Don’t Waste Your Bits! Squeeze Activations and Gradients for Deep Neural Networks via TINYSCRIPT

Fangcheng Fu 1 2  Yuzheng Hu 1  Yihan He 1  Jiawei Jiang 3  Yingxia Shao 4  Ce Zhang 3  Bin Cui 1 5

Abstract
Recent years have witnessed intensive research interests on training deep neural networks (DNNs) more efficiently by quantization-based compression methods, which facilitate DNNs training in two ways: (1) activations are quantized to shrink the memory consumption, and (2) gradients are quantized to decrease the communication cost. However, existing methods mostly use a uniform mechanism that quantizes the values evenly. Such a scheme may cause a large quantization variance and slow down the convergence in practice.

In this work, we introduce TINYSCRIPT, which applies a non-uniform quantization algorithm to both activations and gradients. TINYSCRIPT models the original values by a family of Weibull distributions and searches for “quantization knobs” that minimize quantization variance. We also discuss the convergence of the non-uniform quantization algorithm on DNNs with varying depths, shedding light on the number of bits required for convergence. Experiments show that TINYSCRIPT always obtains lower quantization variance, and achieves comparable model qualities against full precision training using 1-2 bits less than the uniform-based counterpart.

1. Introduction
Deep learning (DL) has brought revolutionary changes to machine learning and made remarkable improvements on various complicated tasks such as image recognition, natural language processing, recommendation, and autonomous driving (He et al., 2016; Devlin et al., 2019; Arun & Govindan, 2018; Chen et al., 2019; Zhang et al., 2020; Gharihshah et al., 2020). Recently, with the incredible surge of big data and trend of edge-computing, many researchers have devoted efforts to training deep neural networks (DNNs) on edge devices. However, due to the large amount of data and increasingly complex model architectures, training DNNs becomes both memory- and communication-bounded. For one thing, activations (a.k.a. feature maps) in forward propagation need to be stored for gradient computation in backward propagation, which leads to large memory footprints. For another, gradients need to be sent to central servers or exchanged among workers for model update, resulting in heavy network transmission.

One approach to alleviating the burden of memory consumption and communication overhead is reducing the data size by quantizing values to fewer bits, at the price of precision loss. For instance, gradient quantization accelerates the overall training speed since the communication time is decreased (Alistarh et al., 2017; Wen et al., 2017), whilst (Chakrabarti & Moseley, 2019) copes with the memory bottleneck by quantizing activations into 4 bits.

Given a tensor $x$ and a set of quantization points $s = \{s_i\}_{i=0}^n$ satisfying $-M = s_0 < s_1 < \ldots < s_n = M$, where $M = ||x||_{\infty}$ acts as a scaling factor, the stochastic quantization on each element $x_i \in [s_t, s_{t+1})$ is defined as

$$\tilde{x}_i = \begin{cases} s_t, & \text{with probability } (s_{t+1} - x_i)/(s_{t+1} - s_t) \\ s_{t+1}, & \text{with probability } (x_i - s_t)/(s_{t+1} - s_t) \end{cases}.$$ 

In a sense, each element is more likely to be quantized

$1$ (Alistarh et al., 2017) scales gradient with $L_2$ norm in their analysis, but uses $L_{\infty}$ norm in practice.
to the nearest point, but the stochastic technique ensures unbiasedness, i.e., $E[\hat{x}] = E[x]$. Although unbiased, the errors between the quantized and original values, which we refer to as quantization variance, can significantly affect the convergence. More aggressive quantization (smaller $n$) results in a higher variance, which inevitably slows down the convergence or even leads to divergence. To obtain better convergence, one may use more quantization levels (larger $n$) to reduce the variance or use a smaller step size to restrict the effect of variance. Nonetheless, such approaches cannot completely solve the problem and will either lower the compression rate or require more training steps.

We notice that existing works for DNNs essentially adopt a value range into $T$ to summarize, we list the main contributions as below:

Framework for DNNs called $T$ this work proposes a novel, distribution-aware quantization technique in training DNNs, which is similar to ours, but only focuses on uniform quantization and conforms to 4 bits. In contrast, we leverage the distribution property and derive a more advanced non-uniform quantization method.

2. Related Works

In this section, we briefly go through the prior researches and preliminary literature related to our topic.

Weibull-related Random Variables Let random variable (r.v.) $X$ with PDF $p(x) = \frac{\lambda x^{k-1}}{\lambda^k} \exp(-\left(\frac{x}{\lambda}\right)^k)$ be a double-Weibull r.v. parameterized by shape $k > 0$ and scale $\lambda > 0$, which is extended from the Weibull distribution. A r.v. $Y$ satisfying $P(|Y| \geq y) \leq \alpha \exp(-\left(\frac{y}{\theta}\right)^k)$ for all $y > 0$ and for some $\theta > 0$ is called a sub-Weibull r.v. with tail $\theta$, denoted by $Y \sim \text{subW}(<\theta>$ (Vladimirova et al., 2018). Note that $X \sim \text{subW}(\frac{1}{4})$ since $P(|X| \geq x) = \exp(-\left(\frac{x}{\theta}\right)^k)$.

Low-memory Training Various works have been developed to reduce the memory footprint of activations. A popular technique is to swap-out activations from GPU memory to CPU memory during forward pass and swap-in during backward pass (Rhu et al., 2016; Wang et al., 2018b). Our work is orthogonal to these works. One can first quantize the activations and then swap-out in order to reduce IO overhead of swapping. Furthermore, the swapping technique does not fit edge devices since there is no sufficient host memory. (Chakrabarti & Moseley, 2019) adopts the quantization technique in training DNNs, which is similar to ours, but only focuses on uniform quantization and conforms to 4 bits. In contrast, we leverage the distribution property and derive a more advanced non-uniform quantization method.

Gradient Compression Since distributed training is common nowadays, numerous researches try to quantize gradients into smaller sizes (Wen et al., 2017; Alistarh et al., 2017; Wu et al., 2018; Mishchenko et al., 2019). The most notable gradient quantization works for DNNs are TernGrad and QSGD. TernGrad proves the convergence of gradient quantization methods. However, it only considers an extreme case with $n = 2$. QSGD discusses the trade-off between $n$ and convergence, but it merely focuses on uniform quantization. Note that TernGrad is a special case for both TinYSCRIPT and QSGD, since $s = [-M, 0, M]$ if $n = 2$.

Another line of research focuses on how to sparsify the gradients by only sending large values and dropping the small ones (Stich et al., 2018; Wang et al., 2018; Wang & et al., 2018a; Sun et al., 2019). These approaches can highly reduce the size of gradients. Our work is orthogonal to the sparsification methods. Since values conform to a unimodal and symmetric distribution, there is a large portion of values quantized to the middle point (in most cases, zero) if we use fewer bits. Therefore, the quantized tensor is automatically sparsified. Moreover, since sparsification accumulates the filtered values for further updates, the memory consumed by activations even increases. So whether the sparsification technique is applicable for activations is still unknown. Consequently, we focus on the quantization-based methods in this work and leave the sparsification as future work.
Low-precision training A number of works have focused on training with low-precision values, e.g., 8-bit fixed-point numbers (Drumond et al., 2018; Banner et al., 2018; Wang et al., 2018c; Cambier et al., 2020). Empirical results have shown that such approaches are suitable for low-power specialized hardware that supports low-precision arithmetic. Nevertheless, our work differs from these methods—we only quantize activations and gradients and all computations are carried out in full precision. As we will describe later, quantization error does not propagate across layers in DNNs. Hence, our method can use fewer bits in quantization. To the best of our knowledge, there is no 4-bit floating-point representation since there will be insufficient bits for mantissa and exponent. As a result, we do not compare with this kind of works due to the difference in goals.

Other Non-uniform Quantization There are also works that adopt non-uniform quantization. The most distinguishable technique is using quantiles. ZipML (Zhang et al., 2017) formulates the problem of searching quantization points to reduce variance, which is similar to our work. However, a heuristic algorithm based on histograms is adopted to solve the problem, which actually results in quantiles. SketchML (Jiang et al., 2018; 2020) uses a data structure called quantile sketch to approximate the quantiles. Moreover, these works mainly focus on generalized linear models (GLMs) rather than DNNs. As a result, we do not compare with this kind of works due to the difference in goals.

LQ-Nets (Zhang et al., 2018) focuses on training QNNs by learning quantizers. Specifically, it learns an encoding for each coordinate and computes the quantization points via learning quantizers. Specifically, it learns an encoding for each coordinate and computes the quantization points via

\[ \text{dequan} \quad \n(\hat{l}) = \n(\hat{l})(\hat{p}(\hat{l}-1)) \]

\[ g^{(l)} = n^{(l)}(\bar{p}^{(l-1)})^T \]

\[ \text{Update} \quad \text{quan} \quad \text{send} \]

\[ \text{quantization error does not propagate across layers in DNNs.} \]

\[ \text{Furthermore, the update procedure of the encoding} \]

\[ \text{requires extra memory and has the same time complexity} \]

\[ \text{as the vanilla uniform quantization.} \]

3. Methodology

In this section, we first formulate the problem of minimizing quantization variance, and then introduce the proposed distribution-aware quantization framework, namely TinySCRIPT.

3.1. Overview and Problem Formulation

Figure 2 illustrates the quantization for activations and gradients with an example of fully connected layer. In forward pass, the \( l \)-th layer takes as input the activations \( v^{(l-1)} \) from previous layer and generates \( v^{(l)} \) to the next layer. Then \( v^{(l-1)} \) is quantized to reduce memory usage. In backward pass, in addition to propagating derivatives \( n^{(l)} \), gradients \( g^{(l)} \) are computed with the quantized activation \( \tilde{v}^{(l-1)} \). Upon model update, we quantize \( g^{(l)} \) to reduce network transmission, and the final update will be performed with the quantized gradient \( \tilde{g}^{(l)} \).

Obviously, the quantization strategy only perturbs the computation of gradients (\( g \)), whilst forward and backward units (\( v \) and \( n \)) are unaffected. It ensures that the error caused by quantization will not be accumulated as the network goes deeper. To limit the impact brought by quantization, our goal is to minimize the expected quantized variance

\[ \mathbb{E}[||x||_2^2] \]

for a given tensor \( x \). Motivated by the facts that quantization variance on \( g \) can be reckoned as part of stochastic variance and \[ ||n\tilde{v}^T - nv^T||_F \]

Suppose the size of \( x \) is \( D \) and each value in \( x \) conform to some distribution \( p(x) \), we have

\[ \mathbb{E}_p[||x||_2^2] = \int_{\mathbb{R}^D} p(x) \mathbb{E}(x - \tilde{x})^2 dx \]

\[ = D \sum_{t=0}^{n-1} \int_{s_t}^{s_{t+1}} p(x)(x - s_t) dx \]

\[ = D \sum_{t=0}^{n-1} \int_{s_t}^{s_{t+1}} p(x)(s_{t+1} - x)(x - s_t) dx \]

Given the number of quantization levels \( n \), an ideal quantization mechanism is expected to minimize Equation 1. There are two critical issues: (1) a good estimation for the distribution \( p(x) \), and (2) a method to search quantization points \( s \) for a given \( p(x) \). In the rest of the paper, we assume tensors to be quantized are flattened as vectors and \( || \cdot || \) denotes the \( L_2 \) norm for simplicity.

3.2. Modeling of Distribution

The distribution \( p(x) \) is vital to the minimum expected variance achieved by the optimal \( s \). An initial thought is to fit
p(x) using methods like kernel density estimation or empirical distribution. However, it would be expensive if we fit p(x) on-the-fly. Worse, the empirical distribution is usually non-differentiable, making the searching of s infeasible. As a result, we determine to assume a prior on when we compute the mean and standard deviation.

The restriction on k guarantees p(x) is monotonic on negative or positive side. The parameter k also controls the shape of p(x) — for a smaller k, p(x) is more central tendency and has a longer tail. Therefore p(x) is able to express distributions with various tails. In Figure 3, and notice that in addition to unimodal and symmetric, p(x) is central tendency and can be easily parallelized with SIMD instructions.

Nevertheless, it needs careful investigation to choose a good prior. One may guess values are Gaussian distributed due to the Central Limit Theorem, whereas it is not true in almost all cases. (Bernstein et al., 2018) concludes that gradient values are unimodal and symmetric, but they focus on the convergence of their algorithm rather than formulating the distribution. We plot some empirical distribution of activations and gradients in Figure 3, and notice that in addition to unimodal and symmetric, p(x) is central tendency and long-tailed. Inspired by this, we propose to model values with the double-Weibull distribution:

\[ p(x) = \frac{k}{2\lambda} \left( \frac{|x|}{\lambda} \right)^{k-1} \exp\left( -\left( \frac{|x|}{\lambda} \right)^k \right), \quad 0 < k \leq 1. \]

The restriction on k guarantees p(x) is monotonic on negative or positive side. The parameter k also controls the shape of p(x) — for a smaller k, p(x) is more central tendency and has a longer tail. Therefore p(x) is able to express distributions with various tails. In Section 4.1, we will discuss the rationality of the modeling in-depth.

In order to fit the distribution efficiently, we develop a moment-based method to approximate the distribution with mean and standard deviation (stddev). We present the method in Algorithm 1 and go through its details below.

![Figure 3. Histograms and fitted curves of activations (top row) and gradients (bottom row) of randomly chosen conv layers and training steps when training WideResNet34 on Cifar10. x-axis represents value range and y-axis represents counts of values.](image)

Algorithm 1 Moment-based distribution fitting

**Input:** tensor \( x \)

**Output:** fitted distribution

1: function INITCVTABLE(table step \( \text{delta} = 0.001 \))
2: Initialize empty table \( \mathcal{T} \)
3: for \( k \leftarrow 0.1 \) to \( \text{delta} \) do
4: \( \mathcal{CV} \leftarrow \sqrt{\Gamma(1 + 2/k)/\Gamma^2(1 + 1/k) - 1} \)
5: Insert \( \langle k, \mathcal{CV} \rangle \) to \( \mathcal{T} \)
6: return \( \mathcal{T} \)
7: function MOMENTESTIMATE(stats \( \text{stat}, \text{table} \, \mathcal{T} \))
8: \( \mu, \sigma \leftarrow \text{stat}.\text{mean}, \text{stat}.\text{stddev} \)
9: Lookup \( \langle k', \mathcal{CV}' \rangle \) in \( \mathcal{T} \) with binary search such that \( |\sigma/\mu - \mathcal{CV}'| \) is the smallest
10: \( X \leftarrow \mu/\Gamma(1 + 1/k') \)
11: return \( \langle k', \lambda' \rangle \)
12: if \( \mathcal{T} \) not initialized then
13: \( \mathcal{T} \leftarrow \text{INITCVTABLE}() \)
14: Compute positive and negative statistics for tensor \( x \)
15: \( \text{pos} \leftarrow \text{MOMENTESTIMATE}(x.\text{pos}_\text{stat}, \mathcal{T}) \)
16: \( \text{neg} \leftarrow \text{MOMENTESTIMATE}(x.\text{neg}_\text{stat}, \mathcal{T}) \)
17: return \( \langle \text{pos}, \text{neg} \rangle \)

Since moments \( \mathbb{E}(X^t) = \lambda'\Gamma(1 + t/k) \), where \( \Gamma(\cdot) \) is the gamma function, the coefficient of variation (CV) satisfies

\[ CV = \frac{\sigma}{\mu} = \sqrt{\frac{\mathbb{E}(X^2) - (\mathbb{E}X)^2}{\mathbb{E}X}} = \sqrt{\frac{\Gamma(1 + 2/k)}{\Gamma^2(1 + 1/k) - 1}}, \]

where \( \mu, \sigma \) are mean and stddev. Therefore, we compute a \( \langle k, CV \rangle \) lookup table beforehand (line 1-6). Given a tensor, we fit two distributions for positive and negative sides individually (line 14-16). Once \( CV \) is computed, we lookup the closest one from the table with a corresponding \( k' \) (line 9). Furthermore, since \( CV \) is monotonic decreasing w.r.t. \( k \), the lookup can be executed with binary search efficiently. Finally, we can compute \( \lambda' \) inspired by the first-order moment \( \mathbb{E}(X) = \lambda\Gamma(1 + 1/k) \) (line 10).

3.3. Searching of Optimal Quantization

Although we have modeled values via a Weibull prior, whether the minimum of Equation 1 exists in the domain of quantization points \( s \) is still unknown. If yes, how to find the optimal \( s^* \) that minimizes Equation 1 for any distribution \( \langle k, \lambda \rangle \) and level \( n \) remains unsolved.

To answer these questions, we formulate the searching of \( s \) that achieves minimum variance as an optimization problem. For simplicity, we make the following adjustments: (1) we only consider the case of \( \lambda = 1 \), whereas we can easily migrate the discussion to arbitrary \( \lambda \) with a scaling factor; (2) we leverage the symmetry in distribution to assume \( n \) to be an even number (so that zero is always a quantization point).
and only consider the non-negative side; (3) we assume there is a sufficiently large value $M$ ($M < \infty$) such that all (absolute) values are not greater than $M$. Now the objective we study becomes the expected quantization variance on the non-negative side:

$$V(\hat{s}) := \frac{\hat{n}}{n} \int_{s_t}^{s_{t+1}} kx^{k-1} \exp(-x^k)(\hat{s}_{t+1} - x)(x - \hat{s}_t)dx,$$

where $\hat{n} = n/2$ and $\{\hat{s}\}_{t=0}^{\hat{n}} = \{s\}_{t=n}$.

Thus, the searching of optimal quantization points can be turned into the following problem:

$$\arg\min_{\hat{s}} V(\hat{s}) \text{ s.t. } 0 = \hat{s}_0 < \hat{s}_1 < \ldots < \hat{s}_{\hat{n}} = M.$$

**Theorem 1.** There is a unique stationary point $\hat{s}^*$ of $V(\hat{s})$ in the domain of $\hat{s}$, and $V(\hat{s})$ attains minimum only at $\hat{s}^*$.

Theorem 1 implies that the minimum quantization variance is attained at a unique stationary point. As a result, we can use gradient descent to solve the optimization problem. The partial derivative of $\hat{s}_t (t = 1, 2, \ldots, \hat{n} - 1)$ is given by:

$$\frac{\partial V(\hat{s})}{\partial \hat{s}_t} = -\Gamma(1 + \frac{1}{k}, \hat{s}_t) + \Gamma(1 + \frac{1}{k}, \hat{s}_{t+1}) + \hat{s}_{t+1} \exp(-\hat{s}_{t+1}) - \hat{s}_t \exp(-\hat{s}_t) (2)$$

where $\Gamma(\cdot, \cdot)$ denotes the incomplete gamma function. A general method of searching optimal quantization points can be: (1) randomly initialize $\hat{s}$, e.g., uniform quantization points; (2) iteratively update $\hat{s}$ via gradient descent by Equation 2 until converged.

Algorithm 2 shows how we compute the quantization points on-the-fly. Optimal points are pre-computed (line 1-8). Motivated by three-sigma rule of thumb, we set $M = 3\sigma$ heuristically (line 5). For tensor $x$, we first fit distributions by Algorithm 1, then we lookup the optimal points and scale by $\lambda$ (line 9-11). Finally the quantization points are reorganized in ascending order (line 16).

### 3.4. Implementation and Complexity Analysis

To implement TINYSCRIPT, we customize forward and backward propagation with PyTorch (Paszke et al., 2019), and we leverage NVIDIA Thrust library to perform statistics reduction and parallel quantization. For practical concerns, we further make two adjustments as described below.

First, we follow (Chakrabarti & Moseley, 2019) to integrate quantization of activations with the checkpointing technique (Chen et al., 2016). For instance, we regard consecutive $bn$-$relu$-$conv$ chain as one block and only quantize and store one activation (input of $bn$), whilst the rest (inputs of $relu$ and $conv$) are re-computed in backward pass. Such an approach is worthy since the time cost for $bn$-$relu$ is much smaller than $conv$ and is able to reduce both memory consumption and quantization overhead. In practice, we store the batch-wise mean and stddev in full precision since they are small in size. Therefore, it is obvious that the re-computed outputs of $bn$ are identical to the directly quantized version in expectation. Consequently, in our following analysis, we simply assume all activations are quantized rather than re-computed.

Second, we apply the bucketing strategy where each tensor is sliced into buckets and each bucket is quantized individually. Such a bucketing technique is widely used to decrease the quantization variance (Seide et al., 2014; Alistarh et al., 2017). By default, we set bucket size as 4096, and we exclude small tensors (<10K values, typically, gradients of $bn$ layers and batch-wise mean and stddev).

We finish this section by summarizing the complexity of TINYSCRIPT. To quantize a tensor, we need to (1) compute statistics of each bucket, (2) fit Weibull distributions, (3) calculate optimal points, and finally (4) perform the quantization. Fortunately, the time complexities of step 2 and 3 are both $O(1)$ for each bucket, and different buckets can be processed concurrently. The most time-consuming phases are step 1 and 4, which are also required in uniform quantization and can be highly parallelizable with SIMD instructions. As a result, the proposed non-uniform quantization algorithm shares the same complexity as that of uniform.

### Algorithm 2 Quantization Calculation

**Input:** fitted distribution $(pos, neg)$, min and max values $x.min, x.max$, #intervals $n$

**Output:** Optimal quantization points

1. function INITQUANTIBLE(max allowed $\hat{n}_{max} = 8$)
2. Initialize empty table $Q$
3. for $k \leftarrow T.keys()$ do
4. for $\hat{n} \leftarrow 2, 3, \ldots, \hat{n}_{max}$ do
5. $M \leftarrow 3\sqrt{\Gamma(1 + 2/k) - \Gamma^2(1 + 1/k)}$
6. Iteratively compute $\{\hat{s}^*\}_{\hat{n}-1}$ by Equation 2 with $\hat{s}_0 = 0$ and $\hat{s}_\hat{n} = M$
7. Insert $(k, \hat{n}, \{\hat{s}^*\}_{\hat{n}-1})$ to $Q$
8. return $Q$
9. function QUANCALC(dist $(k, \lambda)$, table $Q$)
10. Lookup $\hat{s}^* \leftarrow Q[k, n/2]$
11. return $\hat{s}^* \ast \lambda$
12. if $Q$ not initialized then
13. $Q \leftarrow$ INITQUANTIBLE()
14. $\hat{s}_{pos} \leftarrow$ QUANCALC(pos, $Q$)
15. $\hat{s}_{neg} \leftarrow$ QUANCALC(neg, $Q$).negative().reverse()
16. return $\{x.min, \hat{s}_{neg}, 0, \hat{s}_{pos}, x.max\}$

---

Don’t Waste Your Bits! Squeeze Activations and Gradients for Deep Neural Networks via TINYSCRIPT
4. Analysis

In this section, we describe our analysis of the proposed non-uniform quantization algorithm.

4.1. Rationality of Weibull Prior

To comprehensively understand the rationality of the modeling method in Section 3.2, we investigate the distribution property of forward and backward units for an MLP model depicted in Figure 4. To begin with, we introduce the analysis of forward propagation in (Vladimirova et al., 2019).

Assumption 1. For an MLP model depicted in Figure 4, we migrate the assumptions from (Vladimirova et al., 2019): (1) All weights are independent and have zero-mean normal distribution. Specifically, let the model have L layers and $W^{(l)}$ have size $C_{l-1} \times C_l$, then we have $W^{(l)}_{i,j} \sim N(0, \sigma^2)$ for $1 \leq l \leq L, 1 \leq i \leq C_{l-1}, 1 \leq j \leq C_l$; (2) The input data are independent with model weights.

Theorem 2. [Theorem 3.1 of (Vladimirova et al., 2019)] Consider an MLP depicted in Figure 4. If Assumption 1 holds, then for any $1 \leq l \leq L, 1 \leq i \leq C_l$, we have $u_{i}^{(l)}, v_{i}^{(l)} \sim \text{subW}(l/2)$.

Assumption 1 implies a Gaussian prior on weights, and indicates the data are unseen before, which holds at the very first training steps or in online settings. (Vladimirova et al., 2019) concludes that activations conform to a sub-Weibull distribution parameterized by depth $l$. In order to study the gradient distribution, we extend the analysis to backward propagation. However, since it is non-trivial to deduce on the backward units, we have to make another assumption:

Assumption 2. We make the Gradient Independence assumptions following (Yang & Schoenholz, 2017): (1) the model weights used in backward propagation are different copies from those in forward propagation, but drawn i.i.d. from the same distributions; (2) for $1 \leq l \leq L$, the backward units $m^{(l)}$, $n^{(l)}$ are independent from the forward units $u^{(l)}$, $v^{(l)}$.

Assumption 2 is implicitly used in (Schoenholz et al., 2016) and formally introduced by (Yang & Schoenholz, 2017). Although seeming unrealistic, Assumption 2 is actually a well-known statement in the mean field theory since it plays a crucial role in the derivation of back-propagation. Due to space limitation, we skip the discussion of correctness of Assumption 2 in this work and refer readers to other surveys (Yang, 2019). Note that $m^{(l)}$, $n^{(l)}$ are correlated with $u^{(l+1)}$, $v^{(l+1)}$ according to $W^{(l)}$, so we do not assume they are independent. Next, we introduce Theorem 3.

Theorem 3. Assume the MLP model in Theorem 2 satisfies Assumption 2, then for any $1 \leq l \leq L, 1 \leq i \leq C_l$, we have $m_i^{(l)}$, $n_i^{(l)} \sim \text{subW}((L - l)/2)$.

Theorem 3 shows that the backward units conform to a sub-Weibull distribution parameterized by $L - l$. Since $\nabla W^{(l)} = m^{(l)}(u^{(l-1)})^T$, it gives an intuitive idea that gradients are also Weibull-related variables. To formal, we conclude with Theorem 4.

Theorem 4. Considering the MLP model in Theorems 2, 3, for any $1 \leq l \leq L, 1 \leq i \leq C_{l-1}, 1 \leq j \leq C_l$, we have $\nabla W_{ij}^{(l)} \sim \text{subW}((L - 1)/2)$.

Remark 1. Although the above discussion is based on MLP and requires assumptions which are ideal in practice, it provides a hint on the distribution of activations and gradients — for DNNs where $L$ is large, the distribution is strongly related to a sub-Weibull distribution with a large $\theta$ (or a double-Weibull distribution with a small $k$). As stated in Section 3.2, a smaller $k$ leads to more central tendency and long-tailed, which corresponds to the empirical distribution of activations and gradients in DNNs.

4.2. Understanding Convergence against Variance

Till now, readers might suspect the significance of exhaustingly minimizing the quantization variance. In other words, it is non-intuitive how quantization variance affects convergence. To answer this question, we try to analyze the convergence of SGD with quantization on the following (non-convex) empirical risk minimization problem:

$$
\min_{w \in \mathbb{R}^D} f(w) = \frac{1}{N} \sum_{i} f_i(w), \quad w_{t+1} = w_{t} - \alpha \hat{g}_t,
$$

where $\alpha$ is step size, $g_t = \nabla f(w_t)$, and $\hat{g}_t$ represents the quantized version of $g_t$. Our analysis is based on Assumption 3 and given in Theorem 5.

Assumption 3. For all $\forall i \in \{1, 2, \ldots, N\}$, $w, w' \in \mathbb{R}^D$, we make the assumptions below following (Allen-Zhu, 2017):
Let $\sigma$ be the variance of model parameters. For a $\alpha$ larger than $\sigma$, the fractional error of $\alpha\hat{\sigma}$ is smaller than \frac{1}{\sqrt{2\pi}} \cdot \frac{\sigma}{\alpha}$, which gives a more general picture. We plot the expected variance for non-uniform and uniform quantization, which turns Theorem 5 into similar analysis in previous works.

**Theorem 5.** Suppose we run SGD with quantization on an object satisfying Assumption 3 with constant step size $\alpha < 2/L_2$. Assume quantization satisfies $\mathbb{E}[||\tilde{x} - x||] \leq 2\mathbb{E}[||x||]$ for any $x \in \mathbb{R}^D$. After $T$ runs, select $\tilde{w}_T$ randomly from $\{w_0, w_1, ..., w_{T-1}\}$. Then we have

$$\mathbb{E}[||\nabla f(\tilde{w}_T)||^2] \leq \frac{2(f(w_0) - f^*)}{(2\alpha - \alpha^2L_2)^T} + \frac{\alpha L_2 (\sigma^2 + Q^2\sigma_0^2)}{2 - \alpha L_2}.$$

As $T$ increases, the convergence of SGD with quantization is influenced by quantization variance $Q$, especially when the optimization is stuck in a sharp minima/minimum. For uniform quantization, $Q$ can be bounded by $O(||x||_{\infty}/\hat{n})$, which turns Theorem 5 into similar analysis in previous works (Alistarh et al., 2017; Bernstein et al., 2018). However, this only represents a worst-case scenario, which can hardly occur unless all values are close to one side of the intervals but unfortunately quantized to another side. Hence, we should consider the expectation of quantization variance, which gives a more general picture. We plot the expected variance in Figure 5 and summarize the discussion below.

**Remark 2.** Upon Theorem 5, we further assume gradients follow double-Weibull distribution with $k = O(2/(L - 1))$ inspired by Theorem 4. According to Figure 5, we conclude that $\mathbb{E}[Q] = O(q_1^2/q_2^2 \log \hat{n})$ for some constants $q_1 > 0, q_2 > 0$. By doing so, we have two important findings:

(1) Since the expected variance exponentially increases w.r.t. $L$, a larger $\hat{n}$ (which turns into more bits) is needed to obtain comparable convergence for deeper networks;

(2) The proposed non-uniform quantization method can achieve similar variance as the uniform-counterpart with half quantization levels.

5. Experiments

5.1. Experimental Setup

We conduct the experiments on a GPU server equipped with 4 Titan RTX GPUs. We focus on CNNs training since it is a well-studied task and models have non-trivial sizes. Specifically, we consider the widely-used VGG and ResNet architectures (Simonyan & Zisserman, 2014; He et al., 2016; Zagoruyko & Komodakis, 2016). All experiments are conducted on Cifar10 and ImageNet (Krizhevsky et al., 2009; Deng et al., 2009).

5.2. Effectiveness of Non-uniform Quantization

We first assess the effectiveness of the proposed non-uniform quantization algorithm over Cifar10 dataset and WideResNet34 (WRN34) model. Due to space limitation, we only take gradients of a conv layer in the last residual block as a representative since it has the largest size, whilst similar results are also observed on activations and other models.

**Measurement of distribution** To validate our analysis of distribution, we empirically compare gradients fitted with different distributions regarding KL divergence. Specifically, we fit Weibull distribution with Algorithm 1, and fit Gaussian and Laplace distributions via scipy.stats package. As shown in the left of Figure 6, throughout the training, fitting with Weibull distribution can always achieve the lowest KL divergence. In fact, Laplace distribution is a special case of double-Weibull when $k = 1$, thus it does not have enough complexity to describe the distribution. Whilst Gaussian distribution has the poorest performance due to the stark difference to distribution.

**Measurement of quantization** The most critical issue we concern about is the quantization variance. As a result, we measure the quality of quantization by computing the related variance $||\tilde{g} - g||^2/||g||^2$ in different iterations. Results are

---

Since Section 4 targets at feed-forward networks, whether recurrent neural networks (RNNs) share similar distribution properties remains unknown. Therefore, we leave the discussion and evaluation on RNNs as our future work.
shown in the right of Figure 6. In general, our non-uniform approach consistently achieves lower variance compared to the uniform-based counterpart at the same quantization level $n = 4$ since it better adapts to the distribution and executes an optimal quantization. In contrast, the uniform-based approach requires doubling $n$ ($n = 8$) to obtain a comparable variance, which is consistent with Figure 5.

5.3. Convergence Performance

We compare the convergence of TinyScript, uniform-based quantization (Vanilla) $^4$, and full precision (FP32) over Cifar10 and ImageNet datasets. Figure 7 presents the convergence curves and Table 1, 2 list the best performances.

**Convergence on Cifar10** We first experiment on Cifar10 dataset with VGG19, WRN34, and ResNet50. We use SGD with momentum of 0.9 to train the models. In most cases, using a small learning rate is beneficial for the quantization methods since the error of each iteration can be restricted. However, a smaller learning rate requires more iterations to converge and even leads to more communication rounds in distributed training. As a result, to achieve a fair comparison, hyper-parameters are tuned for FP32 and applied to TinyScript and Vanilla directly. We make 5 runs for each method and report the mean and standard deviation. Overall, FP32 achieves the best model performance since it never suffers from quantization variance. Both TinyScript and Vanilla can converge to considerable accuracy (within 1% drop). However, Vanilla requires more quantization levels (larger $n$) to converge properly and even diverges in all cases when $n = 4$. This is unsurprising since TinyScript provides a lower quantization variance as described above, and hence gains better convergence. Furthermore, it is worthy to note that both TinyScript and Vanilla entail a higher performance drop on ResNet50 than VGG19 and WRN34. It matches our discussion in Section 4.2 that a deeper network would cause higher quantization variance and require more quantization levels.

**Convergence on ImageNet** We then compare the performance of each method on ImageNet dataset with ResNet50. Following (He et al., 2016), we set batch size as 256, initial learning rate as 0.1, and use SGD with momentum of 0.9. In general, FP32 still obtains the best model performance and the convergence curves of the quantization competitors are close to that of FP32. With a larger level $n = 32$, Vanilla incurs 0.43% drop in accuracy, whilst suffers from a 0.92% drop with $n = 16$ and diverges for smaller $n$. In contrast, the gap between FP32 and TinyScript is only 0.69% when $n = 8$. Furthermore, Vanilla diverges for both $n = 4$ and $n = 8$ owing to the high quantization variance, whilst the validation error of TinyScript is 10.5% when $n = 4$. The experiment on this large-scale dataset verifies the ability of TinyScript to achieve sound model performance with a more aggressive quantization.

**Compression** Finally, we compare the compression performance of TinyScript and Vanilla with the results in Figure 8. Although the existing uniform quantization conveys a possibility to train DNNs with a small level $n$ so that memory consumption and communication cost can be reduced, our non-uniform quantization further decreases

---

Table 1. Best testing error on Cifar10 (in percentage). We report the mean and stddev of 5 runs. *-* indicates we skip the level since comparable accuracy has been achieved.

<table>
<thead>
<tr>
<th>Model</th>
<th>Batch Size</th>
<th>Init. LR</th>
<th>TinyScript $\times$4</th>
<th>Vanilla $\times$8</th>
<th>Vanilla $\times$16</th>
<th>FP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG19</td>
<td>256</td>
<td>0.1</td>
<td>7.98±0.20</td>
<td>7.81±0.17</td>
<td>7.15±0.03</td>
<td></td>
</tr>
<tr>
<td>WRN34</td>
<td>256</td>
<td>0.1</td>
<td>5.72±0.12</td>
<td>5.78±0.23</td>
<td>5.23±0.25</td>
<td></td>
</tr>
<tr>
<td>ResNet50</td>
<td>128</td>
<td>0.05</td>
<td>6.45±0.11</td>
<td>diverge</td>
<td>5.61±0.14</td>
<td>5.57±0.07</td>
</tr>
</tbody>
</table>

Table 2. Best top-5 validation error on ImageNet (in percentage).

<table>
<thead>
<tr>
<th>Method</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>7.97</td>
</tr>
<tr>
<td>FP32</td>
<td>7.05</td>
</tr>
</tbody>
</table>

---

Figure 7. Convergence curves. For Cifar10, we plot the mean and stddev of training loss over 5 runs. For ImageNet, we plot the top-5 validation error. We start from the quantization level $n = 4$ and double the level until comparable accuracy has been achieved.
$n$ to one-half or even one-fourth. From the perspective of information theory, it requires at most $\log(n+1)$ bits to represent $n$ levels. Thus, TinyScript enjoys approximately 0.8-1.8 bits saving compared to the uniform-based method.

6. Conclusion and Possible Future Works

This work proposes TinyScript, a non-uniform quantization framework for DNNs. TinyScript models the values with a family of Weibull distributions and leverages the distribution property to achieve minimum quantization variance. Empirical results show that TinyScript obtains lower quantization variance than the uniform-based mechanism, and therefore achieves a higher compression rate with model accuracy on par with full precision training.

Beyond this work, we wish to discover the strength of the proposed non-uniform quantization method in three directions. First, since all DNNs discussed in this work are feedforward networks, it will be interesting to see whether our method fits RNNs. Second, whether the modeling on distribution works well on model weights is worth discussing. If yes, a model compression technique based on our method would be possible. Third, a dynamic control, e.g., choosing $n$ w.r.t. $k$, may greatly improve the power of our method. We will leave the exploration of these topics as future works.

Acknowledgements

The work is supported by National Key R&D Program of China (No. 2018YFB1004403), National Natural Science Foundation of China (NSFC) (No. 61832001, 61702016, 61702015, U1936104), PKU-Baidu Fund 2019BD006, Beijing Academy of Artificial Intelligence (BAAI), PKU-Tencent joint research Lab, and Fundamental Research Funds for the Central Universities 2020RC25. CZ and the DS3Lab gratefully acknowledge the support from the Swiss National Science Foundation (Project Number 200021_184628), Swiss Data Science Center, Alibaba, eBay, Google Focused Research Awards, Oracle Labs, Zurich Insurance, Chinese Scholarship Council, and the Department of Computer Science at ETH Zurich.

References


