ONLINE MODEL COMPRESSION FOR FEDERATED LEARNING WITH LARGE MODELS

Tien-Ju Yang, Yonghui Xiao, Giovanni Motta, Françoise Beaufays, Rajiv Mathews, Mingqing Chen
Google LLC, Mountain View, CA, U.S.A.

ABSTRACT
This paper addresses the challenges of training large neural networks under federated learning settings: high on-device memory usage and communication cost. The proposed Online Model Compression (OMC) provides a framework that stores model parameters in a compressed format and decompresses them only when needed. We use quantization as the compression method in this paper and propose three methods, (1) per-variable transformation, (2) weight-matrix-only quantization, and (3) partial variable quantization, to minimize its impact on model accuracy. Our experiments on two recent neural networks for speech recognition and two different datasets show that OMC can reduce memory usage and communication cost of model parameters by up to 59% while attaining comparable accuracy and training speed compared with full-precision federated learning.

Index Terms— Federated learning, speech recognition, deep learning, neural network

1. INTRODUCTION
Federated learning (FL) [1, 2] allows training neural network models directly on edge devices (referred to as clients) instead of transferring their data to a server for centralized training to preserve users’ privacy. FL is composed of multiple federated rounds. In a standard federated round, a server model is first transported to clients. Then, the clients train the model on their local data. The trained models are finally returned to the server and aggregated to improve the server model. This process repeats until the server model converges.

FL involves on-device training and model transportation between servers and clients, which lead to two main challenges. The first challenge is that edge devices usually have limited memory available for training. Given the fact that recent Automatic Speech Recognition (ASR) models typically contain hundreds of millions of parameters or more [3], keeping these parameters uncompressed in memory may exceed the available memory. Although there is a significant effort in the field on reducing memory usage of parameters during inference, such as quantization-aware training [4, 5], it is usually at the cost of higher memory usage during training (Sec. 2). Reducing the memory usage of parameters during training with FL is less explored. The second challenge is the high communication cost. Communication can be costly in terms of speed [6] and bandwidth, and transporting models uncompressed also burdens the communication network.

In this paper, we propose Online Model Compression (OMC) to address the above challenges of on-device FL. Different from regular FL, where each client keeps, updates, and transports uncompressed parameters, OMC keeps and transports the parameters in a compressed format. During training, when an operation needs the value of a compressed parameter, OMC decompresses it on-the-fly and deallocates memory for the decompressed value immediately after it is consumed. Therefore, OMC only keeps the compressed parameters and a small number of transient decompressed parameters in memory, which uses less memory than storing the entire set of original, uncompressed parameters.

The main design challenge of OMC is achieving a favorable accuracy-efficiency trade-off. An important characteristic of OMC is that compression and decompression occur in every training iteration. As a result, the error introduced by compression can accumulate very quickly and degrade model accuracy significantly. On the other hand, we cannot use a very complicated algorithm to control the accumulated error because this will significantly slow down training. Therefore, OMC needs to be as simple and fast as possible and has a minimal impact on accuracy. It achieves this goal by using quantization, per-variable transformation, weight-matrix-only quantization, and partial variable quantization.

The following summarizes the benefits of OMC:

- **Reducing memory usage**: OMC reduces the memory usage of model parameters.
- **Reducing communication cost**: Because models are transported between servers and clients, reducing the parameter size helps reduce communication cost.
- **Lightweight operation**: OMC does not significantly slow down the training process even though compression and decompression occur frequently.

2. RELATED WORKS
Most of the related works in the literature that improve inference or training efficiency focus on centralized training. One widely adopted approach is reducing the complexity of models, such as manual design [7, 8], pruning [9, 10], or neural architecture search [11, 12]. However, reducing complexity typically limits the potential of the model for continuous improvement over growing data. Model transport compression [13] and gradient transport compression [14] keep the model unchanged and compress the transported data to save the communication cost but with the same memory usage.

Similar to OMC with quantization, Quantization-Aware Training (QAT) [4, 5, 15] also quantizes parameters. The main difference is that OMC aims to reduce memory usage during training while QAT focuses on saving memory during inference at the cost of higher memory usage during training. When training a model, QAT stores parameters in FP32 (32-bit single-precision floating-point format) and quantizes them on demand while OMC stores parameters in a compressed format and decompresses them on demand. Storing parame-
ters in FP32 allows QAT to precisely accumulate small gradients to achieve lower bitwidths for inference. However, it is at the cost of higher memory usage of parameters during training due to the extra transient quantized parameters. In contrast, storing parameters in a compressed format enables OMC to reduce the memory usage of parameters during training but makes it more challenging to control the quantization error and reduce bitwidths. This work proposes many methods to address it.

There are a few works aiming to improve the efficiency of federated learning. Federated dropout [16, 17] trains only part of the server model on clients, so that the server model can be much larger than client models. However, because the client models differ from the server model, federated dropout needs to maintain a mapping between them. Similar to federate dropout, group knowledge transfer [18] also uses different models on the server and clients. The clients run a small feature extractors to extract features, which are then used to train the server model. This approach decreases client loading at the cost of increased server loading. Partial vari-

3. METHODOLOGY

3.1. Framework of Online Model Compression

Fig. 1 illustrates the framework of the proposed Online Model Compression (OMC). OMC stores parameters in a compressed format, such as floating-point numbers with reduced bitwidths, but performs computations in uncompressed or other hardware-supported formats. This design decouples compression formats and hardware-supported formats to provide higher flexibility for choosing the compression format and method to achieve better memory usage reduction.

When performing forward propagation for a layer (the blue path in Fig. 1), OMC decompresses the required parameters for that layer on the fly and deallocates the decompressed copies immediately after they are consumed. When performing backward propagation for a layer (the red path in Fig. 1), OMC decompresses the required parameters and applies the gradients to update them. The updated decompressed parameters are then compressed and discarded immediately. Therefore, OMC only keeps the compressed parameters and a small number of transient decompressed copies in memory.

3.2. Quantization-Based Online Model Compression

Given the simplicity of quantization, we adopt it as the compression method in this paper. Quantization reduces the number of bits (i.e., bitwidth) for representing a value. While full precision (32 bits) is usually used in deep learning, many works in the literature have shown that neural networks are error-resilient and allow using much lower bitwidths without harming prediction accuracy [5]. However, such low bitwidths are usually achieved for inference. Reducing memory usage by quantization during training is more difficult because training requires more bits to precisely accumulate the small gradients across training iterations.

OMC adopts the floating-point format in this paper as an example although other formats, such as the fixed-point format, can also be used. The floating-point format consists of three parts: the sign bit, the exponent bits, and the mantissa bits. For example, the format of FP32 is composed of 1-bit sign, 8-bit exponent, and 23-bit mantissa. To quantize a floating-point value, we reduce the numbers of bits for the exponent and the mantissa, which are the two hyper-parameters of floating-point quantization.

3.3. Per-Variable Transformation

Quantization is a lossy operation and, thus, introduces quantization errors. As a result, quantizing parameters every training iteration can lead to a large accumulated error and prevent us from using fewer bits with the original accuracy maintained. To minimize the quantization error, OMC applies a linear transformation on the decompressed parameters, which is illustrated in Fig. 2. This step is performed per variable, such as per weight matrix, so that all the model parameters in a variable can share a few transformation-related parameters to make the memory overhead negligible.

The transformed variable (vector or flattened tensor) \( \hat{V} \in \mathbb{R}^n \) can be written as \( \hat{V} = s \hat{V} + b \), where \( \hat{V} \in \mathbb{R}^n \) denotes the decompressed variable, \( \mathbf{1} \in \mathbb{R}^n \) is a one vector, and \( s \) and \( b \) denote the per-variable scaling factor and bias, respectively. OMC determines the scaling factor and the bias analytically by minimizing the \( \ell^2 \)-norm of the difference between the decompressed-and-transformed variable \( (\hat{V}) \) and the full-precision variable before compression \( (V \in \mathbb{R}^n) \).

The closed-form solutions are

\[
s = \frac{n \sum_k V_k \hat{V}_k - \sum_k \hat{V}_k \sum_k V_k}{n \sum_k V_k^2 - (\sum_k V_k)^2}
\]

and

\[
b = \frac{n \sum_k V_k - s \sum_k \hat{V}_k}{n}
\]

where \( V_k \) denotes the \( k \)-th element of \( V \). Note that the degenerated case is when the denominator of \( s \) is 0, which happens (only) when all elements in each of \( V \) and \( \hat{V} \) are the same according to inequality of arithmetic and geometric means. We set \( s \) to 1.0 for this degenerated case.
Layer N → V̅Ṽ
Decompress → V
Transform → Consume
V
Compress

Fig. 2. The dependency graph of per-variable transformation. The cubes with dashed borderlines are transient variables.

In our implementation, s and b are computed in the 64-bit floating-point precision, but the final s and b are still stored as FP32 values.

3.4. Weight-Matrix-Only Quantization

We empirically found that some types of parameters are more sensitive to quantization than the others. These sensitive parameters include the scaling factors and biases in normalization layers. In contrast, weight matrices in convolutional and feed-forward layers are less sensitive to quantization but dominate the model size. For example, the weight matrices in the streaming Conformer model we use in Sec. 4 accounts for 99.8% of the model size. Hence, OMC only quantizes weight matrices and keeps the remaining variables in FP32. This method helps maintain accuracy while saving memory.

3.5. Partial Variable Quantization

OMC also leverages the feature of federated learning that there are many clients training a model in parallel to further reduce quantization errors. This feature provides an opportunity to quantize only a subset of variables for each client and vary the selection from one client to another. As a result, the server can receive high-quality and precise update of each variable from the clients that do not quantize this variable.

4. EXPERIMENTAL RESULTS

4.1. Experimental Settings

In this section, we validate and demonstrate the effectiveness of OMC across various use cases, including small and large datasets, IID and non-IID data distributions, streaming and non-streaming network architectures, and from-scratch and warm-start training settings.

The first dataset is the LibriSpeech dataset [20]. By partitioning LibriSpeech in two different ways, we derive the IID LibriSpeech and the Non-IID LibriSpeech dataset from the original LibriSpeech dataset to simulate different client data distributions. IID LibriSpeech is generated by random partition while Non-IID LibriSpeech is generated by partitioning by speakers. For LibriSpeech related experiments, the models are trained from scratch. The Word Error Rates (WERs) are reported in the format of dev/dev-other/test/test-other, where each item corresponds to the WER of the dev, dev-other, test, and test-other set from left to right.

The second dataset is an anonymized Multi-Domain (MD) dataset and much larger than LibriSpeech. The MD dataset contains around 400K hours of utterances from domains such as YouTube, Farfield, Search, and Telephony [21, 22]. We partition this dataset into the Medium Form (MF) domain dataset and the Non-MF domain dataset. These two partitions will be used to evaluate OMC under the domain adaptation scenario (from Non-MF domain to MF domain). For MD related experiments, a model will be first trained on the Non-MF domain dataset and then finetuned on the MF domain dataset. The WERs are reported on a disjoint test set from the MF domain.

We also experiment with two ASR models to evaluate OMC under non-streaming and streaming use cases. The first model is similar to the largest Conformer in the paper [3]. The only difference is that we replace batch normalization by group normalization, which is more suitable for federated learning with a small degradation in accuracy [23]. We refer to this model as Non-Streaming Conformer. The second model is our production-grade Conformer variant [24], containing around 130M trainable parameters, supporting streaming use cases, and referred to as streaming Conformer.

Unless otherwise specified, we randomly quantize 90% of the weight matrices and vary the selection from round to round and from client to client. There are 128 clients, and each client trains a model with one local step. The batch size is 16 per client. For resource consumption, we report the theoretical memory usage of parameters, the communication cost, and the training speed on TPUs. The memory saving observed in practice with our implementation is also provided in Sec. 4.4. We use SxEyMz to represent a floating-point format with x sign bits, y exponent bits and z mantissa bits. For example, the FP32 format is represented by S1E8M23.

4.2. Non-Streaming Conformer on LibriSpeech

Table 1 summarizes the results of Non-Streaming Conformer on IID LibriSpeech. Compared with FP32 (S1E8M32), OMC can achieve similar WERs with 64% memory usage of parameters and communication cost by using the 19-bit S1E4M14 format. OMC is also pretty lightweight. In this experiment, OMC only decreases the speed by 9%.

Table 3 summarizes the WERs of Non-Streaming Conformer on Non-IID LibriSpeech with the same bitwidth as that of the IID LibriSpeech experiment. Even with non-IID data, OMC can still attain comparable WERs to using FP32. The reduction in memory usage of parameters and communication cost is the same as the previous IID experiment and, hence, omitted in Table 3. These experiments show the versatility of OMC with both IID and non-IID data distribution.

4.3. Streaming Conformer on Multi-Domain Dataset

We observe that domain adaptation may allow using a smaller bitwidth than from-scratch training. Table 2 summarizes the results of Streaming Conformer on the Multi-Domain dataset. Compared to FP32, OMC can achieve similar WERs with 41% memory usage of parameters and communication cost by using the 11-bit S1E3M7 format. We can further reduce the bitwidth to 6 bits (S1E2M3) and still improve the WERs over the before-adaptation baseline. Moreover, OMC has a negligible impact on the training speed, by 7% in this case.

4.4. Measured Memory Usage on Pixel 4 Phones

We measured the memory usage on Google Pixel 4 with parameters quantized to FP16 (S1E5M10), which achieves the
same WERs as the FP32 baseline. We implemented federated learning with Tensorflow Federated [25] and applied gradient recomputation [26] to force releasing the memory occupied by transient parameters. The code has been uploaded to the Lingvo [27, 28] repository on Github. For the Streaming Conformer, OMC reduces the peak memory usage by 197 MB (38% of the model size). For a smaller Streaming Conformer model with 3 Conformer blocks in the encoder, OMC reduces the peak memory usage by 84 MB (45% of the model size).

4.5. Ablation Study

In the ablation study, we use Streaming Conformer on the Multi-Domain dataset unless otherwise specified.

4.5.1. Impact of Proposed Methods

We start from studying the impact of each of the proposed methods on WERs. The results are summarized in Table 4. After we quantize the parameters to 11 bits (S1E3M7), the WER significantly increases by 2.3. The WER gap is first closed by the proposed per-variable transformation, which reduces the WER by 0.4. Then, the proposed weight-matrix-only quantization reduces the WER by another 1.8. Finally, the proposed partial variable quantization brings down the WER to 4.6, which matches the FP32 baseline.

4.5.2. With and Without Partial Variable Quantization

In the case of quantizing 90% variables with the 11-bit format (S1E3M7), keeping the remaining 10% unquantized increases the average bitwidth by around 2 bits. In this study, we compare this 11-bit format with 90% variables quantized to various 13-bit formats with all variables quantized. We create these 13-bit formats by allocating the extra 2 bits to the exponent and mantissa parts in different ways. These 13-bit formats are S1E3M9, S1E4M8, and S1E5M7. Fig. 3 summarizes the training outcomes. We observe that using the proposed partial variable quantization with 11 bits results in faster convergence than all variable quantization with 13 bits. Moreover, none of these 13-bit formats can achieve a WER as low as that of partial variable quantization with 11 bits.

5. CONCLUSION

In this paper, we proposed Online Model Compression to reduce memory usage and communication cost of model parameters for federated learning. Our realization of OMC consists of floating-point quantization, per-variable transformation, weight-matrix-only quantization, and partial variable quantization. The experiments show that OMC is lightweight and can effectively maintain accuracy with significant efficiency improvement.

6. ACKNOWLEDGMENTS

We thank Petr Zadrazil for inspiring this project, Dhruv Giuliani for reviewing this paper, and Sean Augenstein, Zachary Garrett and Hubert Eichner for discussions on experiments.
7. REFERENCES

[1] P. Kairouz, H. Brendan McMahan et al., “Advances and open problems in federated learning,” arXiv preprint arXiv:1912.04977, 2019.

[2] J. Wang, Z. Charles et al., “A field guide to federated optimization,” arXiv preprint arXiv:2107.06917, 2021.

[3] A. Gulati, J. Qin et al., “Conformer: Convolution-augmented transformer for speech recognition,” in Conference of the International Speech Communication Association (INTERSPEECH), 2020.

[4] M. Rastegari, V. Ordonez et al., “Xnor-net: Imagenet classification using binary convolutional neural networks,” in European Conference on Computer Vision (ECCV), 2016.

[5] A. Abdolrashidi, L. Wang et al., “Pareto-optimal quantized resnet is mostly 4-bit,” arXiv preprint arXiv:2105.03536, 2021.

[6] J. Huang, F. Qian et al., “An in-depth study of lte: Effect of network protocol and application behavior on performance,” in Proceedings of the ACM SIGCOMM Conference on SIGCOMM, 2013.

[7] A. G. Howard, M. Zhu et al., “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” arXiv preprint arXiv:1704.04861, 2017.

[8] M. Sandler, A. Howard et al., “Mobilenetv2: Inverted residuals and linear bottlenecks,” in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[9] S. Han, H. Mao et al., “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” in International Conference on Learning Representations (ICLR), 2016.

[10] T.-J. Yang, Y.-H. Chen et al., “Designing energy-efficient convolutional neural networks using energy-aware pruning,” in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[11] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” in International Conference on Learning Representations (ICLR), 2017.

[12] T.-J. Yang, Y.-L. Liao et al., “Netadaptv2: Efficient neural architecture search with fast super-network training and architecture optimization,” in Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

[13] S. Chraibi, A. Khaled et al., “Distributed fixed point methods with compressed iterates,” arXiv preprint arXiv:1912.09925, 2019.

[14] J. Konečný, H. B. McMahan et al., “Federated learning: Strategies for improving communication efficiency,” in Advances in Neural Information Processing Systems (NeurIPS) Workshop on Private Multi-Party Machine Learning, 2016.

[15] S. Ding, P. Meadowlark et al., “4-bit conformer with native quantization aware training for speech recognition,” in Conference of the International Speech Communication Association (INTERSPEECH), 2022.

[16] S. Caldas, J. Konečný et al., “Expanding the reach of federated learning by reducing client resource requirements,” arXiv preprint arXiv:1812.07210, 2018.

[17] D. Guliani, L. Zhou et al., “Enabling on-device training of speech recognition models with federated dropout,” in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2022.

[18] C. He, M. Annavaram et al., “Group knowledge transfer: Federated learning of large cnns at the edge,” in Advances in Neural Information Processing Systems (NeurIPS), 2020.

[19] T.-J. Yang, D. Guliani et al., “Partial variable training for efficient on-device federated learning,” in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2022.

[20] V. Panayotov, G. Chen et al., “Librispeech: An asr corpus based on public domain audio books,” in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015.

[21] A. Narayanan, R. Prabhavalkar et al., “Recognizing long-form speech using streaming end-to-end models,” in IEEE Automatic Speech Recognition and Understanding Workshop, 2019.

[22] A. Misra, D. Hwang et al., “A Comparison of Supervised and Unsupervised Pre-Training of End-to-End Models,” in Conference of the International Speech Communication Association (INTERSPEECH), 2021.

[23] K. Hsieh, A. Panishayee et al., “The non-iid data quagmire of decentralized machine learning,” arXiv preprint arXiv:1910.00189, 2019.

[24] B. Li, A. Gulati et al., “A better and faster end-to-end model for streaming asr,” in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2021.

[25] https://www.tensorflow.org/federated.

[26] T. Chen, B. Xu et al., “Training deep nets with sublinear memory cost,” arXiv preprint arXiv:1604.06174, 2016.

[27] J. Shen, P. Nguyen et al., “Lingvo: a modular and scalable framework for sequence-to-sequence modeling,” CoRR, vol.abs/1902.08295, 2019.

[28] https://github.com/tensorflow/lingvo.