# Monte Carlo Gradient Quantization

Gonçalo Mordido
Hasso Plattner Institute
Potsdam, Germany
goncalo.mordido@hpi.de

Matthijs Van Keirsbilck
NVIDIA
Berlin, Germany
matthijsv@nvidia.com

Alexander Keller
NVIDIA
Berlin, Germany
akeller@nvidia.com

## Abstract

*We propose Monte Carlo methods to leverage both sparsity and quantization to compress gradients of neural networks throughout training. On top of reducing the communication exchanged between multiple workers in a distributed setting, we also improve the computational efficiency of each worker. Our method, called Monte Carlo Gradient Quantization (MCGQ), shows faster convergence and higher performance than existing quantization methods on image classification and language modeling. Using both low-bit-width-quantization and high sparsity levels, our method more than doubles the rates of existing compression methods from 200× to 520× and 462× to more than 1200× on different language modeling tasks.*

## 1. Introduction

Much of the workload in current deep neural networks are matrix-vector products, whose performance may be increased by executing low-precision operations. This allows for a more efficient use of available hardware as compared to using full-precision values. In addition, computations on zero-valued operands may be skipped to greatly improve performance, depending on the sparsity levels. Moreover, to improve the training speed, we may distribute computations over multiple workers or nodes.

Modern neural networks grow ever larger and more complex. Hence, distributed training becomes more and more important in order to keep training times reasonable. However, the distributed setting also introduces a communication overhead as the different workers have to synchronize gradients. This overhead may grow significantly in larger computing systems. Therefore, to improve training efficiency at scale, it is essential to reduce the amount of communication exchanged between workers.

Repeated random sampling approaches or Monte Carlo methods, have shown great promise to initialize, regularize [20, 24], and optimize [17] neural networks. Recently, Monte Carlo Quantization (MCQ) [14] has been proposed, which uses importance sampling to quantize both weights and activations of pre-trained neural networks. We extend this approach to quantize and sparsify gradients at training time. By combining this approach with sparse tensor representations, our method may also be used to lower the computational load of each worker.

Our approach, called Monte Carlo Gradient Quantization (MCGQ), offers several insights and modifications to the originally proposed algorithm that greatly improve the performance under severe gradient compression regimes. We start by proposing dynamic sampling, a novel strategy to automatically learn optimal sampling hyperparameters at each layer during training. Moreover, we propose sampling proportional to the magnitude of the accumulated gradients at each iteration, leveraging information from rarely sampled gradients to reduce quantization losses even and especially at high compression levels.

We evaluated MCGQ for several tasks, models, and optimizers. Our experiments show faster convergence than existing quantization methods on logistic regression and improved test and train performances on image classification tasks, and higher compression rates at similar performance as well as better compression for language modeling. MCGQ exhibits higher compression rates and less performance degradation when compared to existing gradient sparsification methods.

## 2. Related Work

The communication bandwidth of distributed training may be reduced by gradient quantization and sparsification. Current approaches mostly focus on one of the above or require two different algorithms to achieve both quantization and sparsification. In contrast, MCGQ combines both naturally, with different quantization and sparsity levels being adjustable by the amount of sampling performed.

Focusing on existing gradient quantization approaches, 1-bit SGD [18] and signSGD [3] proposed to quantize all gradients to 1-bit, with the latter work also introducing $signum$ which uses the sign of the momentum to perform the updates. TernGrad [25] quantized to 2-bits, but used full-precision gradients for the last layer and further training modifications on some of the experiments. QSGD [2] proposed a family of compression schemes with several bit-width levels, ranging from 2, 4 and 8 bits. Mem-SGD [21] kept a record of the accumulated errors in memory to achieve a similar convergence rate as 32-bit SGD. Ef-signSGD [7] improved convergence of sign based approaches by introducing error feedback in the next optimization step.

Gradient sparsification approaches often allow for greater degrees of compression by using encoded communication, such as run-length encoding (RLE) or Elias coding. Most existing methods rely on thresholds to sparsify gradients [22, 1, 10], which may be hard to determine in practice. Deep Gradient Compression or DGC [10] used thresholding on the accumulated gradients to achieve high compression rates. However, they require multiple training tricks to achieve the floating-point gradient baseline's performance, such as warm-up training for a varied amount of epochs depending on the data set. [5] extended 1-bit SGD, and manually chose a fixed fraction of the gradients to be nonzero. Finally, AdaComp [4] proposed to dynamically control the rate of parameter updates by analyzing their local gradient activity.

In contrast to most of the aforementioned methods, we use the exact same training setup as the floating-point gradient training. This significantly simplifies the adoption of MCGQ and sparse gradient training for existing models and respective training procedures.

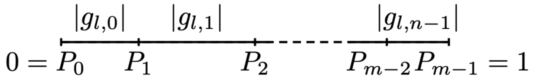
## 3. Monte Carlo Gradient Quantization

In this work, we propose to use importance sampling proportional to the gradient magnitude to approximate floating-point gradients. Similar to sampling discrete probability densities, the quality of the approximation increases with the amount of sampling performed [13].

### 3.1. Gradient Quantization and Sparsification

In standard stochastic gradient descent (SGD), a loss function $f : \mathbb{R}^d \to \mathbb{R}$ is minimized w.r.t parameter $x$ at iteration $t$ using a learning rate $\gamma$ by the following update rule: $x_{t+1} := x_t - \gamma g_{x_t}$, where $g_{x_t}$ is the stochastic gradient such that $\mathbb{E}[g_{x_t}] = \nabla_{x_t} f$.

In our approach, we first compute all stochastic gradients for all layers and then use importance sampling to approximate the floating-point values in a layer-wise manner. Given layer $l$ at a certain iteration, we first normalize the $n$ gradient components by their $l_1$-norm such that $\sum_{k=0}^{n-1} |g_{l,k}| = \|g_l\|_1 = 1$. From this probability density function (PDF) we construct the respective cumulative density function (CDF) to define a partition of the unit interval by $P_m := \sum_{k=0}^{m-1} |g_{l,k}|$:



(1)

Finally, given $N$ uniformly distributed samples $x_i$ in $[0, 1)$, we approximate each $g_{l,k}$ by counting the number of times its corresponding interval has been selected during the sampling process. The value $g_{l,k_i}$ corresponding to sample $x_i$ is denoted as $g_{l,k_i} \in \{-1, 0, 1\}$, where 0 means the sample $x_i$ missed the interval of $g_{l,k}$ and $\pm 1$ represents a hit, i.e. $P_k \leq x_i < P_{k+1}$, with the sign corresponding to the sign of the respective component $g_{l,k}$ of the gradient. We have:

$$g_{l,k} \approx \frac{1}{N} \sum_{i=0}^{N-1} g_{l,k_i}. \tag{2}$$

Our importance sampling technique is further described in Algorithm 1. The number of samples $N$ is controllable by the hyperparameter $K$. We denote the quantized (integer) gradient as $g_{l_{int}}$. To reduce the cost of finding the index of the interval hit by a sample, we make use of the fact that the CDF is monotonically increasing. We use $start_{idx}$ to keep track of the index of the value hit by the last sample. For the next sample, we need to search only over the values in the CDF with an index larger than or equal to $start_{idx}$. The full quantization requires only a single pass over the CDF, thus reducing the search cost from $\mathcal{O}(S^2)$ to $\mathcal{O}(S)$ where $S = len(g_l)$.

In contrast to the original MCQ algorithm [14], we do not sort the values before constructing the PDF to reduce the approximation noise since such noise proved to be beneficial for convergence speed in practice. This was also previously observed by [15].

Algorithm 2 describes the importance sampling technique to quantize and sparsify gradients during training. For simplicity, we assume an optimizer invariant to gradient scale, *e.g.* ADAM or Lazy ADAM. When using a scale-sensitive optimizer like SGD or RMSProp, the quantized gradients $g_{l_{int}}$ or, alternatively, the learning rate, must be scaled by $\frac{\|g_l\|_1}{len(g_l) \times K}$.

Note that even when training in a single worker environment, *i.e.* not using a distributed setting, with a scale-variant optimizer, MCGQ still reduces the computational cost of the parameter updates by inducing high levels of sparsity.

### 3.2. Dynamic Sampling

Since our algorithm is performed in a layer-wise manner, different layers may require different sparsity or quantization levels, which depend on the sampling hyperparameter $K$. To lower the computational cost, it is desirable to set K as low as possible on each layer while minimizing performance loss. By using the distance between quantized and

**Algorithm 1:** Monte Carlo Gradient Quantization (MCGQ).

```
Input: gradients of layer $l$ $g_l$, sampling amount $K$
Init.: $\xi \leftarrow \text{random}(0,1)$,
       $N \leftarrow \lceil len(g_l) \times K \rceil$,
       $g_{l_{int}} \leftarrow [0] \times len(g_l)$,
       $start_{idx} \leftarrow 0$
// construct PDF and CDF
$g_{l_{PDF}} \leftarrow \frac{|g_l|}{\|g_l\|_1}$;
$g_{l_{CDF}} \leftarrow \text{cumsum}(g_{l_{PDF}})$;
// perform importance sampling
for $i = 0, \ldots, N-1$ do
    // get sample $x_i$ on interval $[0,1]$
    $x_i \leftarrow \frac{\xi + i}{N}$;
    // find hit index
    $hit_{idx} \leftarrow \text{argmax}(g_{l_{CDF}}[start_{idx} :]|g_{l_{CDF}} \geq x_i)$;
    $start_{idx} \leftarrow hit_{idx}$;
    // count according to the original sign
    $g_{l_{int}}[hit_{idx}] \leftarrow g_{l_{int}}[hit_{idx}] + sign(g_l[hit_{idx}])$;
end
Output: $g_{l_{int}}$
```

**Algorithm 2:** Gradient quantization and sparsification.

```
Input: learning rate $\delta$, $n$ parameters of layer $l$ $x_l$,
       sampling amount $K$
Init.: $g_l \leftarrow [0] \times n$
// compute gradients
for $i = 0, \ldots, n-1$ do
    $g_l[i] \leftarrow \text{StochasticGradient}(x_{l_i})$;
end
// quantize and sparsify
$g_{l_{int}} \leftarrow \text{MCGQ}(g_l, K)$;
// update parameters
$x_l \leftarrow x_l - \delta g_{l_{int}}$;
```

full-precision distributions, we propose to learn different K values automatically for each layer during training. We used the Wasserstein-1 distance, also known as the Earth-Mover distance [23], due to the inherent relation between quantization errors and Wasserstein distances [8].

The first Wasserstein distance represents the minimum cost needed to transform one probability distributions into another, with a transformation cost function $\gamma$ and using infinitesimal random mass transfers. The joint distribution of X and Y corresponds to the space where mass is transferred between floating-point and quantized probability distributions $\mathbb{P}_f$ and $\mathbb{P}_q$, respectively, with $\Pi(\mathbb{P}_f, \mathbb{P}_q)$ the set of all joint distributions of random variables X and Y:

$$\text{Wasserstein}(\mathbb{P}_f, \mathbb{P}_q) = \inf_{\gamma \in \Pi(\mathbb{P}_f, \mathbb{P}_q)} \mathbb{E}_{(X,Y)\sim\gamma}[\|X - Y\|], \tag{3}$$

The dynamic sampling method is demonstrated in algorithm 3. The key idea is that each layer learns a number of samples to approximate the floating-point distribution up to a sufficient degree, controlled by the step-size $\delta_K$ and the initial value given to $K$. For increased performance, a grid-search over possible $\delta_K$ and initial $K$ values is advised.

**Algorithm 3:** Dynamic sampling.

```
Input: learning rate $\delta_K$, floating-point gradients of
       layer $l$ $g_l$, quantized gradients of layer $l$
       $g_{l_{int}}$, Wasserstein dist. $d'$ and sampling
       amount $K$ from previous iteration
// compute distance between PDFs
$d \leftarrow \text{Wasserstein}(\frac{g_l}{\|g_l\|_1}, \frac{g_{l_{int}}}{\|g_{l_{int}}\|_1})$;
// calculate distance difference
$\Delta_d = d' - d$;
// update K
$K \leftarrow K + \delta_K \times \Delta_d$;
```

### 3.3. Local Gradient Accumulation

Due to the use of importance sampling, parameters with consistently small gradients might rarely be updated during training. Gradient information is lost when not sampling such gradients at a given iteration, which we found to cause convergence issues when using low sampling rates. To ameliorate this, we propose to sample proportional to the accumulated gradients instead, retaining gradient information throughout different iterations.

Accumulating gradients throughout training was proposed by [10], where parameters with smaller gradients are updated less frequently during training. In our use case, this allows us to use less sampling, promoting higher sparsity and lower bit-width quantization at each iteration, while smaller gradient updates are still considered over time. The accumulated gradients variant of MCGQ is described in Algorithm 4. Note that, similarly to Algorithm 2, the quantized gradients must be scaled when using an optimizer sensitive to gradient scale.

### 3.4. Compression Scheme

After sparsification and quantization, one may use compression schemes to further reduce the communication cost, which is important in a distributed training environment with multiple workers. Examples of such techniques are run-length encoding (RLE) [10] or Elias coding [2].

**Algorithm 4:** Gradient quantization and sparsification using accumulated gradients.

```
Input: learning rate $\delta$, $n$ parameters of layer $l$ $x_l$,
       accumulated gradients of layer $l$ $ag_l$,
       sampling amount $K$
// compute gradients
for $i = 0, \dots, n-1$ do
    $g_l[i] \leftarrow$ StochasticGradient($x_{l_i}$);
end
// accumulate gradients
$ag_l \leftarrow ag_l + g_l$;
// quantize and sparsify
$g_{l_{int}} \leftarrow$ MCGQ($ag_l, K$);
// update parameters
$x_l \leftarrow x_l - \delta g_{l_{int}}$;
// reset used accumulated gradients
$ag_l[g_{l_{int}} \neq 0] \leftarrow 0$
```

Since MCGQ enables both low bit-width as well as highly sparse gradient representations, RLE is a good choice to deal with zero values, by compressing a sequence of zeros with a counter and sending only the number of bits required to represent the non-zero quantized values.

Our compression scheme for the tensor of $n$ quantized gradients of layer $l$, $g_{l_{int}} = \{g_{l_{int_0}}, g_{l_{int_1}}, \dots, g_{l_{int_{n-1}}}\}$, is as follows:

1. Send the $B_g \in \mathbb{N}$ bits required to represent the signed quantized representation of the gradient with maximum magnitude:

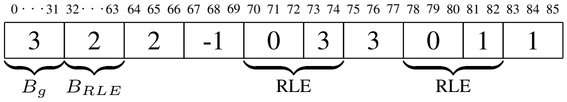
$$B_g = 1 + \left\lfloor \log_2 \left( \max_{0 \leq i \leq n-1} |g_{l_{int_i}}| \right) + 1 \right\rfloor .$$

2. Send the $B_{RLE} \in \mathbb{N}$ bits required to represent the highest count $c_0 \in \mathbb{N}$ of consecutive zeros:

$$B_{RLE} = \lfloor \log_2 (c_0) + 1 \rfloor .$$

3. Send non-zero values and first zero occurrences using $B_g$-bits, and send the count of each zero-valued sequence using $B_{RLE}$-bits.

For illustration purposes, let us consider $g_{l_{int}} = [2, -1, 0, 0, 0, 3, 0, 1]$. Following the aforementioned nomenclature, we have $B_g = 3, c_0 = 3$, and $B_{RLE} = 2$, with the following compression data transmitted:



Thus, in total, 86 bits are used to represent $g_{l_{int}}$ instead of $len(g_{l_{int}}) \times 32 = 256$ bits, achieving a compression of $\approx 2.97\times$. The compression rates reported in our experimental results (Section 4) are calculated at each iteration and are relative to the total number of bits required to represent the floating-point gradients for all L layers, *i.e.* $\sum_{l=0}^{L-1} len(g_l) \times 32$.

Note that, when using an optimizer sensitive to gradient scale, scaling of the gradients to their original form is necessary before updating the parameters. In this case, one extra float (32 bits) with the gradient norm $\|g_l\|_1$ is required to be communicated for each layer $l$.

### 3.5. Distributed Setting

Gradient compression may be used in a distributed environment to reduce the communication between workers in the communication phase. In this use case, each worker calculates its gradients separately and then all gradients computed by the different workers are merged by using all-reduce [6]. The parameters stored locally in each worker can then be updated by averaging the total sum of the gradients. Note that each worker will accumulate gradients independently. In the case dynamic sampling is used (Section 3.2), each worker can have a different $K$ so appropriate rescaling may be necessary.

On a single node, MCGQ achieves faster computation time than the floating-point baseline when using sparse tensor representations on a high-sparsity level, despite the additional computation needed for the sampling process. When scaling a multi-worker system, communication becomes more of a bottleneck than compute, so even at high sample counts trading some extra compute for lower communication cost is a favorable trade-off.

## 4. Experimental Results

We evaluated MCGQ's performance on a variety of tasks: logistic regression (Section 4.1), image classification (Section 4.2), and language modeling (Section 4.3). For each task, we compare to previous work that quantizes and/or sparsifies gradients. Note that we did not perform multi-node training, however, when comparing to methods that did train in a distributed setting, we used the same total batch size as in their experiments.

### 4.1. Logistic Regression

First, we followed the experiments on logistic regression presented in Mem-SGD ([21]) in order to compare MCGQ to their method, QSGD [2], and 32-bit SGD on the epsilon data set [19]. We used the following objective function: $\frac{1}{N_T} \sum_{i=0}^{N_T-1} \log(1 + \exp(-b_i a_i^T x)) + \frac{\lambda}{2}\|x\|^2$, where $a_i \in \mathbb{R}^d$ and $b_i \in \{-1, +1\}$ are the samples, and $N_T$ is the number of training samples. L2-regularization ($\lambda = \frac{1}{N_T}$)

was used and the initial learning rate was set to 1.0 and updated at each iteration $t$ of epoch $ep \in \{0, 1, \ldots, 10\}$ by

$$\frac{1}{1 + \lambda \times (ep \times N_T + t)}.$$

The epsilon dataset contains 400k samples with 2k nonzero dimensions, *i.e.* no sparsity. Comparison results using MCGQ ($K = 1.0$) are shown in Figure 1. Under similar training settings, we observe that both MCGQ with and without gradient accumulation converges faster than all compared methods, including 32-bit SGD.

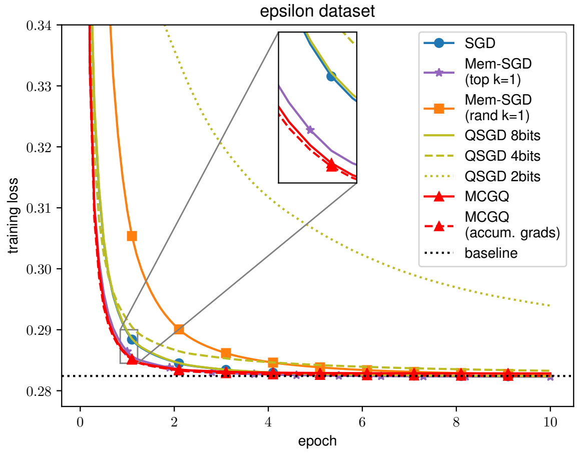


Figure 1: Comparison results on the epsilon data set. Both variants of MCGQ show a faster convergence rate than all the compared methods, even more so when using accumulated gradients.

To study the effects of using different sampling rates and accumulated gradients, we take a look at the convergence of MCGQ with $K = \{0.1, 0.2, \ldots, 1.0\}$ in Figure 2. Using the variant without gradient accumulation, with higher sampling rates most gradient information is still maintained at each iteration, resulting in overall convergence. Note that using lower sampling rates has less negative effects when accumulating gradients since no information is lost during each iteration. The baseline results in Figures 1 and 2 are the scikit-learn's LogisticSGD [16] results as reported by Mem-SGD.

## 4.2. Image Classification

We further tested our approach on image classification, specifically using ResNet-110 and ResNet-18 on the CIFAR-10 and CIFAR-100 datasets [9], respectively.

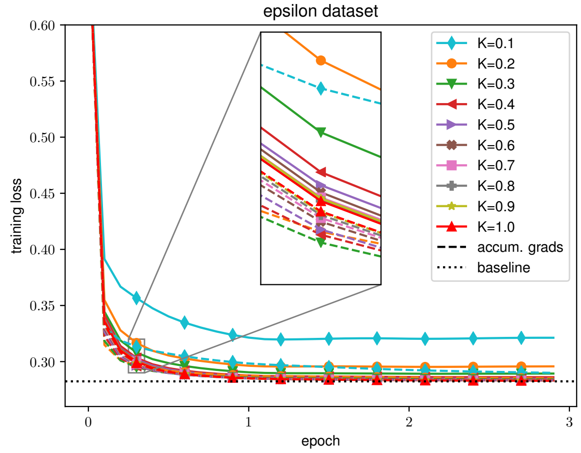


Figure 2: MCGQ on the epsilon data set with different sampling rates and gradient variants. Using larger sampling rates and accumulating the gradients improves convergence over time.

### 4.2.1 CIFAR-10

For CIFAR-10 we used ResNet-110 with the training details described in [1], reaching 91.72% accuracy with batch size 128, learning rate decaying by a factor of 10 at epochs 81 and 122, and Lazy Adam as the optimizer.

Instead of reporting absolute accuracy for the quantized variants, we use the accuracy difference relative to the baseline models of each respective work. Due to the flexible quantization level of our approach, we report the average of the maximum number of bits ('# max bits') required to represent all gradients in a given layer at each iteration. We used dynamic sampling for both variants of MCGQ, using the same initial value for $K$ for all layers. We note that dynamic sampling was not used for any of the further experiments of this paper due to the increased performance of using accumulated gradients with a static $K$.

Figure 3 shows the comparison results against QSGD and sparsity methods, *i.e.* Gradient Dropping and Deep Gradient Compression (DGC). Regarding QSGD, the accumulated gradients variant of MCGQ achieves equivalent 2-bit QSGD performance at $\approx 2.1$ bits. At higher bit-widths, however, MCGQ outperforms QSGD, especially with gradient accumulation. This variant of MCGQ also reaches close to floating-point performance at around 3 bits instead of QSGD's 4 bits. Moreover, both accumulated gradients and gradient magnitude variants improve the baseline's accuracy by +0.51% and +0.50% at $\approx 5$ and $\approx 6$ bits, respectively, outperforming 8-bit QSGD's +0.33%. We also observe that at higher bit-width, *i.e. with sufficient sam-*

[1] https://github.com/bearpaw/pytorch-classification

*pling*, both MCGQ variants perform similarly. We further note that although the gradients of DGC and Gradient Dropping are highly sparse, they are not quantized. We perform a thorough comparison to sparsity methods in Section 4.3.

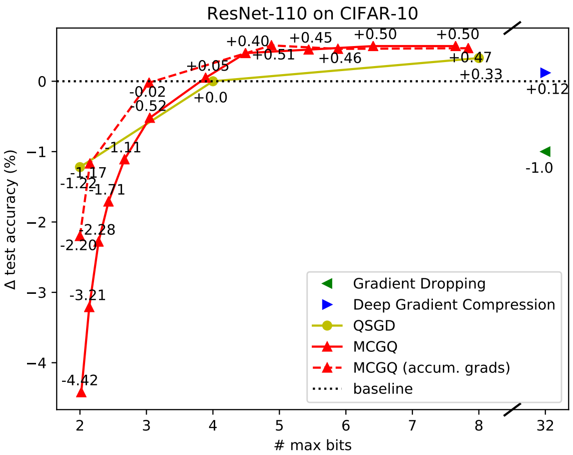


Figure 3: Test accuracy for ResNet-110 on CIFAR-10, varying bit-width of the gradients. Both variants of MCGQ generally outperform QSGD at similar gradient bit-widths, with gradient accumulation improving the overall performance.

Figure 4 compares the training characteristics of MCGQ to full-precision SGD. Both MCGQ variants show an identical training loss at the baseline while achieving higher test accuracy.

### 4.2.2 CIFAR-100

We compared MCGQ with gradient accumulation to several binary quantization methods: signSGD, signum, and ef-signSGD, using ResNet-18 on CIFAR-100. We follow the experimental setup [2] from ef-signSGD [7], with a batch size of 128 and averaging the results over 3 runs. We evaluated MCGQ at the same compression level of 32× as the compared binary methods, using the compression scheme described in Section 3.4.

Contrary to [7], we did not grid-search for the best learning rate (LR) for our method and simply use the best learning rate reported by signSGD and ef-signSGD. We also use this LR for the 32-bit SGD baseline. Moreover, we compare to the best reported values for each method in the original paper. The results in Table 1 show that MCGQ outperforms the compared methods in terms of accuracy at the similar compression rates. Figure 5 illustrates the train and test accuracy throughout training, where we observe that MCGQ converges faster than the rest of the methods.

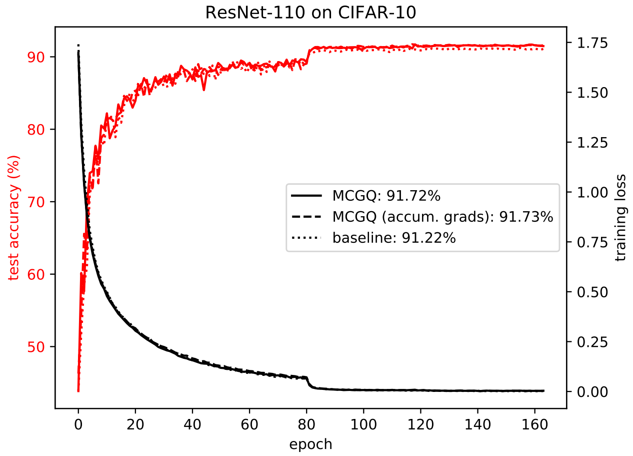


Figure 4: Training loss and accuracy for ResNet-110 on CIFAR-10. Line styles denote the different MCGQ variants. Red lines denote top-1 test accuracy, while black lines represent train loss. All MCGQ variants show similar training loss and better test accuracy than the 32-bit gradient baseline.

| ResNet-18 on CIFAR-100 | | | |
|---|---|---|---|
| Method | Acc. (%) | Comp. | LR |
| SGD (baseline) | 74.02 | 1× | $5.6e^{-2}$ |
| **MCGQ** | **74.89** | **≈ 32×** | $\mathbf{5.6e^{-2}}$ |
| ef-signSGD | 74.43 | 32× | $5.6e^{-2}$ |
| signSGD | 73.14 | 32× | $5.6e^{-2}$ |
| signum | 72.20 | 32× | $3.2e^{-4}$ |

Table 1: Comparison results on CIFAR-100 using the best learning rate for the compared methods.

| ResNet-18 on CIFAR-100 | | | |
|---|---|---|---|
| Method | Acc. (%) | Comp. | LR |
| SGDm | 75.20 | 1× | $1.0e^{-2}$ |
| SGD (baseline) | 69.75 | 1× | $1.0e^{-2}$ |
| **MCGQ** | **72.57** | **≈ 32×** | $\mathbf{1.0e^{-2}}$ |
| ef-signSGD | 69.69 | 32× | $1.0e^{-2}$ |
| signSGD | 67.13 | 32× | $1.0e^{-2}$ |
| signum | 58.90 | 32× | $1.0e^{-2}$ |

Table 2: Comparison results on CIFAR-100 with the same learning rates.

It is a significant advantage if a compression method will not require the modification of any hyperparameter. Hence, we analyze the sensitivity of the aforementioned methods by measuring their performance with the optimal learning rate for SGD with momentum (SGDm) reported in [7] of $1.0e^{-2}$. Results are shown in Table 2. We observe that

[2] https://github.com/epfml/error-feedback-SGD

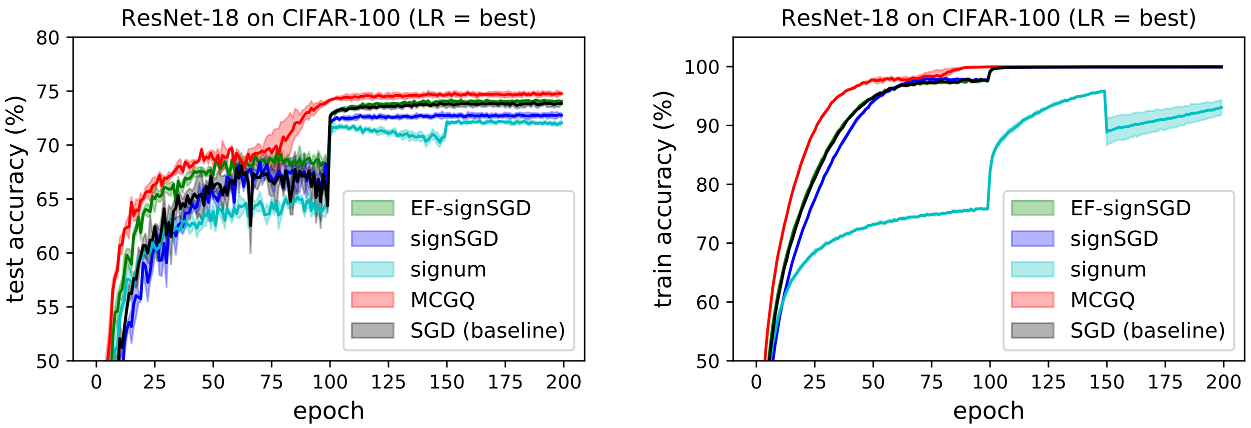


Figure 5: Comparison results on CIFAR-100 of the learning ability while using the searched best learning rates of the compared methods. MCGQ achieves faster convergence than all the compared methods, while ef-signSGD matches the baseline. On the other hand, signSGD and signum show poorer performance.

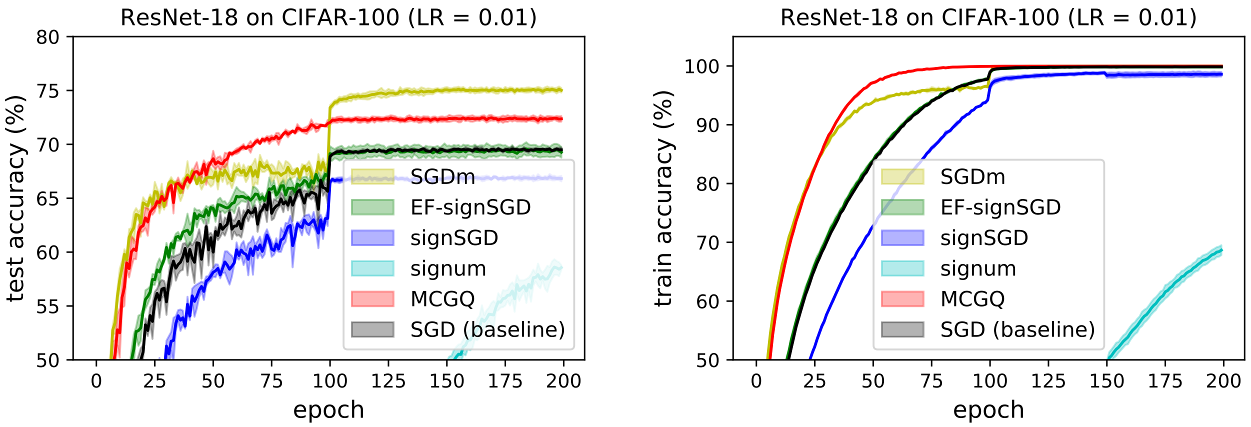


Figure 6: Comparison results on CIFAR-100 of the learning ability with all methods using the best learning rate of SGDm. MCGQ achieves faster training convergence than all the compared methods, including SGDm, while signum fails to be competitive with the rest of the methods under the same number of iterations.

MCGQ is least affected by the suboptimal learning rate, outperforming the 32-bit SGD baseline by +2.82% accuracy. The learning evolution is illustrated in Figure 6, where we observe faster convergence for MCGQ as compared to all other methods.

## 4.3. Language Modeling

We further compared the accumulated gradients variant of our approach to high-compression sparsity approaches. Since MCGQ provides both sparsity and quantization benefits, we are able to achieve higher compression rates than existing methods with the communication scheme previously described in Section 3.4.

### 4.3.1 Penn Treebank Corpus

We evaluated world-level language modeling on the well-known Penn Treebank corpus (PTB) dataset [11], using [12]'s implementation [3]. We used vanilla SGD with gradient clipping on a 2-layer LSTM with 1500 hidden neurons and embedding size of 1430, decaying the learning rate when validation loss does not improve, to replicate the model and training details used by DGC [10]. Since no further training details were provided in the compared work, we used the recommended training configuration of the aforementioned public implementation without weight dropout and batch size 80.

Compression comparisons with DGC are shown in Ta-

[3] https://github.com/salesforce/awd-lstm-lm

| 2-LSTM on PTB | | | |
|---|---|---|---|
| Method | Perplexity ↓ | Gradient size | Comp. |
| Baseline | 82.03 | 194.69 MB | 1× |
| **Δ MCGQ** | **-0.21** | **0.16 MB** | **1218×** |
| **Δ MCGQ** | **-4.30** | **0.42 MB** | **469×** |
| Δ DGC | -0.06 | 0.42 MB | 462× |

Table 3: Compression results on PTB. MCGQ outperforms DGC, having lower perplexity for the same compression.

ble 3. Note that DGC's results are relative to their reported baseline of 72.30 perplexity, as we were unable to reproduce this baseline using the training details in that work. At the same compression rate, our method achieves $\approx -4$ lower perplexity than DGC. At $\approx 1200\times$ compression, MCGQ starts to match the floating-point baseline while still outperforming DGC at a much higher compression rate. Figure 7 compares the training loss of MCGQ with different compression levels to the baseline.

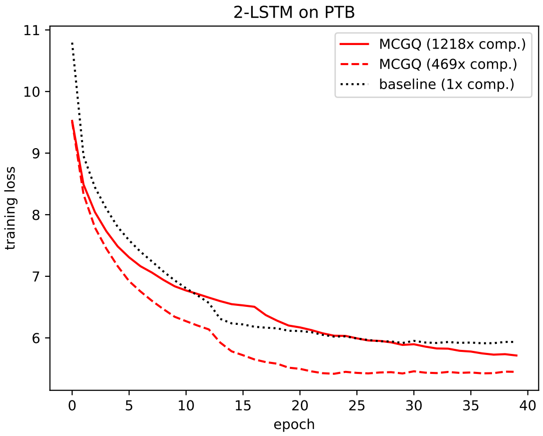


Figure 7: Training losses on PTB of MCGQ at different compression levels. At 469× compression, MCGQ shows significant reductions in the overall training loss. At 1200x compression, the loss is similar to the baseline.

### 4.3.2 Shakespeare

We also evaluated training performance on character-level language modeling on Shakespeare text, using Char-RNN's training settings [4]. For a fair comparison to AdaComp's [4] experiments, we used a 2-layer LSTM of 512 neurons each, trained for 45 epochs with batch size 10.

We used the validation cross-entropy loss of the model's predictions and the true future text as the comparison metric. Table 4 shows the comparison of MCGQ and AdaComp to our method achieving lower loss than the floating-point gradient baseline at 215x compression level. At a similar loss, MCGQ achieves 520x compression compared to 200× for AdaComp. Figure 8 shows the evolution of the loss during training, with MCGQ at different compression rates showing similar behavior as the baseline.

| Char-RNN on Shakespeare | | | |
|---|---|---|---|
| Method | Val. loss ↓ | Gradient size | Comp. |
| Baseline | 1.578 | 13.28 MB | 1× |
| **Δ MCGQ** | **+0.020** | **0.03 MB** | **520×** |
| **Δ MCGQ** | **-0.016** | **0.06 MB** | **215×** |
| Δ AdaComp | +0.020 | 0.07 MB | 200× |

Table 4: Compression results on Shakespeare. MCGQ matches AdaComp at more than double the compression.

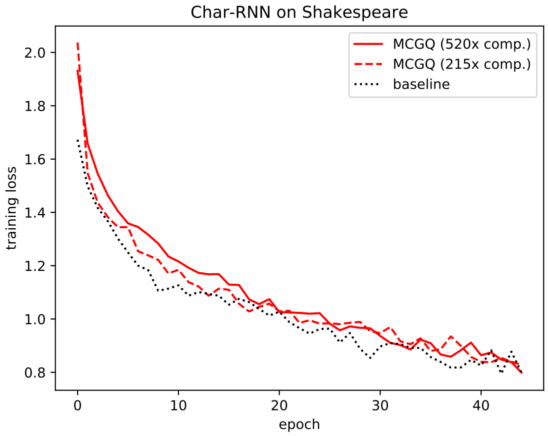


Figure 8: Training losses on Shakespeare of MCGQ at different compression levels. At 215x gradient compression, loss evolution is similar to the baseline. At over 500× compression, the loss is only slightly higher than the baseline.

## 5. Conclusion

In this work, we apply Monte Carlo methods to quantize and sparsify gradients during training. When compared to other quantization approaches, our method achieves faster convergence rates and better test performance on the evaluated tasks. Moreover, the accumulated gradients variant of our method allows for higher gradient compression rates as compared to existing sparsity methods due to both low-bit-width-quantization and high-levels of sparsity. Our method does not require any modification of the training hyperparameters, simplifying adoption. In future work, it would be interesting to evaluate the scalability of MCGQ in a real distributed setting by performing training across multiple workers.

[4] https://github.com/karpathy/char-rnn

# References

[1] Alham Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021*, 2017. 2

[2] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pages 1709–1720, 2017. 2, 3, 4

[3] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signSGD: Compressed optimisation for non-convex problems. *arXiv preprint arXiv:1802.04434*, 2018. 1

[4] Chia-Yu Chen, Jungwook Choi, Daniel Brand, Ankur Agrawal, Wei Zhang, and Kailash Gopalakrishnan. AdaComp: Adaptive residual gradient compression for data-parallel distributed training. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 2, 8

[5] Nikoli Dryden, Tim Moon, Sam Jacobs, and Brian Van Essen. Communication quantization for data-parallel training of deep neural networks. In *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)*, pages 1–8. IEEE, 2016. 2

[6] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999. 4

[7] Sai Karimireddy, Quentin Rebjock, Sebastian Stich, and Martin Jaggi. Error feedback fixes signSGD and other gradient compression schemes. *arXiv preprint arXiv:1901.09847*, 2019. 2, 6

[8] Wolfgang Kreitmeier. Optimal vector quantization in terms of Wasserstein distance. *Journal of Multivariate Analysis*, 102(8):1225–1239, 2011. 3

[9] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 5

[10] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017. 2, 3, 7

[11] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. 7

[12] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. *arXiv preprint arXiv:1708.02182*, 2017. 7

[13] Nicholas Metropolis and Stanislaw Ulam. The Monte Carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949. 2

[14] Gonçalo Mordido, Matthijs Van keirsbilck, and Alexander Keller. Instant quantization of neural networks using Monte Carlo methods. In *2019 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, to appear. 1, 2

[15] Arvind Neelakantan, Luke Vilnis, Quoc Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015. 2

[16] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011. 5

[17] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951. 1

[18] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014. 1

[19] Soeren Sonnenburg, Vojtech Franc, Elad Yom-Tov, and Michele Sebag. Pascal large scale learning challenge. In *25th International Conference on Machine Learning (ICML2008) Workshop. http://largescale.first.fraunhofer.de. J. Mach. Learn. Res*, volume 10, pages 1937–1953, 2008. 4

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 1

[21] Sebastian Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified SGD with memory. In *Advances in Neural Information Processing Systems*, pages 4447–4458, 2018. 2, 4

[22] Nikko Strom. Scalable distributed DNN training using commodity GPU cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015. 2

[23] Sergei Vallender. Calculation of the Wasserstein distance between probability distributions on the line. *Theory of Probability & Its Applications*, 18(4):784–786, 1974. 3

[24] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using DropConnect. In *International conference on machine learning*, pages 1058–1066, 2013. 1

[25] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. TernGrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in neural information processing systems*, pages 1509–1519, 2017. 1