Investigating Layer-wise Masking and Dynamic Step Selection for Distributed Learning

Ali Abbasi
Politecnico di Torino
Turin, Italy
ali.abbasi@studenti.polito.it

Mohammadmahdi Gheisari
Politecnico di Torino
Turin, Italy
m.gheisari@studenti.polito.it

Fatemeh Ahmadpour
Politecnico di Torino
Turin, Italy
fatemeh.ahmadpour@studenti.polito.it

Abstract

Distributed learning has become essential for scaling computation across multiple machines as datasets and machine learning models continue to grow. LocalSGD, a widely adopted distributed learning approach, has advanced the field by decreasing communication overhead during model aggregation; with the introduction of SlowMo, convergence speed has further improved. However, challenges such as device heterogeneity and determining the optimal number of local steps remain. This paper investigates a layer-wise masking approach inspired by Setayesh et al.(2023) [4] to address device heterogeneity by dynamically freezing model layers based on each device's computational capacity. Additionally, a confidence interval-based method is introduced to determine the optimal number of local steps, enabling efficient synchronization across workers. The code can be accessed via this link: https://github.com/aliarabbasi5155/mldl2024

1. Introduction

Deep learning models, particularly convolutional neural networks (CNNs), have achieved remarkable success in various tasks, but training these models efficiently on large datasets remains challenging due to the computational demands. Traditional methods like Stochastic Gradient Descent (SGD) are often limited by the use of small batch sizes, which restricts the scalability of training. To address these limitations, large batch algorithms were introduced to accelerate the training process by leveraging greater computational resources. However, these methods introduce new challenges, such as reduced generalization ability and convergence instability, especially when increasing the learning rate to accommodate larger batch sizes.

To overcome the optimization difficulties associated with large batch training, You et al.(2017) [7] proposed the Layer-wise Adaptive Rate Scaling (LARS) algorithm. LARS addresses these issues by dynamically adjusting the learning rate for each layer based on the ratio of the L2 norm of the weights to the gradients. This layer-wise adaptation allows for stable training with very large batch sizes—up to 32K—without decreasing model accuracy.

Building on the principles of LARS, the Layer-wise Adaptive Moments (LAMB) optimizer was introduced by You et al.(2019) [8] for large-scale training scenarios. Like LARS, LAMB adapts the learning rate for each layer, but it also incorporates adaptive moment estimation (Adam). This dual approach balances stability and convergence speed, making LAMB particularly effective for training models with very large batch sizes—more than 32K—without performance degradation.

While large batch methods like LARS and LAMB improve training efficiency, distributed approaches have become essential to scale deep learning models across multiple devices. In this context, the global batch is split into smaller subsets, which are processed independently across different workers. This reduces computation time but introduces new challenges, such as communication overhead due to synchronization between the workers.

To mitigate these issues, Lin et al.(2019) [3] developed LocalSGD as a more communication-efficient variant of SGD. LocalSGD allows workers to perform several local updates before synchronizing with other workers, thereby reducing the frequency of communication and computation time in distributed settings. However, the reduced commu-

nication can result in slower convergence and generalization.

To address the slow convergence and generalization problems in distributed training, particularly in LocalSGD, the SlowMo (Slow Momentum) algorithm was introduced by Wang et al.(2020) [6]. SlowMo improves the performance of LocalSGD by periodically averaging the parameters across devices while allowing slow momentum accumulation between synchronization steps. This approach maintains model consistency, and improves convergence rates. By balancing local computation and global synchronization, SlowMo achieves better optimization and generalization performance, making it suitable for large-scale machine learning tasks that require efficient distributed training.

A layer-wise masking approach, inspired by Setayesh et al.(2023) [4], is investigated in this paper to address the challenge of device heterogeneity as an alternative to asynchronous methods. In this approach, certain model layers are dynamically frozen based on the computational capacity of each device, results in optimizing resource utilization.

Additionally, a solution based on confidence intervals is introduced to determine an optimal number of local steps in distributed learning. This method allows each worker to decide whether to continue local training or synchronize with other workers based on the relational error of its training accuracies, thereby improving the overall efficiency and convergence of the distributed training process.

2. Related Work

Recent research in distributed deep learning has aimed to address the challenges of device heterogeneity. The Scale and Heterogeneity-aware Asynchronous distributed Training algorithm (SHAT) algorithm, proposed by Ko and Kim.(2022) [2], tackles these challenges by dynamically adjusting the update of each worker's local model based on both the scale of distributed training and the degree of heterogeneity among workers. SHAT uses an adaptive weighting factor to balance the contribution of local models to the global model, reducing differences that could delay convergence. Empirical evaluations show that SHAT improves accuracy and convergence rates over existing methods, demonstrating its robustness and efficiency in heterogeneous environments.

The PerFedMask algorithm, introduced by Setayesh et al.(2023) [4], provides an innovative approach to address this challenge by optimizing masking vectors for each device. This method dynamically determines which parts of the global model should be updated or frozen based on the computational capacity of each device, thereby reducing convergence bias without altering the overall model architecture. PerFedMask also improves personalization by fine-tuning local models with each device's data after the global model converges, ensuring better performance in heterogeneous environments. This approach demonstrates superior accuracy and efficiency compared to existing methods, making it well-suited for federated learning scenarios where both device and data heterogeneity are prevalent.

3. Methods

3.1. Layer-wise masking approach in LocalSGD

In distributed learning scenarios where device heterogeneity exists, i.e., where different devices have varying computational capabilities, it is crucial to adapt the training process to improve these differences without reducing overall model convergence. Inspired by the approach introduced by Setayesh et al.(2023) [4], which uses optimized masking vectors to handle device heterogeneity in federated learning, the proposed method extends this concept to distributed learning by freezing specific layers of the model on devices with limited computational power.

To implement this, a dynamic layer-wise freezing mechanism is introduced, in which a subset of model layers is frozen according to the computational capabilities of each participating device. This mechanism reduces the computational load on less capable devices by preventing updates to certain layers during the local training phase. The following steps outline the approach:

1. **Layer selection for freezing:** Before training begins, devices are profiled based on their computational resources (e.g., CPU/GPU computation, memory availability). For devices identified with lower capabilities, a set of layers—particularly those contributing less to the overall model's performance—is selected to be frozen. This selection process involves optimizing a masking vector that determines which layers remain trainable and which are frozen, thereby minimizing the potential biases introduced into the convergence process, as described by Setayesh et al.(2023) [4]

2. **Local update procedure with freezing:** During local training iterations, gradient updates are performed only on the unfrozen layers, while the frozen layers' parameters are kept unchanged. Let $m_n \in \{0,1\}^d$ denote the masking vector for device n, where each entry indicates whether a specific layer is frozen (0) or trainable (1). For device n, the local update for model parameters θ at iteration i is given by:

$$\theta^{(i+1)} = \theta^{(i)} - \eta\, m_n \odot \nabla f_n(\theta^{(i)}, b^{(i)}), \qquad (1)$$

where η is the learning rate, $b^{(i)}$ represents the mini-batch of local data at iteration i, and $\odot$ denotes the element-wise product.

3. **Global synchronization and fine-Tuning:** After completing a predefined number of local iterations, the unfrozen model parameters are synchronized across the workers. To mitigate any residual bias introduced by layer freezing, a fine-tuning step is performed on all devices using their local data, allowing updates to all layers without freezing constraints.

4. **Advantages of layer freezing:** By dynamically selecting which layers to freeze, the method adapts to varying device capabilities while ensuring efficient and stable convergence of the learning model. It has been empirically demonstrated that this approach effectively maintains robustness against device heterogeneity in distributed learning settings.

3.2. Confidence interval approach

Theoretical background: To dynamically determine the number of local steps before synchronizing the models in a distributed learning setting, a method based on the confidence interval estimation for the mean of training accuracy observed by each worker is utilized. The approach leverages statistical measures to make an informed decision on whether further local training should continue or synchronization should be performed.

Collection of training accuracies: At the end of each local step, each worker stores the training accuracy in a list denoted by:

$$A = [a_1, a_2, \ldots, a_n] \tag{2}$$

where n represents the number of local steps completed. Each element a_i represents the training accuracy after the i-th local step.

Computation of statistical measures: The mean of accuracies $\bar{a}$ is computed as:

$$\bar{a} = \frac{1}{n} \sum_{i=1}^{n} a_i \tag{3}$$

The standard deviation s of the accuracies is given by:

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (a_i - \bar{a})^2} \tag{4}$$

Determination of estimated confidence interval: A two-tailed t-distribution is employed to estimate the confidence interval of the mean of accuracies. The critical value $t_{\alpha/2,n-1}$ is obtained for a significance level α and degrees of freedom $n-1$. The estimated confidence interval CI is computed as:

$$CI = t_{\alpha/2,n-1} \cdot \frac{s}{\sqrt{n}} \tag{5}$$

Computation of relational error: The relational error E_r quantifies the deviation of the estimated confidence interval relative to the mean accuracy and is calculated as:

$$E_r = \frac{CI}{\bar{a}} \tag{6}$$

Determination of the accuracy threshold: A target accuracy T is predefined. The adjusted accuracy A_{adj} is derived as:

$$A_{\text{adj}} = 1 - E_r \tag{7}$$

If $T > A_{\text{adj}}$, additional local training is conducted; otherwise, model synchronization across workers is performed. Algorithm 1

4. Experiments

The experiments were designed to comprehensively assess the performance of different optimization methods, the effectiveness of large-batch and distributed training approaches, and to tackle challenges associated with heterogeneous device settings in a distributed training environment. The study focused on comparing the advanced optimizers LARS and LAMB with the baseline optimizers, SGDM and AdamW, across a range of configurations to evaluate their relative effectiveness. Further comparisons will explore the impact of varying batch sizes and learning rates on convergence stability and generalization for each optimizer, examine the trade-offs between communication frequency and model accuracy in distributed settings using LocalSGD and SlowMo, and investigate the performance benefits of dynamic step selection methods against fixed-step approaches. Additionally, the study will compare layer-wise masking with SHAT under a device heterogeneity scenario to determine the most efficient strategy for balancing computation load and synchronization costs while maintaining robust model performance across different distributed environments.

4.1. Datasets and model architecture

All experiments were conducted using the CIFAR-100 [1] dataset, a standard benchmark for image classification tasks. The model architecture is based on a modified LeNet5 design as introduced on [1], which included two convolutional layers with a kernel size of 5, zero padding,

[1]The CIFAR-100 dataset contains 60000 32x32 color images in 100 classes of 600 each. CIFAR-100 has 20 superclasses and 100 classes. The "fine" and "coarse" labels of each image indicate its class and superclass. There are 50000 training and 10000 test photos. [5]

and 64 output channels, each followed by max-pooling layers with a kernel size and stride of 2. This configuration was supplemented by three fully connected layers with 384, 192, and 100 output channels, respectively, applying ReLU activation functions to all layers except the final output layer. No activation function was used in the last layer to allow the cross-entropy loss to directly apply the softmax function internally, ensuring compatibility with the loss function and maintaining numerical stability. The learning rate was managed with a Cosine Annealing scheduler over 150 epochs. Data augmentation was applied uniformly across all experiments, consisting of random cropping to a size of 32 with a padding of 4 and padding mode set to 'reflect,' random horizontal flipping, and normalization.

Test accuracy is used as the primary metric to compare the performance of different approaches. In distributed training settings, the number of communications is also considered an important metric.

4.2. Centralized training and baseline model

The baseline model was developed using the SGDM and AdamW optimizers, with hyperparameters optimized through a grid search over learning rates of 0.001 and 0.01, and weight decays of 0.0001, 0.001, and 0.0004. The dataset was partitioned into a training set comprising 80% of the data and a validation set consisting of the remaining 20%. An early stopping criterion was implemented, terminating training if no improvement in validation accuracy was observed over 10 consecutive epochs. This approach aimed to save computational resources by skipping further training with ineffective hyperparameters. Among the combinations tested in Figure 1, the configuration with lr=0.01 and wd=0.001 was chosen as the optimal setting because it achieves a balance between convergence speed and stability, reaching the highest stable accuracy compared to other settings. This combination allows the SGDM optimizer to converge rapidly while maintaining test accuracy at a relatively high level. Similarly, based on Figure 2 The combination of lr=0.001 and wd=0.0004 was selected as the optimal parameter setting, as it effectively balances stability and convergence speed.

4.3. Large-batch training setting

To assess the impact of large-batch training, experiments were conducted using the LARS and LAMB optimizers, with hyperparameters carefully tuned for each optimizer, beginning with a base batch size of 64. For LARS, learning rates of 0.01, 0.05, 0.1, 0.5, 1, 1.5, and 2 were tested with a fixed weight decay of 0.001. Figure 3 illustrates the test accuracy for different learning rate (lr) values using the LARS optimizer. The configuration with lr=1.5 was chosen as the optimal setting because it provides a good balance between convergence speed and stability.

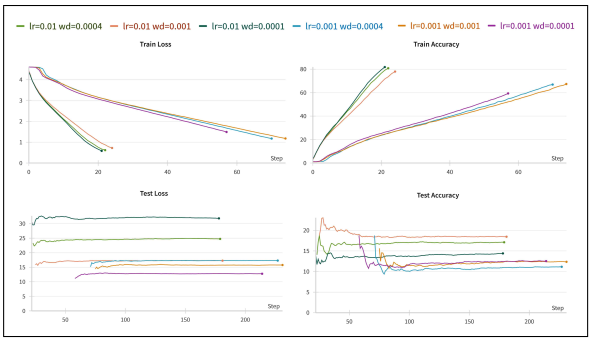

Figure 1. Train loss, train accuracy, test loss, and test accuracy over training steps for different learning rate (lr) and weight decay (wd) combinations using SGDM optimizer.

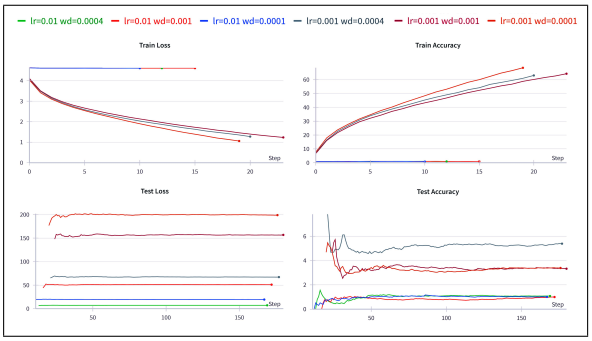

Figure 2. Train loss, train accuracy, test loss, and test accuracy over training steps for different learning rate (lr) and weight decay (wd) combinations using AdamW optimizer.

For LAMB, learning rates of 0.0009, 0.00095, 0.001, 0.0015, and 0.002 were evaluated with a fixed weight decay of 0.0004. Figure 4 shows the test accuracy for various learning rate (lr) values using the LAMB optimizer. The configuration with lr=0.0015 was chosen as the optimal setting because it achieves a stable convergence.

The learning rate for LARS was scaled using a square root method, whereas a linear scaling approach was employed for LAMB. These adaptive scaling strategies were implemented to dynamically adjust the learning rates in proportion to the batch size, thereby optimizing the training performance. Moreover, due to the limited memory capacity of the GPU, an accumulation step approach was employed for batch sizes greater than 4K allowing for effective large-batch training while remaining within hardware constraints.

4.4. Distributed training setting

LocalSGD: In the distributed training experiments using the LocalSGD optimizer, various configurations were tested to identify the optimal settings. The number of workers was varied among 2, 4, and 8, and local steps were tested at 4,

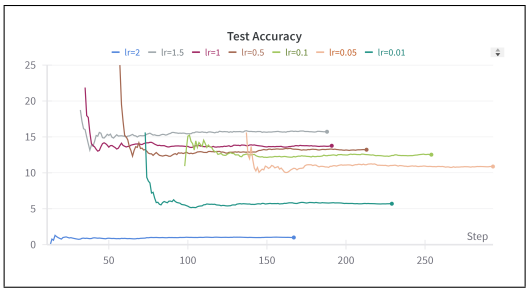

Figure 3. Test accuracy over training steps for different learning rate (lr) values using LARS optimizer. Configurations include lr=2, lr=1.5, lr=0.5, lr=0.1, lr=0.05, and lr=0.01.

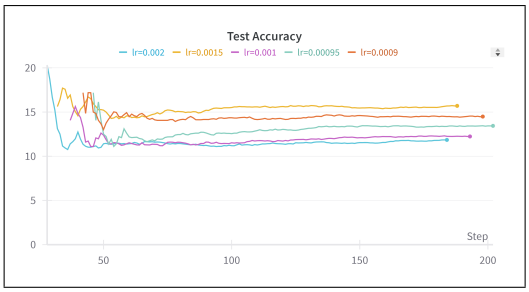

Figure 4. Test accuracy over training steps for different learning rate (lr) values using LAMB optimizer. Configurations include lr=0.002, lr=0.0015, lr=0.001, lr=0.00095, and lr=0.0009.

Test Accuracy Across Different Batch Sizes		
Batch Size	**LARS**	**LAMB**
64	56.63	53
512	56.18	47.49
1K	55.35	42.42
2k	54.15	36.42
4K	54.48	29.16
8K	Diverged	21.19
16K	Diverged	14.82
32K	Diverged	7.73
SGDM	*Batch size 64*	*57.55*
AdamW	*Batch size 64*	*49.46*

Table 1. Test accuracy of LARS and LAMB optimizers across different batch sizes, with a comparison to the baseline performance of SGDM and AdamW at a batch size of 64. The table illustrates the varying effectiveness of these optimizers in handling different batch sizes.

8, 16, 32, and 64 to evaluate their impact on test accuracy and communication efficiency. Based on Table 2, the optimal configuration selected was 2 workers with 4 local steps, achieving the highest test accuracy (56.1%) while maintaining a reasonable communication count (37). This setting provided a balance between model performance and communication overhead.

SlowMo: For the SlowMo optimizer, a similar set of experiments was conducted to determine the most effective configuration. The number of workers was also set to 2, 4, and 8, and local steps were varied across 4, 8, 16, 32, and 64. Based on Table 3, the optimal setting identified was 2 workers with 4 local steps, which achieved the highest test accuracy (56%) with a manageable communication count (37). This configuration demonstrated a favorable balance between accuracy and communication costs, indicating robustness to variations in local steps.

SHAT: In the SHAT algorithm experiment, a configuration with 2 workers was considered, using the same optimal parameter settings identified for LocalSGD to ensure a fair comparison with other methods. The second worker was assumed to have less computational power, reflecting a heterogeneous environment. Synchronization occurred the number of workers (k) times over the total number of iterations in the outer loop ($2 \times 150 = 300$ in this case). Each worker aggregating its local model with the global model after every local step. As presented in Table 5, the SHAT algorithm achieved a test accuracy of 42.86%.

Layer-wise masking: In the Layer-wise Masking experiment, a configuration with 2 workers was considered, following the optimal parameter settings identified for LocalSGD. The second worker was assumed to have less computational power. To address this, two layers that contributed less to the overall model performance—the first convolutional layer and the second fully connected layer—were chosen to be frozen. The model was trained with this layer-freezing strategy for 37 iterations, and in the final iteration, all layers were unfrozen to allow for fine-tuning.

Two experiments were conducted to assess the impact of this approach. In the first experiment, the model was trained without fine-tuning, achieving a test accuracy of 48.84% presented in Table 5. In the second experiment, fine-tuning was performed after unfreezing all layers, resulting in a slightly higher test accuracy of 49.05%. Table 5. These results suggest that fine-tuning after initial layer freezing provides a modest improvement in model performance.

Dynamic selection of number of local steps: In this approach, experiments were conducted using a dynamic selection method for the number of local steps based on different confidence interval accuracy thresholds. Three values for the confidence interval accuracy threshold were tested:

Number of Workers	Metrics	Local steps				
		4	**8**	**16**	**32**	**64**
2	Test Accuracy (%)	56.1	55.35	55.45	53.23	42.83
	Communications (count)	37	18	9	4	2
4	Test Accuracy (%)	54.28	52.61	51.48	50.1	43.93
	Communications (count)	37	18	9	4	2
8	Test Accuracy (%)	50.76	48.01	46.16	43.52	37.72
	Communications (count)	37	18	9	4	2

Table 2. Test accuracy and communication counts for different number of workers (2, 4, and 8) and local steps using the LocalSGD optimizer. It demonstrates the trade-off between communication efficiency and model performance.

Number of Workers	Metrics	Local steps				
		4	**8**	**16**	**32**	**64**
2	Test Accuracy (%)	56	54.61	53.2	48.24	40.55
	Communications (count)	37	18	9	4	2
4	Test Accuracy (%)	53.42	52.3	49.88	47.16	40.36
	Communications (count)	37	18	9	4	2
8	Test Accuracy (%)	50.52	47.81	43.5	40.04	33.98
	Communications (count)	37	18	9	4	2

Table 3. Test accuracy and communication counts for different number of workers (2, 4, and 8) and local steps using the SlowMo optimizer. It illustrates the trade-off between communication frequency and model performance.

0.8, 0.9, and 0.95, with a fixed significance level of 0.8. The number of workers was set to 2, following the optimal configuration identified for LocalSGD. The results are presented in Table 4.

4.5. Results

To analyze the experimental results, different approaches are compared to effectively highlight the key findings.

Centralized baseline vs. Large batch optimizers: As Table 1 shows, LARS demonstrates strong performance up to a batch size of 4K. However, LARS diverges at higher than 4K batch sizes, indicating limitations when scaling to very large batches. In contrast, LAMB shows a gradual decline in accuracy as the batch size increases. This suggests that while LAMB maintains some stability at smaller batch sizes, it struggles to sustain performance as the batch size grows. Compared to the baseline results, SGDM achieves the highest accuracy at a batch size of 64, outperforming both LARS and LAMB at this specific configuration. The results emphasizes the need to choose the appropriate optimizer based on specific batch size requirements.

Centralized baseline vs. LocalSGD: By comparing the centralized baseline methods (SGDM and AdamW) presented in Table 1 and LocalSGD in Table 2, LocalSGD shows a decrease in test accuracy; however, the results remain within an acceptable range. While SGDM achieves the highest accuracy of 57.55% , LocalSGD achieves a slightly lower accuracy of 56.1% with 2 workers and 4 local steps, balancing accuracy with the benefits of reduced communication in a distributed setting.

LocalSGD vs. SlowMo optimizers: The results in Table 2 and Table 3 compare the performance of the LocalSGD and SlowMo optimizers in a distributed learning setting, focusing on test accuracy and communication efficiency for different number of workers and local steps. Both tables illustrate a similar trend: as the number of local steps increases, the communication count decreases, however test accuracy also declines. This trade-off between communication efficiency and model performance is a critical consideration when selecting the optimal configuration. Both LocalSGD and SlowMo optimizers exhibit similar behavior in terms of balancing accuracy and communication costs, but SlowMo may offer marginally better stability at higher local step counts(specifically in 32, 64). These findings highlight the importance of carefully selecting the number of local

Dynamic Selection of Local Steps		
Threshold	**Communications (Count)**	**Test Accuracy**
0.8	28	55.28%
0.9	40	55.35%
0.95	26	54.22%
LocalSGD	*18*	*55.35%*
	37	*56.1%*

Table 4. Test accuracy and communication counts for various confidence interval (CI) accuracy thresholds (0.8, 0.9, and 0.95) in the dynamic selection of local steps, compared to the baseline performance of LocalSGD with different communication counts.

SHAT & Layer-wise Comparison		
Method	**Fine-Tuning**	**Test Accuracy**
Layer-wise Masking	Yes	49.05%
Layer-wise Masking	No	48.84%
SHAT (Baseline)	-	42.86%

Table 5. Test accuracy comparison between different methods: Layer-wise Masking (with and without fine-tuning) and the SHAT baseline.

steps and workers to optimize performance and communication efficiency in distributed training scenarios.

LocalSGD vs. Dynamic selection approach: Based on table 4, LocalSGD with 37 communications achieved a higher test accuracy (56.1%) than any of the dynamic selection configurations. These findings suggest that while dynamic selection of local steps can reduce communication costs at higher thresholds, it may not always provide the best balance between accuracy and communication efficiency, particularly when compared to the LocalSGD baseline.

SHAT vs. Layer-wise masking approach: According to the Table 5 while SHAT algorithm showed a nearly acceptable result with 42.86% accuracy, but Layer-wise Masking could prove its superiority to overcome the computation heterogeneity in a distributed learning system with 48.84% without fine-tuning. Even the accuracy can increase to 49.05% with employing fine-tuning during Layer-wise Masking. In SHAT algorithm the aggregation in our case is 300 which is really high but because of working in an asynchronous process, the idle time is a near zero value. However, in the Layer-wise Masking algorithm, the the number of synchronizations are extremely lower, in this case 37 iterations, and the communication overhead is not near zero but it is decreased.

Algorithm 1 Dynamic Local Step Selection Algorithm

Require: Accuracy threshold T, significance level α, maximum epochs M per worker
Ensure: Models synchronized across all workers
Initialize cumulative steps for each worker i to 0
while Cumulative steps for all workers $\leq M$ **do**
  for each worker i **do**
    if Cumulative steps for worker $i \geq M$ **then**
      continue to next worker.
    else
      Initialize empty list $\mathcal{A}$ to store local accuracies 2
      Conduct local training for one step
      Increment cumulative steps for worker i by 1
      Append obtained training accuracy a_i to list $\mathcal{A}$
      Compute mean accuracy ($\bar{a}$) 3
      Compute standard deviation: (s)4
      Compute estimated confidence interval: (CI)5
      Calculate relative error (E_r) 6
      Adjust computed mean accuracy: (A_{adj}) 7
      if $A_{\text{adj}} < T$ **then**
        Continue local training
      else
        Synchronize across workers
      end if
    end if
  end for
end while

5. Conclusion

In scenarios where training deep learning models on large datasets is computationally demanding, both distributed learning and large-batch methods offer viable solutions, balancing accuracy and generalization according to available hardware resources. While large-batch methods like LARS and LAMB can accelerate training, their stability varies with batch size, and they may not always perform as reliably as distributed methods. When addressing device heterogeneity in distributed learning environments, the layer-wise masking approach proves to be more effective than SHAT, achieving superior generalization with fewer communication rounds by dynamically freezing layers based on device capabilities. Additionally, although the confidence interval-based dynamic selection method for determining local steps provides flexibility and can reduce communication costs, it may not always achieve the highest accuracy or communication efficiency compared to fixed-step approaches like LocalSGD, which demonstrated better overall performance in the experiments. These findings highlight the importance of selecting strategies that align with the specific constraints and objectives of the distributed learning scenario.

References

[1] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Federated visual classification with real-world data distribution. In *Proceedings of the Conference on Machine Learning and Systems (MLSys)*. MIT CSAIL and Google Research, 2020. 3

[2] Yunyong Ko and Sang-Wook Kim. SHAT: A novel asynchronous training algorithm that provides fast model convergence in distributed deep learning. *Applied Sciences*, 12(1):292, 2022. 2

[3] Tao Lin, Sebastian U. Stich, Kumar Kshitij Patel, and Martin Jaggi. Don't use large minibatches, use local SGD. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020. 1

[4] Mehdi Setayesh, Xiaoxiao Li, and Vincent W.S. Wong. Perfedmask: Personalized federated learning with optimized masking vectors. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023. 1, 2

[5] Fede Soriano. Cifar-100 dataset, 2023. Accessed: 2024-09-12. 3

[6] Jianyu Wang, Vinayak Tantia, Nicolas Ballas, and Michael Rabbat. SLOWMO: Improving communication-efficient distributed SGD with slow momentum. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020. 2

[7] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. Technical report, University of California at Berkeley, Carnegie Mellon University, NVIDIA, 2017. Technical Report. 1

[8] Yang You, Jing Li, Sashank J. Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *International Conference on Learning Representations (ICLR)*, 2020. 1