

To understand how to improve the model accuracy attained by pure 16-bit training, in this section we analyze the impact of nearest rounding, the primary source of numerical error in the standard 16-bit training algorithm. In Section 3.1, we show that when model updates are small relative to the model weights, nearest rounding for model weight updates ignores small updates and consequently impedes convergence of stochastic gradient descent. In contrast, we show that nearest rounding in the forward and backward compute can have a much weaker impact on the convergence throughout training. These insights emphasize the importance of more precise model weight updates for pure 16-bit training. To achieve such precise updates using pure 16-bit training, in Section 3.2 we con-

sider using two simple existing numerical techniques: stochastic rounding and Kahan summation. Although we use a least-squares regression model for its simplicity in exposing insights, we will show in Section 4 that our insights transfer empirically to representative deep learning models.

3.1 Understanding the Impact of Nearest Rounding

In this section, we use a simple least-squares regression model $\frac{1}{2n}\sum_{i=1}^{n}\|\boldsymbol{x}_i^T\boldsymbol{w}-y_i\|^2$ with batch size 1 as a proxy to expose the impact of errors due to nearest rounding. First, we discuss the impact of nearest rounding for model weight updates. Then, we show that nearest rounding for forward and backward compute has comparatively much weaker influence on accuracy. More concretely, we focus on the overparameterized setting (Li et al., 2018; Jacot et al., 2018). In this setting, the model has the capacity to perfectly fit the dataset with $y_i = \boldsymbol{x}_i^T\boldsymbol{w}^*$ where $\boldsymbol{w}^*$ is the minimizer of $f(\boldsymbol{w})$ (**A1**). To ensure convergence with a bounded gradient variance, we assume bounded input data $\|\boldsymbol{x}_i\|^2 \leq L$ (**A2**). We let ϵ denote the machine epsilon of our floating point format, such that if u and v are adjacent representable numbers in our floating point format, $\epsilon|u| \leq |u-v| \leq 2\epsilon|u|$. Under this standard assumption for numerical analysis on floating point numbers (Stoer & Bulirsch, 2013), nearest rounding $\boldsymbol{Q}(\cdot)$ will have a bounded error of $|\boldsymbol{Q}(u)-u| \leq \epsilon|u|$ for any u in range. To simplify the presentation, we ignore overflow and underflow in our analysis here, and disregard factors of ϵ^2 (as is standard in analyses of floating point error).

Nearest Rounding for Model Weight Updates When stochastic gradient descent updates model weights with nearest rounding, the model weights evolve as $\boldsymbol{w}_{t+1} = \boldsymbol{Q}\left(\boldsymbol{w}_t - \alpha\nabla f_{\sigma(t)}(\boldsymbol{w}_t)\right)$. For a weight dimension i, if the model update $\left[\alpha\nabla f_{\sigma(t)}(\boldsymbol{w}_t)\right]_i$ is smaller than half of the difference between $[\boldsymbol{w}_t]_i$ and its neighboring representable value in a certain precision format, nearest rounding cancels this model update. This often emerges in the mid-to-late training stage when the magnitude of gradient becomes small or learning rate decays small. Formally, we show that nearest rounding can cancel updates across all dimensions for a least-squares regression model when approaching the optimal weights in Theorem 1; we defer the proof to Appendix A.

Theorem 1. *Consider running one step of SGD on a least-squares regression model under assumptions **A1** and **A2**. The model weight update will be entirely canceled by nearest rounding if*

$$\|\boldsymbol{w}-\boldsymbol{w}^*\| \leq \frac{\epsilon}{\alpha L+\epsilon}\cdot\min_j\left|w_j^*\right|. \tag{1}$$

where w_j^* denotes the j-th dimension of the optimal solution $\boldsymbol{w}^\star = \arg\min_{\boldsymbol{w}\in\mathbb{R}^d} f(\boldsymbol{w})$. Additionally, if we run multiple steps of SGD using nearest-rounded weight updates and fixed learning rate $\alpha \leq 1/L$, then the distance of the weights $\boldsymbol{w}_t$ at any timestep t from the optimum is bounded by

$$\|\boldsymbol{w}_t-\boldsymbol{w}^*\| \geq \min\left(\frac{\epsilon\left(1-\alpha L\right)}{\alpha L+\epsilon}\cdot\min_j\left|w_j^*\right|, \|\boldsymbol{w}_0-\boldsymbol{w}^*\|\right).$$

Theorem 1 reveals that for the least-squares regression model, nearest rounding cancels the entire model updates when the distance towards the optimal solution $\boldsymbol{w}^*$ is small relative to the magnitude of $\boldsymbol{w}^*$. Thus in the late stage of training, the model weights can halt in a region with radius $\frac{\epsilon}{\alpha L+\epsilon}\min_j\left|w_j^*\right|$ around $\boldsymbol{w}^*$. Our lower bound shows that this region limits the convergence of SGD with nearest rounding for model weight updates on least square regression models. In this bound, the key property is the dependency on the step size: as the step size becomes small, this error lower bound becomes *worse*, which is the opposite of the usual effect of diminishing the step size in SGD. Given that $\boldsymbol{w}^*$ can be arbitrarily far from a zero vector, this property implies a fundamental barrier for stochastic gradient descent to converge extremely close to $\boldsymbol{w}^*$. Because the lower bound on $\|\boldsymbol{w}_0-\boldsymbol{w}^*\|$ is also in the order of $\mathcal{O}(\epsilon)$, this bound is worse for lower precision formats with a large ϵ value. In Section 4, we will empirically show that these insights from the least-squares regression model can also generalize to deep learning models, which explains the convergence and model accuracy degradation due to the small updates cancellation in model weight updates.

Nearest Rounding for Forward and Backward Compute In contrast to the significant convergence degradation imposed by nearest rounding on model weight updates, we show that the nearest rounding in the gradient computation (in the forward and backward passes of backpropagation) can impact convergence minimally. To show this, we consider stochastic gradient descent with nearest rounding only for compute operations which generate activations and gradients. Here to compute the gradient for least-squares regression models, the linear layer passes the rounded activation $a_i =$

$\boldsymbol{Q}\left(\boldsymbol{x}_i^T \boldsymbol{w} - y_i\right)$ to the loss layer. (We see no quantization error within the dot product $\boldsymbol{x}_i^T \boldsymbol{w}$ itself, as all accumulation here is done with the higher-precision accumulator of the FMAC.) In the backward stage, the loss layer feeds the rounded activation gradients $g_{a,i} = \boldsymbol{Q}\left(a_i\right)$ back to the linear layer. The weight gradient is then computed as $\nabla_{\boldsymbol{Q}} f_i(\boldsymbol{w}) := \boldsymbol{Q}\left(g_{a,i}\boldsymbol{x}_i\right) = \boldsymbol{Q}\left(\boldsymbol{Q}\left(\boldsymbol{Q}\left(\boldsymbol{x}_i^T \boldsymbol{w}_t - y_i\right)\right)\boldsymbol{x}_i\right)$. To isolate the impact of nearest rounding for activations and gradients, we do not round model weights in this setting. Formally, we show that SGD with activation and gradient rounding allows for an upper bound on $\|\boldsymbol{w}_t - \boldsymbol{w}^\star\|$ which can be even smaller than the lower bound in Theorem 1.

Theorem 2. *Consider running multiple steps of SGD on a least-squares regression model under assumptions **A1** and **A2**, using nearest rounding for only forward and backward compute, but exact arithmetic for model updates. Then if the step size is small enough that $\alpha \leq 1/L$, the distance of the weights $\boldsymbol{w}_t$ at any timestep t from the optimum will be bounded by*

$$\mathbf{E}\left[\|\boldsymbol{w}_t - \boldsymbol{w}^*\|^2\right] \leq \exp\left(-\alpha\mu t\left(1 - \frac{4\epsilon L}{\mu}\right)\right) \cdot \|\boldsymbol{w}_0 - \boldsymbol{w}^*\|^2,$$

where μ is the smallest eigenvalue of the data covariance matrix $\frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i \boldsymbol{x}_i^T$.

As t can be made arbitrarily large, this bound guarantees us substantially more accurate solutions than the lower bound attained by using nearest rounding only for model weight updates in Theorem 1 allows. This *shows that rounding for the weight updates is the primary source of error*, as even with all other operations quantized, the algorithm is guaranteed to converge closer to the optimum than is even possible with just the weight updates quantized with nearest rounding. Note that the bound in Theorem 2 is able to guarantee arbitrarily accurate solutions because we are ignoring underflow here. In practice, precision would eventually be limited by underflow even in the setting of Theorem 2; however, the underflow threshold for BFloat16 is small enough that this represents a level of error that deep learning applications are generally able to tolerate.

Theory Validation To validate our insights, we compare the impact of nearest rounding for model weight updates against that of nearest rounding in forward and backward compute on a synthetic 10-dimensional least-squares regression problem. Specifically, the input data are sampled from a zero-mean unit-variance normal distribution while the model weight is generated from a uniform distribution in the range of $[0, 100)$. We perturb the label with a zero-mean normal distribution with standard deviation 0.5. As shown in Figure 2, when using a learning rate 0.01 and 16-bit nearest rounding for model weight updates, we observe that the training loss saturates at a magnitudes higher level than stochastic gradient descent without rounding because of updates cancellation. Meanwhile, when using nearest rounding only for forward and backward compute, the loss saturates at a level close to that attained by training without rounding. These observations align with our insights on the relative impact of nearest rounding for model weight updates and for forward and backward compute.

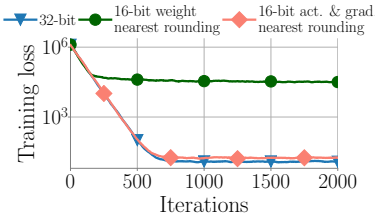

Figure 2: **Theory validation**. On a least square regression model, (smoothed) training losses with 16-bit nearest rounding for weight updates saturate at a higher level than 32-bit training. With only using nearest rounding for forward and backward compute, the losses saturate much closer to 32-bit training.

3.2 High-accuracy Pure 16-bit Training

In Section 3.1, we showed that nearest rounding for model weight updates is the bottleneck for convergence in the standard pure 16-bit training algorithm; this is because it cancels small model updates which degrades the model weight precision. This motivates us to consider two existing techniques, *stochastic rounding and Kahan summation* (Kahan, 1965) for improving weight updates. These techniques have been reliably applied in different numerical domains (Hopkins et al., 2020; Antonana et al., 2017) and can hypothetically enable high-accuracy pure 16-bit training.

Stochastic Rounding Stochastic rounding for floating point numbers has been used in training certain model components (Zhang et al., 2018) and can hypothetically improve the model accuracy of pure 16-bit training for general models. Specifically, let $\mathbb{S}$ be the set of all values representable by a limited precision format: the upper and lower neighboring values for $a \in \mathbb{R}$ are $a_u = \min_{x \geq a, x \in \mathbb{S}} x$ and $a_l = \max_{x \leq a, x \in \mathbb{S}} x$. Stochastic rounding randomly rounds a up to a_u with probability $(a - a_l)/(a_u - a_l)$ and otherwise rounds down to a_l. We consider pure 16-bit training

using stochastic rounding only for the subtraction output in the model update $\boldsymbol{w}_t - \alpha \nabla f_{\sigma(i)}(\boldsymbol{w}_t)$. We keep nearest rounding for all the other compute operations. Here, the rounded model weight is an unbiased estimate of the precise value, so it will still make progress in expectation; this prevents the halting effect from nearest rounding for model weight updates.

Kahan Summation The Kahan summation algorithm uses an auxiliary variable to track numerical errors and to compensate the accumulation results. In the context of 16-bit training, we use a 16-bit auxiliary variable $\boldsymbol{c}_t \in \mathbb{R}^d$ to track the error in model weights. To ensure pure 16-bit data paths, we keep nearest rounding for all operators in compute graphs, including those during Kahan accumulation in Algorithm 1. At iteration t, we first compensate the current model update $\boldsymbol{u}_{t+1}$ by subtracting the previous error $\boldsymbol{c}_t$. We then compute the new model weights by adding the compensated updates $\boldsymbol{y}_t$ to the current weights $\boldsymbol{w}_t$. We reversely subtract previous model weights $\boldsymbol{w}_t$ and the compensated updates $\boldsymbol{y}_t$ to acquire the new numerical error $\boldsymbol{c}_{t+1}$ in the updated weights $\boldsymbol{w}_{t+1}$. For small updates $\boldsymbol{u}_t$ which would cause no change in the weights after nearest rounding, this reverse subtraction records the canceled updates in the weight error $\boldsymbol{c}_{t+1}$. Across iterations, small updates can be accumulated in $\boldsymbol{c}_{t+1}$ until they growing large enough to affect the model weights; this allows convergence to continue when it would otherwise halt due to nearest-rounding effects.

Algorithm 1 SGD updates with Kahan summation

1: Auxiliary value $\boldsymbol{c}_0 \leftarrow 0$ at initialization
2: **Input:** Model updates $-\alpha \nabla f_{\sigma(i)}(\boldsymbol{w}_t)$ at iter. t
3: $\boldsymbol{u}_{t+1} \leftarrow -\alpha \nabla f_{\sigma(i)}(\boldsymbol{w}_t)$
4: $\boldsymbol{y}_{t+1} \leftarrow \boldsymbol{u}_{t+1} - \boldsymbol{c}_t$ ▷ Compensate updates
5: $\boldsymbol{s}_{t+1} \leftarrow \boldsymbol{w}_t + \boldsymbol{y}_{t+1}$ ▷ Accumulate updates
6: $\boldsymbol{c}_{t+1} \leftarrow (\boldsymbol{s}_{t+1} - \boldsymbol{w}_t) - \boldsymbol{y}_{t+1}$ ▷ Measure errors
7: $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{s}_{t+1}$
8: **Return:** $\boldsymbol{w}_{t+1}$

4 Experiments in Deep Learning

Our theory in Section 3 reveals that nearest rounding for model weight updates is the primary source of numerical error during training. This motivates us to suggest using stochastic rounding and Kahan summation in pure 16-bit training for improved model accuracy. To first validate our theory, in this section we start by demonstrating that by ablating nearest rounding for model weight updates from the standard 16-bit training algorithm, the model accuracy gap compared to 32-bit precision training can be closed on deep learning models. Next, we show empirically that with either stochastic rounding or Kahan summation on model weight updates, pure 16-bit training can match the accuracy of 32-bit precision training across representative deep learning applications.

Experiment Setup To validate the accuracy bottleneck, we consider three representative models: ResNet18 (He et al., 2016) on the CIFAR10 image classification (Krizhevsky et al., 2009), BERT-Base (Devlin et al., 2018) on the MNLI natural language inference (Wang et al., 2018a), and DLRM model (Naumov et al., 2019) on the Kaggle Advertising Challenge (CriteoLabs, 2014). To extensively evaluate pure 16-bit training with stochastic rounding and Kahan summation, we additionally consider larger datasets and more applications: ResNet50 on the ImageNet (Deng et al., 2009), BERT-Base on the Wiki103 language model[2] (Merity et al., 2016), DLRM model on the Criteo Terabyte dataset (CriteoLabs, 2018), and Deepspeech2 (Amodei et al., 2016) on the LibriSpeech datasets (Panayotov et al., 2015). As there is no publicly available accelerator with the software and hardware support necessary for our study, we simulate pure 16-bit training using the QPyTorch simulator (Zhang et al., 2019). QPyTorch models PyTorch kernels such as matrix multiplication as compute graph operators, and effectively simulates FMAC units with 32-bit accumulators[3]. For all the training algorithms, we use the same hyperparameters as their original papers or code repositories. We report statistically meaningful results with averaged metrics and standard deviations across runs with 3 random seeds. Additional details are in Appendix C.1.

The Model Accuracy Bottleneck To validate our insights from Section 3, we first show empirically that nearest rounding on the model weights is the primary model accuracy bottleneck on several deep learning models. To do this, we keep the model weights in 32-bit precision and turn off nearest rounding on the model weight updates while keeping nearest rounding for all other operators in the compute graph. Figure 3 shows that the standard pure 16-bit training algorithm with

[2] We subsampled 25% of the Wiki103 and 10% of Librispeech training set because of the training time.

[3] Following the convention in mixed precision training (Micikevicius et al., 2017), our simulator uses fused operators for computationally inexpensive activation and normalization layers.

Table 1: **Model accuracy bottleneck for the standard pure 16-bit training algorithm.** This algorithm shows validation accuracy gap compared to 32-bit training. In an ablation of this algorithm, we use 32-bit model weights and only turn off nearest rounding on model weight updates. This eliminates the gap, suggesting that nearest rounding on weight updates is the accuracy bottleneck.

Model	Dateset (Metric)	32-bit	Standard 16-bit	Standard 16-bit & 32-bit weights
ResNet18	CIFAR10 (acc%)	95.45 ± 0.07	94.23 ± 0.12	95.40 ± 0.05
DLRM	Kaggle (auc%)	80.27 ± 0.01	78.49 ± 0.08	80.26 ± 0.01
BERT-Base	MNLI (acc%)	84.26 ± 0.08	77.53 ± 0.07	84.34 ± 0.04

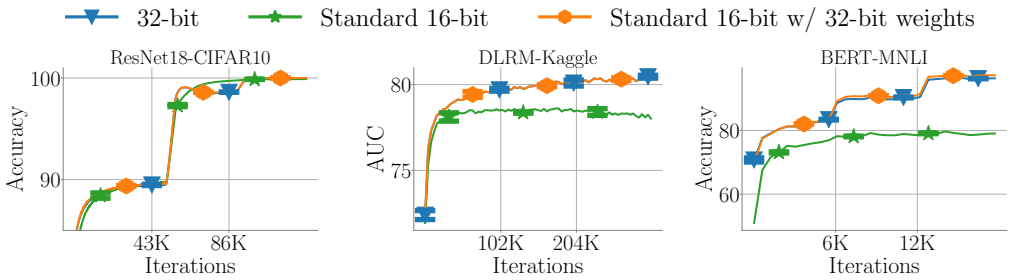

Figure 3: **Training accuracy gap imposed by the standard pure 16-bit training.** The standard algorithm fail to match the training accuracy of 32-bit training, especially in the middle-to-late stage. We close this accuracy gap by ablating nearest rounding for model weight updates from the standard algorithm. This indicates that nearest rounding for model weight update is the accuracy bottleneck.

nearest rounding on all operators has up to 15% training accuracy gap compared to 32-bit training. Although this gap can be small in the early training phase, it grows larger in later stages. In contrast, by ablating nearest rounding on model weight updates, the standard algorithm can fully match the training accuracy attained by full 32-bit training. We notice in Table 1 that this ablation can also close the 1.2% to 6.7% validation accuracy gap when comparing the standard pure 16-bit training to 32-bit training. These observations validate our insights from Section 3.1 and motivate the use of stochastic rounding and Kahan summation on model weight updates.

High-accuracy Pure 16-bit Training Next, we validate empirically that enabling stochastic rounding or Kahan summation for model weight updates allows pure 16-bit training to attain the same model accuracy as 32-bit training. In Table 2, we first show that by using stochastic rounding for model weight updates, pure 16-bit training matches the validation accuracy of 32-bit training with at most 0.1% difference on the CIFAR10, Kaggle, Terabyte, MNLI and Librispeech datasets, a majority of the applications in our experiments. For applications where stochastic rounding still shows a non-negiglable accuracy gap with more than 0.1% discrepancy, we show that Kahan summation for model weight updates can enable pure 16-bit training to match the model accuracy of 32-bit training algorithms. we show that Kahan summation for model weights can boost pure 16-bit training to higher validation accuracy than using stochastic rounding. More concretely, the Kahan summation for model weight updates shows 0.2% higher top-1 accuracy and 0.1% higher AUC respectively for ResNet50 on ImageNet and for recommendation on Terabyte than using stochastic rounding. Consequently as shown in Table 2, by using Kahan summation for model weight updates, pure 16-bit training match the model accuracy attained by 32-bit precision training across all the applications in our experiments. This validates that stochastic rounding and Kahan summation can enable pure 16-bit training algorithms to match the model accuracy of 32-bit training.

Memory efficiency and model accuracy trade-off Additionally, we show that stochastic rounding and Kahan summation can be combined for pure 16-bit training, which exposes a memory efficiency and model accuracy trade-off for practitioners to exploit. As an example, in Figure 4 we demonstrate this trade-off by incrementally replacing stochastic rounding with Kahan summation on various model weights in the DLRM model on the Kaggle dataset. As we apply Kahan summation to more model weights, the weight memory cost increases by up to $2\times$. As this cost increases

Table 2: **Pure 16-bit training can match the model accuracy attained by 32-bit training.** With stochastic rounding or Kahan summation for model weight updates, pure 16-bit training can attain 0.1% lower to 0.2% higher absolute value for validation accuracy metrics across applications.

Model	Dateset (Metric)	32-bit	16-bit Stochastic	16-bit Kahan	Standard 16-bit
ResNet18	CIFAR10 (acc%)	95.45 ± 0.07	95.33 ± 0.08	95.36 ± 0.07	94.23 ± 0.12
ResNet50	ImageNet (acc%)	75.70 ± 0.05	75.45 ± 0.03	75.61 ± 0.14	67.10 ± 0.24
DLRM	Kaggle (auc%)	80.27 ± 0.01	80.18 ± 0.02	80.26 ± 0.01	78.49 ± 0.08
	Terabyte (auc%)	80.32 ± 0.00	80.25 ± 0.00	80.32 ± 0.00	78.79 ± 0.02
BERT	MNLI (acc%)	84.26 ± 0.08	84.35 ± 0.12	84.45 ± 0.03	77.53 ± 0.07
	Wiki103 (ppl)	5.50 ± 0.50	5.84 ± 0.53	5.45 ± 0.51	56.88 ± 1.77
DeepSpeech2	Librispeech (WER)	62.71 ± 0.07	62.85 ± 0.07	62.87 ± 0.18	69.42 ± 0.22

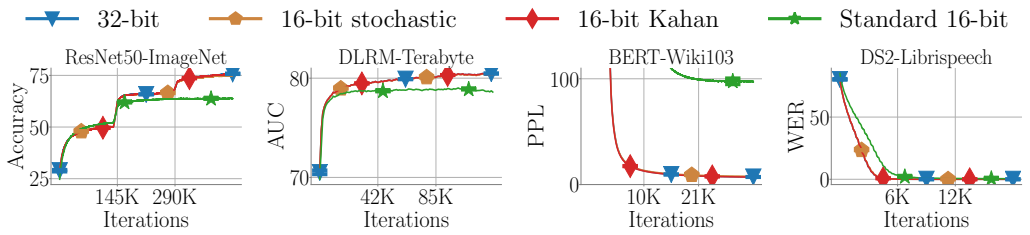

Figure 4: **Training accuracy for pure 16-bit training** With stochastic rounding or Kahan summation enabled for model weight updates, pure 16-bit training matches 32-bit precision training in terms of training accuracy with negligible differences across the applications in our experiments.

we also observe up to 0.04% improvement in AUC. This exploits a memory efficiency and model accuracy trade-off that users should consider when deciding which technique to leverage.

5 Related Work

There is a plethora of research work on low-precision training for deep learning models. On certain specific models such as convolutional or recurrent neural networks, training with fixed point or floating point precisions lower than 16-bit has been shown to be feasible when using customized techniques (Wang et al., 2018b; Zhou et al., 2016; Hubara et al., 2017; Courbariaux et al., 2014; Ott et al., 2016). Our study focuses on generic training algorithms for new general-purpose deep learning accelerators. State-of-the-art low-precision generic training algorithms use mixed precision, still requiring accelerators to contain 32-bit compute units (Micikevicius et al., 2017; Kalamkar et al., 2019; Mellempudi et al., 2019). Our study presents algorithms requiring only 16-bit units in the accelerator architecture, potentially unlocking substantially improved accelerator performance.

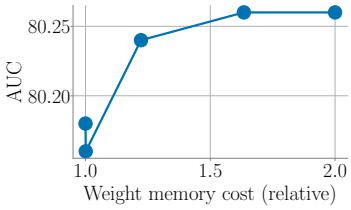

Figure 5: **Efficiency and accuracy trade-off.** With stochastic rounding and Kahan summation on different parts of the DLRM-Kaggle, it attains higher model accuracy at the cost of more weight memory.

6 Conclusion

In this paper we study pure 16-bit training algorithms that require only 16-bit compute units. We show that nearest rounding on model weight updates is the primary cause of convergence and model accuracy degradation in standard pure 16-bit training. To alleviate this issue, we apply two existing techniques: stochastic rounding and Kahan summation. With these techniques, we demonstrate that pure 16-bit training can match the model accuracy of 32-bit precision training across many deep learning models. Our study suggests that it is feasible to design high-accuracy deep learning accelerators using only 16-bit compute units if stochastic rounding and Kahan algorithm are supported.

REFERENCES

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pp. 173–182, 2016.

Mikel Antonana, Joseba Makazaga, and Ander Murua. Reducing and monitoring round-off error propagation for symplectic implicit runge-kutta schemes. *Numerical Algorithms*, 76(4):861–880, 2017.

Neil Burgess, Jelena Milanovic, Nigel Stephens, Konstantinos Monachopoulos, and David Mansell. Bfloat16 processing for neural networks. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, pp. 88–91. IEEE, 2019.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

CriteoLabs. Display advertising challenge, 2014. URL https://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/.

CriteoLabs. Criteo 1tb click logs dataset, 2018. URL https://ailab.criteo.com/criteo-1tb-click-logs-dataset/.

Christopher De Sa, Matthew Feldman, Christopher Ré, and Kunle Olukotun. Understanding and optimizing asynchronous low-precision stochastic gradient descent. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 561–574, 2017.

Christopher M De Sa, Ce Zhang, Kunle Olukotun, and Christopher Ré. Taming the wild: A unified analysis of hogwild-style algorithms. In *Advances in neural information processing systems*, pp. 2674–2682, 2015.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Sameh Galal, Ofer Shacham, John S Brunhaver II, Jing Pu, Artem Vassiliev, and Mark Horowitz. Fpu generator for design space exploration. In *2013 IEEE 21st Symposium on Computer Arithmetic*, pp. 25–34. IEEE, 2013.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pp. 1737–1746, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Greg Henry, Ping Tak Peter Tang, and Alexander Heinecke. Leveraging the bfloat16 artificial intelligence datatype for higher-precision computations. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, pp. 69–76. IEEE, 2019.

Michael Hopkins, Mantas Mikaitis, Dave R Lester, and Steve Furber. Stochastic rounding and reduced-precision fixed-point arithmetic for solving neural ordinary differential equations. *Philosophical Transactions of the Royal Society A*, 378(2166):20190052, 2020.

Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14. IEEE, 2014.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

Intel. Bfloat16-hardware numerics definition, 2018. URL https://software.intel.com/content/www/us/en/develop/download/bfloat16-hardware-numerics-definition.html.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pp. 8571–8580, 2018.

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. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 1–12, 2017.

W Kahan. Further remarks on reducing truncation errors, commun. *Assoc. Comput. Mach*, 8:40, 1965.

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset, 2009. URL https://www.cs.toronto.edu/~kriz/cifar.html.

Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. In *Conference On Learning Theory*, pp. 2–47. PMLR, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S Vetter. Nvidia tensor core programmability, performance & precision. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 522–531. IEEE, 2018.

Naveen Mellempudi, Sudarshan Srinivasan, Dipankar Das, and Bharat Kaul. Mixed precision training with 8-bit floating point. *arXiv preprint arXiv:1905.12334*, 2019.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019. URL https://arxiv.org/abs/1906.00091.

Nvidia. Floating point and ieee 754 compliance for nvidia gpus, 2020. URL https://docs.nvidia.com/cuda/floating-point/index.html.

Joachim Ott, Zhouhan Lin, Ying Zhang, Shih-Chii Liu, and Yoshua Bengio. Recurrent neural networks with limited numerical precision. *arXiv preprint arXiv:1608.06902*, 2016.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5206–5210. IEEE, 2015.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimization towards training a trillion parameter models. *arXiv preprint arXiv:1910.02054*, 2019.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Josef Stoer and Roland Bulirsch. *Introduction to numerical analysis*, volume 12. Springer Science & Business Media, 2013.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018a.

Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *Advances in neural information processing systems*, pp. 7675–7684, 2018b.

Hantian Zhang, Jerry Li, Kaan Kara, Dan Alistarh, Ji Liu, and Ce Zhang. Zipml: Training linear models with end-to-end low precision, and a little bit of deep learning. In *International Conference on Machine Learning*, pp. 4035–4043, 2017.

Jian Zhang, Jiyan Yang, and Hector Yuen. Training with low-precision embedding tables. In *Systems for Machine Learning Workshop at NeurIPS*, volume 2018, 2018.

Tianyi Zhang, Zhiqiu Lin, Guandao Yang, and Christopher De Sa. QPyTorch: A low-precision arithmetic simulation framework. *arXiv preprint arXiv:1910.04540*, 2019.

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.