Distributed Model Training With Dynamic Gradient Compression

Yashwanth Kumar Tirupati
Arizona State University
Tempe, USA
ytirupat@asu.edu

Raghavendra Rao Suresh
Arizona State University
Tempe, USA
rsures21@asu.edu

Akschat Arya
Arizona State University
Tempe, USA
aarya9@asu.edu

***Abstract*—Training machine learning models on large datasets with high-dimensional data often demands significant computational resources. Sometimes, these requirements exceed the capabilities of a single device. To overcome this, distributed training is employed, where multiple devices work together over a network to pool their resources for efficient model training. However, this approach can lead to network congestion, especially when transferring large gradient information, which can reach terabytes in size. This congestion can slow down training, making distributed training less effective. One solution to this is gradient compression, but it has its own challenges. To address these issues, we propose a new technique called dynamic gradient compression. This technique adjusts gradient compression during training based on various factors, improving the efficiency of distributed training and enabling effective training of machine learning models on large datasets.**

I. Introduction and Problem Statement

In the realm of modern technology, machine learning models have evolved into intricate systems that demand extensive data and computational resources for training. These models, with their ability to learn from data, have revolutionized various industries, from healthcare to finance, by enabling intelligent decision-making and predictive analytics.

The process of training machine learning models involves feeding them with vast amounts of data to enable them to draw patterns and make inferences based on that data. This training process can be computationally intensive, especially when dealing with large datasets.

To effectively handle such data, machine learning models need considerable hardware resources for computation. This includes computing power for tasks like matrix multiplications, which are common in loss calculation algorithms. Moreover, the model adjusts its weights iteratively to minimize the loss function. This adjustment usually occurs through optimization algorithms like gradient descent, which compute gradients of the loss function with respect to the model's weights. These gradients are then used to update the weights to minimize the loss. Computing these gradients involves performing backpropogation which is computationally intensive.

To tackle the intensive computational requirements inherent in training machine learning models, sophisticated methods such as distributed training have become indispensable. Distributed training leverages the collective power of multiple devices operating in parallel, significantly enhancing the efficiency of model training. Multiple interconnected devices exchange gradients and synchronize their advancements throughout the training process. While distributed training admirably reduces training time and eliminates the need for a single, overpowering machine, it does introduce the potential hurdle of network congestion. This bottleneck arises from the substantial volume of gradients exchanged between the devices, posing a significant challenge within the network infrastructure.

Gradient compression techniques can be used to address this challenge, enabling models to train effectively with compressed gradient representations. These techniques streamline communication among training devices by transmitting compressed information, thereby reducing the required network bandwidth. Nevertheless, the adoption of any compression method entails inherent drawbacks, including information loss and increased computational overhead. Consequently, it might be beneficial to use gradient compression selectively during specific phases of the training process to maximize its benefits while also mitigating its limitations.

In this report we explore how the model performs when we use our approach of dynamically altering the level of compression for the gradients being shared between devices in the network for training. Our effective goal is to help increase accuracy while trying to reap the benefits of gradient compression for certain periods of the training phase.

II. Prior Works

Gradient compression has become a crucial technique for speeding up distributed machine learning training by reducing communication costs. This section reviews existing works relevant to our proposed approach of dynamic gradient compression with an adaptive strategy.

A. Gradient Sparsification

[1] Zhou et al. (2021): Effective Sparsification of Neural Networks with Global Sparity Constraint: This work proposes two sparsification techniques: magnitude-based and random. Magnitude-based sparsification transmits only a user-defined percentage of the largest gradient elements, achieving significant communication efficiency. Random sparsification randomly selects a subset of elements for transmission. Both techniques achieve good communication efficiency while main-

taining model accuracy. However, their approach focuses on a global sparsity constraint, meaning the same sparsity ratio is applied throughout the training process. This might not be optimal as gradient magnitudes typically change during training. In the initial stages, where updates are large, a higher sparsity ratio might be beneficial.

B. Gradient Compression via Quantization

[3] Lin et al. (2017): Reducing the Communication Bandwidth for Distributed Training: This work introduced Deep Gradient Compression (DGC), a pioneering approach for gradient compression using quantization. DGC quantizes the gradients to a lower precision format (e.g., fewer bits) before transmission, significantly reducing communication costs. While effective, DGC uses a fixed compression ratio throughout the training. This might be suboptimal as gradients in the initial stages tend to have larger magnitudes, allowing for more aggressive compression without sacrificing accuracy. Additionally, DGC might require careful selection of the quantization scheme to balance communication efficiency and model convergence.

[7] Alistarh et al. (2017): Communication-Efficient SGD via Gradient Quantization and Encoding: Similar to DGC, this work proposes QSGD, which utilizes gradient quantization and encoding for communication efficiency. QSGD employs a stochastic quantization scheme, where each gradient element is independently quantized with a certain probability. This approach can introduce noise into the gradients, but with careful selection of the quantization scheme and noise injection strategy, it can achieve good communication-accuracy trade-offs. However, like DGC, QSGD utilizes a fixed compression ratio throughout training, potentially overlooking the opportunity for more aggressive compression in the initial phases.

C. Distributed Training with Communication Efficiency

[2] Horvath et al. (2022): Natural Compression for Distributed Deep Learning: This work explores "natural compression" for distributed deep learning, leveraging inherent redundancy in gradients to reduce communication costs. They observe that gradients often exhibit redundancy, with many elements having similar values. Their approach exploits this redundancy by transmitting only unique gradient elements and their corresponding counts. While offering communication efficiency, this method might not be suitable for scenarios where aggressive compression in specific training phases is desired. Additionally, it might not be effective for all gradient distributions, potentially leading to situations where the communication cost reduction is not significant.

D. Adaptive Techniques for Distributed Training

[5] Hardy et al. (2017): Distributed Deep Learning on Edge-Devices: This work, similar to our approach, proposes an adaptive compression scheme for distributed deep learning. However, their work targets resource-constrained edge devices, where communication bandwidth is a critical concern. They dynamically adjust the compression ratio based on the available network bandwidth. While their approach demonstrates communication efficiency benefits, it might require modifications for broader distributed training settings, which might have different resource constraints compared to edge devices.

E. Related Work in Distributed Optimization

[6] Ringström et al. (2018): Communication-efficient Federated Averaging for Distributed Deep Learning: This work explores federated averaging, a distributed training approach where local models are updated on devices and only model updates are exchanged, reducing communication compared to synchronous training. While federated learning offers communication benefits, it might not be suitable for all scenarios compared to our focus on gradient compression in synchronous training. Additionally, federated learning might introduce challenges like non-IID (independent and identically distributed) data distributions across devices, which can impact convergence.

[8] Stich et al. (2018): Pipelined Asynchronous Distributed Training: This work provides an overview of pipelined asynchronous distributed training, a technique that improves communication efficiency by overlapping communication and computation. However, it does not delve into gradient compression strategies, which is the focus of our work.

[9] Zhang et al. (2018): Adaptive Batch Sizes for Deep Learning: This work explores adaptive batch sizes for deep learning. While not directly related to gradient compression, using larger batch sizes in the initial stages of training can lead to faster convergence but can also increase the communication cost due to larger gradients.

III. Methodology

A. Data Parallelism

Distributed training using multiple GPUs necessitates the implementation of parallelism to distribute the workload across resources efficiently. Various techniques can be employed to achieve this, including data parallelism, tensor parallelism, and model parallelism. However, it's crucial to recognize that there is no universal solution, and the optimal approach depends on the specific hardware configuration and the specifications of the model being trained.

In this paper, we mainly deal with data parallelism, with a specific emphasis on synchronous data parallelism, wherein the model replicas remain synchronized after processing each batch. This synchronization ensures that the model's convergence behavior mirrors that of single-device training. However, our approach can also be easily expanded to encompass model parallelism and tensor parallelism.

Data parallelism involves replicating a single model across multiple devices or machines. Each device processes different batches of data, and their results are merged afterward. There are variations in how the model replicas merge their results and whether they remain synchronized at every batch or are more loosely coupled. In data parallelism, the current batch of data, called the global batch, is divided into sub-batches,

one for each GPU. Each GPU's model replica processes its assigned local batch independently, executing a forward pass followed by a backward pass to generate gradients. These local gradients are then efficiently integrated across GPUs through inter-GPU communication methods. Synchronization occurs at the end of each step to maintain consistency across replicas.

B. Gradient Synchronization

We utilize all reduce operation to achieve gradient synchronization across multiple GPUs. Conceptually, this algorithm involves each process or worker sharing its data with all others and then applying a reduction operation. This operation, such as sum, multiplication, max, or min, effectively condenses the target arrays across all processes or workers into a single array, which is then returned to each process. Standard all reduce is preferred for its simplicity and effectiveness, particularly in evaluating compression effects during experiments.

C. Gradient Compression

During training, gradient compression reduces the data exchanged between devices via inter-process communication during backpropagation. This reduction in communication bandwidth can improve training scalability and efficiency. Various techniques are available for implementing gradient compression.

1) Quantization: Quantization is a compression technique wherein the precision of gradient values is decreased by converting them into lower bit representations. Quantization methods have also shown promising results. In [11], a family of algorithms called QSGD was proposed, based on lossy gradient compression through quantization. These algorithms enabled training the ResNet-152 [12] network to full accuracy on ImageNet [13], achieving a speedup of 1.8× compared to variants with full-precision gradients.

2) Sparsification: Sparsification selectively transmits only the non-zero gradient values during communication between devices. This approach is particularly advantageous for models with sparse gradients, where a significant portion of the gradient values are zero. Sparsification techniques have been explored extensively in recent studies. For instance, one study [14] proposed sparsifying the gradient tensor by replacing values below a specified threshold with zeros, achieving a remarkable 99% sparsification during the training of a fully connected deep neural network on MNIST [15], with an impressive accuracy of 99.42%.

D. Proposed Approach

Traditionally, gradient compression techniques utilize a static compression ratio throughout the training cycle. Some existing approaches dynamically determine the data to be exchanged during training using complex pre-trained machine learning models. But these approached add significant complexity to the training process. Our proposed methodology introduces a simple approach of dynamic gradient compression in distributed training. It utilizes an adaptive compression strategy that evolves across training epochs, responding to shifts in the model's validation accuracy. Our goal is to optimize convergence speed and model accuracy while maintaining network bandwidth utilization comparable to standard compression models.

In the initial training phases, we employ aggressive gradient compression because fine-tuned gradient values may not be necessary. This is because the model is still learning basic patterns and features from the data, and the updates made during these phases are often large and general, so precise gradient values may not significantly impact the overall training process. However, as training progresses, excessive compression can lead to loss of vital information needed for fine-tuning. To mitigate this risk, we gradually reduce compression, aligning with the evolving needs of the model. This approach allows for precise adjustments and fine-tuning while maintaining efficiency and speeding up convergence rates.

Compression levels are tailored according to key convergence metrics such as validation accuracy and the difference in cross-entropy training loss. Employing a step function-based approach, we systematically lower compression levels based on predefined thresholds, which serve as tuning parameters for this algorithm. Additionally, to address potential stagnation in model learning, we incorporate a mechanism to temporarily remove compression if the model remains trapped at the same step of the compression step function for an extended period across multiple epochs.

IV. Experimentation

A. Experiment Configuration

1) Hardware Configuration: The experimentation utilized the computational power of two NVIDIA T4 GPUs. The NVIDIA T4 GPUs are high-performance graphics processing units designed for accelerated computing tasks, including deep learning and artificial intelligence. They are known for their efficiency in processing complex computations, making them ideal for training machine learning models.

We used the PyTorch Distributed Data Parallel (DDP) module to coordinate and manage the GPUs in our program. This allowed them to work together simultaneously, speeding up the processing and training of our machine learning model. By leveraging the capabilities of these GPUs and the DDP module, we achieved significant improvements in training speed.

2) Machine Learning Model Architecture: ResNet stands out as a prominent deep learning architecture renowned for its ability to tackle the challenges of training extremely deep neural networks. Leveraging residual connections, ResNet mitigates the vanishing gradient problem, thereby facilitating the training of deep networks effectively. ResNet architectures have consistently demonstrated strong performance across a wide range of tasks and datasets. This reliability makes them a good starting point for exploring new methodologies or techniques. Also, ResNet architectures can be easily adapted to different input sizes and domains, allowing for experimen-

tation across various datasets and problem domains without significant architectural modifications.

3) Dataset Used: We have used the CIFAR-10 dataset for our experiment. It is a widely recognized benchmark dataset for image classification tasks within the realms of machine learning. It comprises 60,000 32x32 color images distributed across 10 distinct classes, each containing 6,000 images. The dataset is partitioned into 50,000 training images and 10,000 test images, facilitating robust model evaluation.

B. Experiment Setup

For our experiment, we designed three distinct training scenarios to assess the effectiveness of our proposed methodology against existing approaches. These scenarios were carefully selected to represent different aspects of the training process and to provide a comprehensive evaluation of our method's performance. By comparing the results generated from these scenarios, we aimed to gain insights into the strengths and limitations of our proposed methodology compared to existing methodologies.

1) Standard Training: In the initial scenario, known as "standard training", the machine learning model undergoes training without the application of any compression techniques.

2) Static Quantization Training: The second training scenario, termed "Static Quantization", involved training the machine learning model while applying a constant 8-bit gradient quantization method throughout the training process.

3) Dynamic Compression Training: Lastly, the third training scenario, termed "Dynamic Compression," implemented our proposed methodology. In this scenario, the machine learning model underwent training with a dynamically adjusted compression rate for gradients, ranging from 4-bit to 16-bit quantization.

V. Results

In Figure 1, we note that employing the static quantization compression method, fixed at an 8-bit quantisation, usually hits a saturation point at around 80-85% accuracy for training data. Beyond this threshold, weight updates slow down considerably, indicating minimal improvement despite ongoing training efforts. Conversely, dynamic compression methods exhibit initially slower training progress due to high compression in the early stages. Nevertheless, as training advances, it swiftly converges and approaches the performance level of models trained without compression.

This pattern echoes in the test accuracy, as depicted in Figure 2. The static compression method typically plateaus at approximately 80-85%, whereas the proposed dynamic compression approach achieves accuracy nearing 91%. Similarly, a comparable trend emerges in the cross-entropy training loss, as illustrated in Figure 3.

Additionally, when comparing the training durations of all methods, we observed that both static and dynamic compression approaches showed similar training times, with the dynamic approach being slightly faster. However, it's important

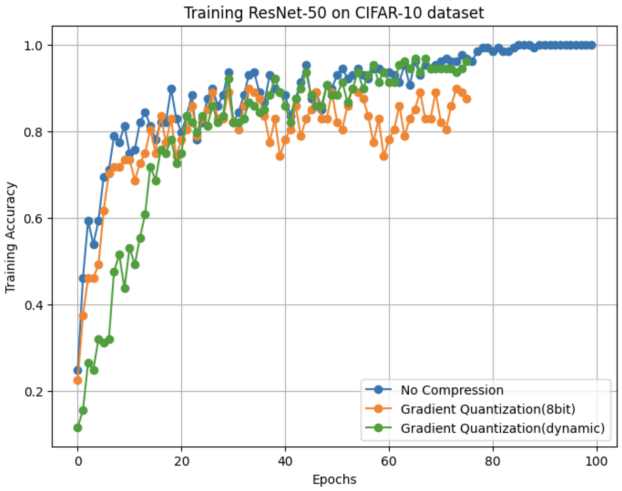

Fig. 1: Training Accuracy vs Epochs

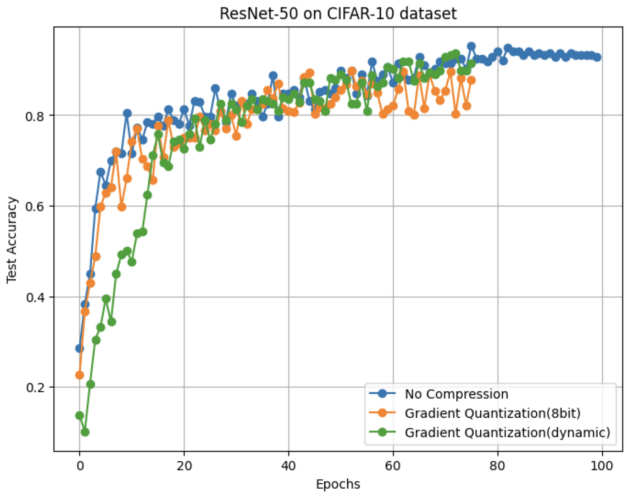

Fig. 2: Test Accuracy vs Epochs

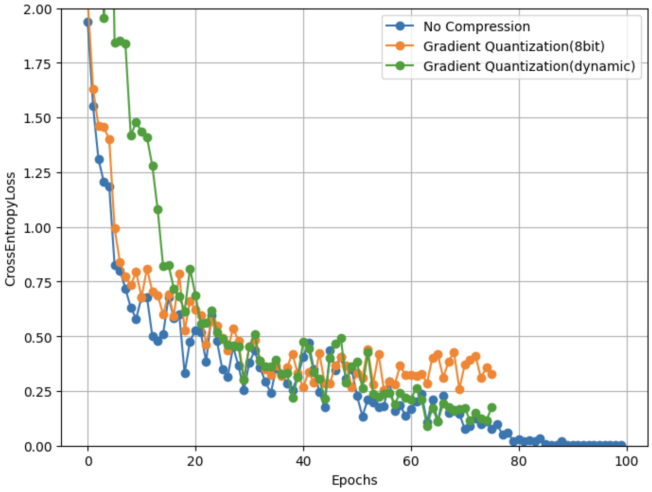

Fig. 3: Cross Entropy Loss vs Epochs

TABLE I: Performance Evaluation

Compression Method	Num Epochs	Training Accuracy	Test Accuracy	Training Time(min)	Data Transferred(TB)
No Compression	70	96.09	92.96	65.12	2.31
Static Compression	70	82.03	85.97	122.13	1.82
Dynamic Compression	70	94.53	91.40	101.64	2.03

to highlight that both these methods entail longer training times compared to the model trained without any compression. This outcome aligns with our expectations, as compression introduces additional computational complexity in each training step. Given that our project's primary objective is to improve the static quantization training method's convergence speed, this is an area we can investigate further in future endeavors.

As depicted in Table 1, we conducted measurements on the number of bits exchanged between the GPUs during gradient synchronization using allreduce. To evaluate this metric, we computed the sizes of the compressed gradient tensors exchanged among the GPUs. Upon analyzing this data, we noticed that the models employing static and dynamic compression exhibit reduced data communication compared to the uncompressed model. Although the dynamic approach involves slightly higher data transmission, it is still better than the no compression method.

VI. Conclusion and Future Work

In conclusion, adjusting the compression level of gradients based on various training factors proves to be generally effective in a distributed training environment. While it results in higher network bandwidth utilization than the static compression method and longer training times than the no compression method, it combines the benefits of both methods, achieving faster convergence speed and maintaining respectable model accuracy. This methodology provides a flexible and efficient way to handle network congestion during training for distributed training architectures, while achieving a decent model accuracy.

Future work should focus on exploring a broader range of scenarios to validate the robustness and scalability of the proposed approach. This includes testing the methodology across different machine learning models and architectures to understand its impact on a wider variety of training setups. Additionally, scaling to a larger number of GPUs can provide insight into the approach's efficiency and effectiveness in high-performance distributed training environments.

Furthermore research should also examine the application of alternative compression techniques and their potential to improve training outcomes. This includes investigating different algorithms for determining compression levels on their impact on training time, model convergence, and accuracy. By evaluating the approach in diverse and challenging settings, we can gain a deeper understanding of its strengths and limitations, ultimately guiding the development of more advanced and efficient distributed training techniques in the future.

References

[1] Zhou, Xiao, et al. "Effective sparsification of neural networks with global sparsity constraint." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021.

[2] Horvóth, Samuel, et al. "Natural compression for distributed deep learning." Mathematical and Scientific Machine Learning. PMLR, 2022.

[3] Lin, Yujun, et al. "Deep gradient compression: Reducing the communication bandwidth for distributed training." arXiv preprint arXiv:1712.01887 (2017).

[4] Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149 (2015).

[5] Hardy, Corentin, Erwan Le Merrer, and Bruno Sericola. "Distributed deep learning on edge-devices: feasibility via adaptive compression." 2017 IEEE 16th international symposium on network computing and applications (NCA). IEEE, 2017. Here are the references formatted in LaTeX code and listed starting with

[6] Ringström, P., et al. "Communication-efficient distributed deep learning with federated averaging." *arXiv preprint arXiv:1802.07688* (2018).

[7] Alistarh, D., et al. "QSGD: Communication-efficient SGD via gradient quantization and encoding." *Advances in Neural Information Processing Systems*, vol. 31, pp. 1709-1720, 2017.

[8] Stich, A., et al. "Pipelined asynchronous distributed training: An overview." *arXiv preprint arXiv:1805.07821* (2018).

[9] Zhang, H., et al. "Adaptive batch sizes for deep learning." *arXiv preprint arXiv:1804.06124* (2018).

[10] De Sa, C., et al. "High-performance distributed machine learning with elastic averaging." *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2014).

[11] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in Advances in Neural Information Processing Systems (NIPS), 2017, pp. s 1707–1718.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei., "ImageNet: A large-scale hierarchical image database," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009.

[14] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in Empirical Methods in Natural Language Processing (EMNLP), 2017.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.