iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 11 **Issue:** XI **Month of publication:** November 2023

DOI: https://doi.org/10.22214/ijraset.2023.56860

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue XI Nov 2023- Available at www.ijraset.com

A Comprehensive Review of Advancements in Communication-Efficient Distributed Optimization

Bhavika Balani[1], Lakshita Sejra[2]

Abstract: ***In the ever-evolving landscape of machine learning and deep neural networks, the demand for training large models on massive datasets has driven research into innovative approaches for efficient distributed optimization. These approaches aim to simultaneously improve model accuracy and generalization while mitigating the communication overhead challenges inherent to distributed training. Reduced communication complexity is highly desirable, as communication overhead often poses a performance bottleneck in distributed systems. This literature review explores the progression of communication-efficient techniques, culminating in the introduction of the Slow Momentum (SLOWMO) framework. It traces the trajectory of distributed optimization, addresses decentralization strategies, and highlights the role of Local Stochastic Gradient Descent (Local SGD) and momentum in communication-efficient algorithms. It delves into Block-Wise Model Update Filtering (BMUF) and introduces SLOWMO, a framework consistently enhancing optimization and generalization performance across various base algorithms. This review unveils the evolving field of communication-efficient distributed optimization, offering theoretical guarantees and practical insights for improving large-scale model training.***

Keywords: Communication efficiency, Decentralized Algorithms, Distributed Learning, Distributed Optimization, Momentum SGD, Slow Momentum (SLOWMO) framework, Stochastic Gradient Descent (SGD)

I. INTRODUCTION

In the realm of machine learning, the remarkable progress achieved in deep learning is unquestionably the result of continuous innovations in the field [1]. These innovations have allowed us to train models of unprecedented scale and complexity, which have proven instrumental in tasks ranging from image recognition to language translation [2, 3]. Yet, the promise of these large models is counterbalanced by a significant challenge: the efficient training of these colossal neural networks on extensive datasets. To meet this challenge head-on, researchers have turned their attention to the development of distributed optimization techniques. Distributed optimization refers to the practice of training large machine learning models using a distributed system, where multiple machines or nodes collaborate to improve the model's accuracy and capabilities collectively. The core objective of distributed optimization is to train large models efficiently on extensive datasets while simultaneously elevating their accuracy and generalization capabilities. Distributed optimization techniques aim to ensure that this can be achieved within a reasonable time frame without overwhelming computational demands. These techniques address the challenges of coordinating and communicating between multiple machines, ensuring that they work in harmony to collectively improve the model's performance [4, 5]. This is essential for handling the increased complexity of large models and the demands of extensive datasets.

The concept of communication-efficient distributed optimization is central to this pursuit. This term encapsulates strategies and methodologies designed to reduce the communication overhead inherent in distributed systems. Efforts to minimize the communication complexity of distributed optimization have become a central focus of machine learning research. Reducing the frequency and volume of communication between distributed components is of paramount importance, as this communication overhead often acts as a significant bottleneck, impacting both the pace and quality of model training [6]. The advantages of communication-efficient distributed optimization are evident. Firstly, it allows us to train large models that can handle complex tasks with high precision. Secondly, it enables the training process to be scaled efficiently, accommodating larger datasets and models. Moreover, these advancements can lead to considerable savings in both computation time and resource utilization.

This literature review embarks on an extensive journey through the timeline of these advancements, with a particular focus on the introduction of the Slow Momentum (SLOWMO) framework [7]. Our exploration begins by retracing the trajectory of distributed optimization, recognizing seminal works that laid the foundation for the field's progress. We delve into strategies that have been developed to confront communication overhead challenges and decentralization strategies that have reshaped the landscape of distributed machine learning. In this context, we highlight the significance of Local Stochastic Gradient Descent (Local SGD) and the innovative integration of momentum in communication-efficient algorithms [8].

Additionally, we navigate through the intricacies of Block-Wise Model Update Filtering (BMUF), emphasizing the trade-off it strikes between model accuracy and communication overhead. BMUF, though instrumental in the training of speech models, has long grappled with the absence of theoretical convergence guarantees and widespread applicability. Inspired by the principles of BMUF, the literature introduces the SLOWMO framework, which aims to consistently elevate optimization and generalization performance across a spectrum of base algorithms. SLOWMO not only bolsters the efficiency of communication but also provides theoretical guarantees, enhancing its significance within the field. This literature review, therefore, provides a critical synthesis of the evolving field of communication-efficient distributed optimization, offering both theoretical and practical insights. It aims to shed light on the path of progress and offer directions for further research, encapsulating the spirit of innovation and efficiency that drives machine learning forward.

II. DISTRIBUTED OPTIMIZATION ADVANCES

A. Synchronous Optimization

Synchronous optimization refers to a class of optimization methods where all participating agents or nodes in a distributed system coordinate and update their models or variables simultaneously, typically in a lockstep fashion. This synchronization can occur at regular intervals or steps, ensuring that all nodes in the system update their information concurrently. In the realm of distributed optimization, synchronous optimization techniques have been pivotal in coordinating large-scale machine learning tasks. These approaches employ mini-batch versions of stochastic gradient descent (SGD) and other stochastic optimization algorithms like AdaGrad, RMSProp, and ADAM. Stochastic gradient descent (SGD) is one of the foundational optimization algorithms used in machine learning. It updates model parameters based on the gradients of the loss function with respect to those parameters. By employing mini-batches, the stochasticity in SGD is reduced, leading to more stable and quicker convergence. Additionally, other stochastic optimization algorithms like AdaGrad, RMSProp, and ADAM introduce adaptive learning rates. They dynamically adjust the learning rates for each model parameter based on the historical gradient information. This adaptation allows for faster convergence in many cases and is especially effective when dealing with non-convex optimization problems. While these synchronous optimization techniques using mini-batches are highly effective in training large machine learning models, they do face challenges, particularly in terms of synchronization delays. The need to ensure that all workers complete their tasks within a batch introduces dependencies on the slowest worker. The computational workload and communication times of the workers can vary, leading to synchronization bottlenecks.

B. Asynchronous Optimization

To overcome synchronization delays and accelerate the training process, asynchronous optimization techniques emerged as a promising alternative. These methods prioritize speed, allowing workers to operate independently and potentially use stale information for computation. Stale information refers to gradient information that may not be the most up-to-date or accurate due to the inherent asynchrony of the system. In a distributed machine learning environment, there are multiple workers operating independently and asynchronously. Each worker processes its own mini-batch of data and computes gradients based on the model parameters it has at that moment. The staleness arises because these model parameters might not be the most current. As workers operate independently, they fetch the model parameters from a shared set of servers, but these parameters may be updated by other workers while computation is ongoing. This means that when a worker starts processing a mini-batch, the model parameters it is using might not be the latest ones, as other workers could have already made updates. The decision to use potentially stale information is deliberate in asynchronous optimization because it allows for faster computation. Workers do not need to wait for the most up-to-date model parameters, and they can proceed with their computations immediately. While asynchronous approaches offer faster training, they often lead to convergence issues and require careful tuning to balance speed and accuracy.

C. Revisiting Distributed Synchronous SGD

In recent years, significant advancements have shaped the landscape of distributed optimization. To address the challenges posed by Asynchronous Stochastic Gradient Descent (Async-SGD), Synchronous Stochastic Gradient Descent (Sync-SGD) was revisited and introduced the concept of backup workers [9]. These additional workers work in tandem with the primary workers and play a pivotal role in addressing stragglers. When parameter servers receive gradients from enough primary workers, they do not wait for stragglers but proceed with updating the model parameters. While the gradients from the slowest workers are eventually dropped, this strategy significantly improves the efficiency of synchronous mini-batch SGD. It ensures that the effective batch size remains consistent and is equal to the sum of all mini-batch sizes across workers. By doing so, this advancement helps maintain the synchronization of the distributed system while mitigating the impact of slower machines.

D. Large Minibatch SGD

The introduction of large minibatch stochastic gradient descent (SGD) marked a pivotal advancement in the efficient training of deep neural networks, especially in the context of large-scale tasks like ImageNet [10]. This approach sought to overcome the optimization challenges associated with larger mini-batches, which had the potential to hinder the training process. One key innovation was the introduction of a hyperparameter-free linear scaling rule designed to adjust learning rates based on the mini-batch size. This rule ensures that as mini-batch sizes grow, the learning rate adapts accordingly, preventing potential obstacles to efficient training. This adjustment turns the larger mini-batches into an advantage rather than a hindrance, contributing to improved generalization. Additionally, a novel warmup scheme was introduced to tackle early optimization challenges during the training process. This warmup scheme effectively addresses issues that can arise at the outset of training, ensuring that the optimization process gets off to a strong start. The combination of these techniques showcased the possibility of training deep neural networks with large mini-batches efficiently. This advancement strikes a balance between speed and accuracy and has been instrumental in addressing the challenges associated with training large models on extensive datasets.

E. ALLREDUCE-Based Methods

At present, the prevailing approaches in the field involve a parallelized process where individual workers perform computations on small mini-batch gradients locally. Subsequently, these local computations are aggregated using a blocking communication technique known as ALLREDUCE before proceeding to take an optimizer step. This process allows for the efficient utilization of multiple workers, making parallel processing of mini-batch gradients a common practice in distributed optimization. The use of ALLREDUCE helps synchronize these gradient updates, ensuring that the model's parameters are updated collectively across all workers. This approach is foundational to many distributed optimization methods and has played a crucial role in coordinating large-scale machine learning tasks.

III. DECENTRALIZED TRAINING

The most widely used approaches involve workers computing mini-batch gradients locally and then aggregating these gradients using the ALLREDUCE communication primitive. However, communication overhead poses a significant challenge to the scalability of these methods.

This issue has been investigated by researchers, who explored strategies to deal with stragglers and improve distributed SGD. Multiple complementary approaches have recently been investigated to reduce or hide communication overhead. To reduce idling due to blocking and stragglers, decentralized training [11,12,13] methods have been proposed. These approaches employ approximate gradient aggregation techniques, such as gossip algorithms or distributed averaging, which allow for efficient collaboration among distributed components while mitigating the challenges associated with synchronization. Decentralized methods fundamentally redefine the way in which gradients are aggregated in a distributed setting. Instead of relying on centralized aggregation through mechanisms like ALLREDUCE, decentralized approaches distribute the responsibility of gradient computation and updates across the network's nodes, promoting a more distributed and asynchronous framework. This shift in approach significantly reduces communication overhead and synchronization delays, contributing to the overall efficiency of the distributed optimization process.

IV. COMBINING DECENTRALIZED ALGORITHMS WITH LOCAL SGD

A. Limitations of Mini-Batch SGD and Introduction to Local SGD

Mini-batch stochastic gradient descent (SGD) is the state of the art in large-scale distributed training. However, this method can often encounter challenges in practice.

While it theoretically offers linear speedup with respect to the number of workers, this speedup is rarely realized in real-world applications. One primary reason is the presence of significant network delays and bandwidth constraints, which can lead to substantial communication bottlenecks [14]. To address these communication bottlenecks, recent research has proposed techniques that reduce the frequency of communication. Local SGD, an algorithm of this type, tackles these challenges by running independent SGD processes in parallel on different workers. It ensures that sequences are only averaged periodically rather than after every mini-batch, significantly reducing the communication overhead. The Local SGD scheme has shown promising results theoretically and in practice. [14].

B. Combining Local SGD with Decentralized Training

Combining the principles of Local SGD with decentralized training offers a promising approach to addressing the limitations of traditional mini-batch SGD. By integrating Local SGD with decentralized training, researchers aim to strike a balance between communication efficiency and model quality. Local SGD brings the advantage of fewer communication rounds by allowing workers to perform multiple updates before synchronization. Decentralized training leverages the benefits of distributed and asynchronous gradient computation, further reducing communication bottlenecks [15,16]. The advantages of this combination are two-fold. Firstly, it allows for a reduction in communication overhead, making distributed training more efficient. Secondly, the integration of Local SGD with decentralized training provides a pathway to achieving high-quality model updates without the burden of frequent synchronization. This approach reduces communication overhead while injecting additional noise into the optimization process. Consequently, although it runs faster than large minibatch methods, the resulting model may not achieve the same quality in terms of training loss or generalization accuracy after the same number of iterations. In essence, it introduces additional noise into the training process, which can lead to more robust convergence in practice. The challenge is to maintain communication efficiency while achieving high-quality model updates.

V. INCORPORATING MOMENTUM

Momentum is a key component in training deep neural networks, and it has been empirically demonstrated to improve both optimization and generalization [17]. In essence, momentum introduces a sense of inertia to the training process, allowing the model to maintain direction and velocity, which helps navigate through optimization landscapes and accelerate convergence. It acts as a moving average of gradients, which helps smooth out the optimization trajectory and escape local minima more effectively. By doing so, momentum improves both the speed of convergence and the final quality of the trained model. The concept of momentum is not new; it has been widely adopted in traditional stochastic gradient descent (SGD) to improve training performance. However, in the context of communication-efficient distributed training algorithms, there is no one-size-fits-all approach to incorporating momentum. The challenge lies in balancing the benefits of momentum with the added complexities introduced by distributed and decentralized training methods. Typically, momentum is incorporated into communication-efficient training algorithms by having workers maintain separate momentum buffers that are not synchronized [18, 19, 20, 21]. This approach allows each worker to calculate and update their momentum values independently. These separate momentum buffers contribute to the noisiness of the optimization process, which can be advantageous in certain scenarios. This additional noise can help the optimization algorithm escape local minima more effectively and improve the robustness of the training process.

A more recent study [22] introduced an alternative perspective focusing on the linear speedup analysis of communication-efficient momentum SGD for distributed non-convex optimization. This study examined whether distributed momentum SGD possesses the same linear speedup property as distributed SGD and whether it can reduce communication complexity. The research considered a distributed communication-efficient momentum SGD method and provided evidence of its linear speedup property. The linear speedup property is a significant advantage in distributed machine learning as it allows the system to scale out computing capability effectively by adding more computing nodes. This property, combined with the benefits of momentum in optimization, can lead to faster convergence and improved generalization. However, it's worth noting that while this synchronization of the momentum buffer enhances accuracy, it comes at the cost of doubling the communication overhead.

VI. SYNCHRONIZATION OF MOMENTUM BUFFERS IN BMUF

In BMUF, the traditional approach to optimizing deep learning models is reimagined, offering unique advantages while also introducing a set of challenges [23]. BMUF introduces a decentralized paradigm in which nodes within a distributed system perform multiple local optimization steps between communication rounds, similar to the principles of local SGD. However, what sets BMUF apart is the incorporation of a momentum buffer, a critical component in the optimization of deep neural networks. This buffer, unlike the frequent update mechanism employed by traditional optimization techniques, is refreshed only after each communication round. The synchronization of the momentum buffer at this specific juncture is central to BMUF's operation.

The motivation behind this approach is the amalgamation of two fundamental principles: the benefits of momentum in optimization and the alleviation of communication overhead. The momentum buffer enhances the optimization process by allowing the network to retain and build upon past gradients, which can significantly expedite convergence and enhance generalization [23]. It provides a memory of the network's previous state, aiding in navigating through complex loss landscapes. However, this advantageous feature is not without its drawbacks.

The synchronization of the momentum buffer, while undeniably valuable in terms of improving optimization accuracy, comes at the cost of increased communication overhead. This trade-off is a critical consideration when assessing the applicability of BMUF to various machine learning tasks. For instance, BMUF has seen substantial use in training speech models, demonstrating notable improvements in convergence and recognition accuracy for tasks such as large vocabulary continuous speech recognition [23]. However, its utilization in other domains, such as computer vision or natural language processing, remains limited. The primary impediment to its wider adoption is the absence of theoretical convergence guarantees. The lack of such guarantees raises questions regarding the reliability and consistency of BMUF when applied to diverse tasks and models.

VII. SLOWMO FRAMEWORK

The Slow Momentum (SLOWMO) framework, introduced as a novel approach to improving communication-efficient distributed training, is inspired by the Block-Wise Model Update Filtering (BMUF) technique [7]. SLOWMO operates on top of a base algorithm, such as Local Stochastic Gradient Descent (Local SGD) or a decentralized method like stochastic gradient push (SGP) [24,25]. Periodically, workers average their parameters using ALLREDUCE and perform a momentum update. Its primary goal is to enhance the accuracy of communication-efficient distributed training methods.

A. Structure of the SLOWMO Framework. Refer to Fig. 1.

Input: Base optimizer with learning rate γ_t; Inner loop steps τ; Slow learning rate α; Slow momentum factor β; Number of worker nodes m. Initial point $\boldsymbol{x}_{0,0}$ and initial slow momentum buffer $\boldsymbol{u}_0 = \mathbf{0}$.

for $t \in \{0, 1, \ldots, T-1\}$ **at worker** i **in parallel do**
  Reset/maintain/average base optimizer buffers
  for $k \in \{0, 1, \ldots, \tau-1\}$ **do**
    Base optimizer step: $\boldsymbol{x}_{t,k+1}^{(i)} = \boldsymbol{x}_{t,k}^{(i)} - \gamma_t \boldsymbol{d}_{t,k}^{(i)}$
  end
  Exact-Average: $\boldsymbol{x}_{t,\tau} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_{t,\tau}^{(i)}$
  Update slow momentum:
  $\boldsymbol{u}_{t+1} = \beta \boldsymbol{u}_t + \frac{1}{\gamma_t}(\boldsymbol{x}_{t,0} - \boldsymbol{x}_{t,\tau})$
  Update outer iterates: $\boldsymbol{x}_{t+1,0} = \boldsymbol{x}_{t,0} - \alpha\gamma_t \boldsymbol{u}_{t+1}$
end

Fig. 1 SlowMo Algorithm

SLOWMO is structured with a nested loop and operates in the following manner:

1) *Initialization:* Each worker maintains a local copy of the parameters, $x^{(i)}_{t,k}$ at worker *i* after the k^{th} inner step of the t^{th} outer iteration. Each worker begins with an initial set of parameters, represented as $x_{0,0}$, and a slow momentum buffer u_0, which is initially set to zero.
2) *Outer Iteration Loop:* The framework conducts a series of outer iterations, denoted by 't'. In parallel, all workers participate in these iterations.
3) *Base Algorithm Steps:* Within each outer iteration, workers perform τ steps of the chosen base optimizer. This base optimizer can be one that involves no communication (e.g., SGD) or a decentralized algorithm with limited communication (e.g., SGP). The parameters are updated locally at each worker using the specified learning rate and update direction.
4) *Exact-Average and Slow Momentum Update:* After τ base optimizer steps, workers calculate the average of their parameters, denoted as $x_{t,\tau}$ by employing the ALLREDUCE communication mechanism. Subsequently, a slow momentum update is performed using the computed average. The update of the slow momentum buffer u_{t+1} and the outer iterate $x_{t+1,0}$ is conducted in parallel across all workers.

- The slow momentum buffer u_{t+1} is updated using the previous buffer u_t, the learning rate γ_t, and the difference between $x_{t,0}$ and $x_{t,\tau}$. This update is designed to ensure invariance to the fast-learning rate γ_t, which may vary throughout training due to learning rate schedules.
- The outer iterate $x_{t+1,0}$ is updated by adjusting $x_{t,0}$ based on the slow momentum buffer u_{t+1} and the product $\alpha\gamma_t$ of the slow and fast learning rates.

The nested structure of the framework, with base algorithm steps, exact-averaging, and momentum updates, allows SLOWMO to adapt to a variety of scenarios and improve training efficiency without significantly sacrificing model accuracy.

B. Empirical Results

Empirical results from the research paper [7] demonstrate that SLOWMO consistently enhances optimization and generalization performance across various base algorithms, including training ResNets on CIFAR-10 and ImageNet and training a transformer on WMT'16 En-De. It achieves the goals of speeding up the training process and improving the scaling performance of communication-efficient distributed methods, all while maintaining high levels of accuracy. A graph comparing the training time and training loss of the original algorithm versus algorithm with using SlowMo shown below (Fig. 2 and Fig. 3).

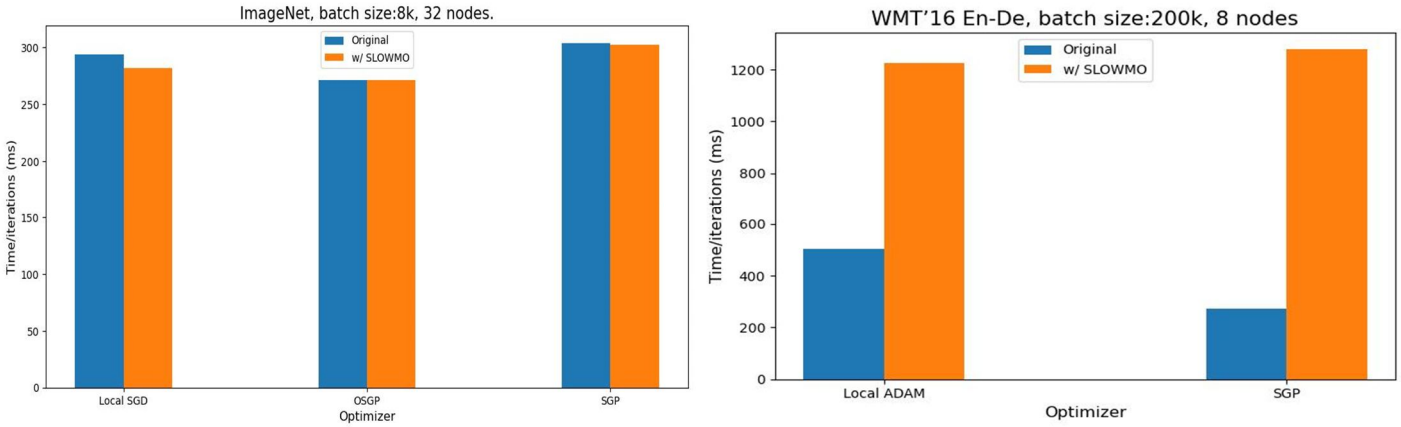


Fig. 2 Training Time SlowMo vs Original)

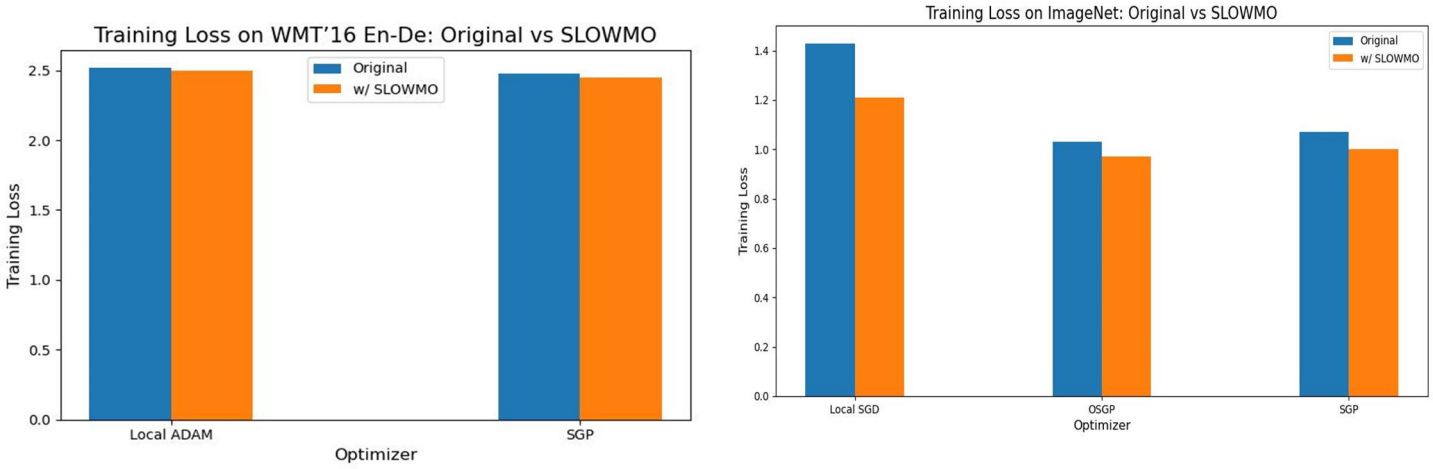


Fig. 3 Training Loss (SlowMo vs Original)

C. Theoretical Convergence Bounds

The research provides theoretical guarantees that SLOWMO converges to a stationary point of smooth non-convex functions at a rate of $O(1/\sqrt{(mT\tau)})$ after $T\tau$ total inner optimization steps and T SLOWMO updates, with m worker nodes. These convergence results demonstrate that SLOWMO is, in terms of convergence speed, at least as fast as stochastic gradient descent (SGD). This makes it a promising approach for a wide range of applications where SGD is commonly used.

D. Future Directions

The introduction of the SLOWMO framework opens up various avenues for future research and development:

1) Hyperparameter Optimization: Further exploration of hyperparameter settings within the SLOWMO framework to understand how different parameter choices impact the performance of the algorithm.

2) Integration with Compression Techniques: Investigating the integration of SLOWMO with gradient compression techniques to reduce communication overhead further while maintaining or improving accuracy.

3) Theoretical Analysis: Expanding the theoretical analysis to encompass additional scenarios, such as non-convex optimization in the presence of noise and other real-world factors.

4) Applications Beyond Image Classification and Machine Translation: Extending the use of SLOWMO to other domains and tasks to assess its broad applicability in various fields of machine learning and deep learning.

5) Benchmarking and Scalability Studies: Conducting benchmark studies and scalability assessments of the SLOWMO framework to understand its potential in large-scale distributed systems and cloud computing environments.

6) Integration into Popular Deep Learning Frameworks: Implementing SLOWMO in popular deep learning frameworks to make it accessible and convenient for the broader machine learning community.

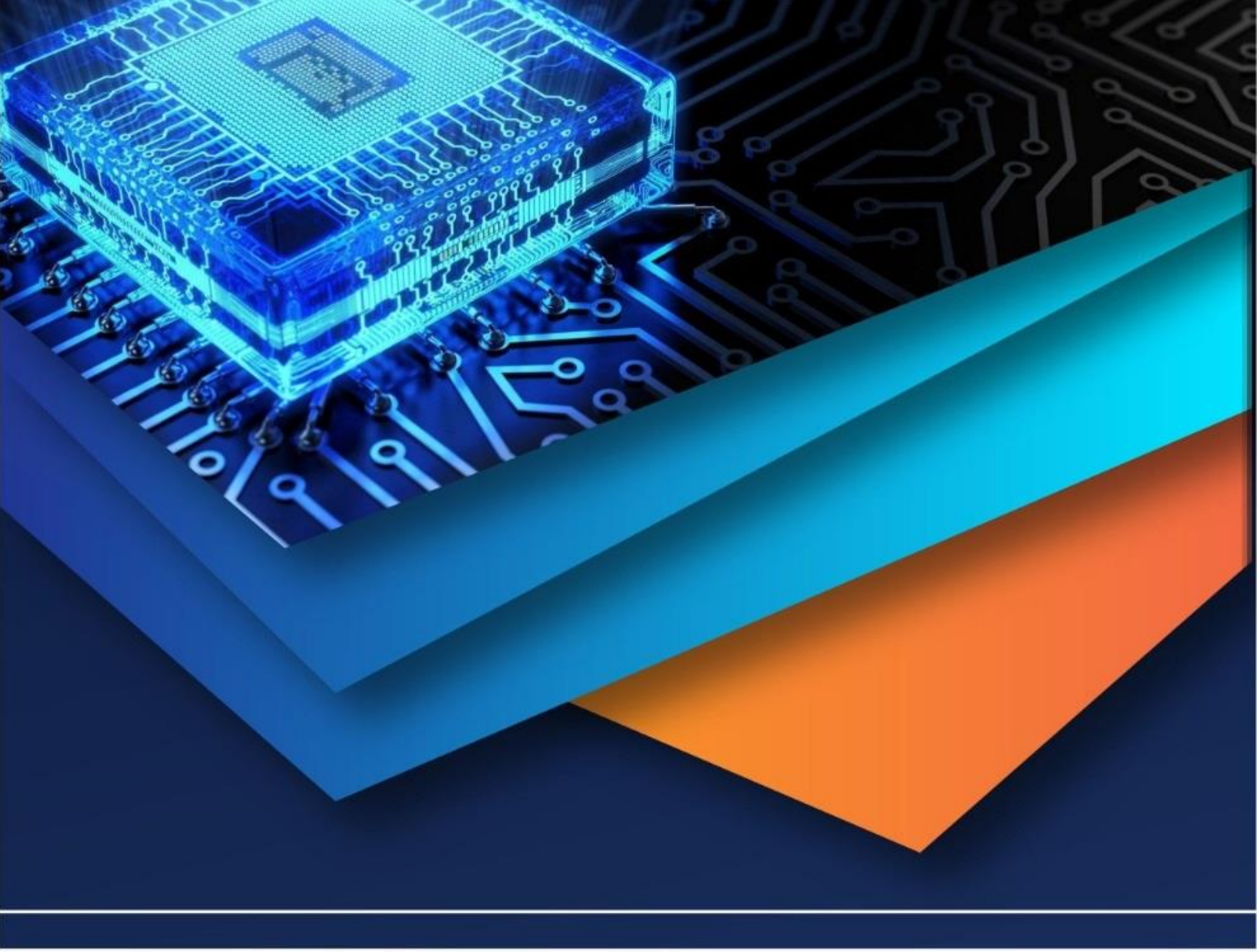
VIII. CONCLUSION

In this comprehensive review, we've traced the evolution of communication-efficient distributed optimization in machine learning, addressing the challenges of training large models on extensive datasets. We explored various strategies, from synchronous and asynchronous optimization methods to decentralized training, integrating momentum to enhance convergence and generalization. The innovative Slow Momentum (SLOWMO) framework, inspired by Block-Wise Model Update Filtering (BMUF), introduces a nested structure that seamlessly blends base algorithms with exact-averaging and momentum updates. Empirical results and theoretical guarantees demonstrate SLOWMO's consistent enhancement of optimization and generalization across multiple base algorithms. The journey through communication-efficient distributed optimization presents a promising path forward, with opportunities for further exploration in hyperparameter optimization, compression techniques, theoretical analysis, broadened applications, benchmarking, scalability, and integration into popular deep learning frameworks. In the quest for efficiency and innovation, we are poised on the threshold of an era of more effective machine learning and deep neural network training.

REFERENCES

[1] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature 521, 436–444 (2015). https://doi.org/10.1038/nature14539
[2] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I. & others (2019). Language models are unsupervised multitask learners. OpenAI blog, 1, 9.
[3] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. ArXiv. /abs/1706.03762
[4] Chen, J., Pan, X., Monga, R., Bengio, S., & Jozefowicz, R. (2016). Revisiting Distributed Synchronous SGD. ArXiv. /abs/1604.00981
[5] Priya Goyal, Piotr Dollar, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. arXiv preprint arXiv:1706.02677, 2017.
[6] Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD. In International Conference on Artificial Intelligence and Statistics, pp. 803–812, 2018.
[7] Wang, J., Tantia, V., Ballas, N., & Rabbat, M. (2019). SlowMo: Improving Communication-Efficient Distributed SGD with Slow Momentum. ArXiv. /abs/1910.00643
[8] K. Chen and Q. Huo, "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Shanghai, China, 2016, pp. 5880-5884, doi: 10.1109/ICASSP.2016.7472805.
[9] Chen, J., Pan, X., Monga, R., Bengio, S., & Jozefowicz, R. (2016). Revisiting Distributed Synchronous SGD. ArXiv. /abs/1604.00981
[10] Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., & He, K. (2017). Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. ArXiv. /abs/1706.02677
[11] Jiang, Z., Balu, A., Hegde, C., & Sarkar, S. (2017). Collaborative Deep Learning in Fixed Topology Networks. ArXiv. /abs/1706.07880
[12] Lian, X., Zhang, C., Zhang, H., Hsieh, C., Zhang, W., & Liu, J. (2017). Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent. ArXiv. /abs/1705.09056
[13] Assran, M., Loizou, N., Ballas, N., & Rabbat, M. (2018). Stochastic Gradient Push for Distributed Deep Learning. ArXiv. /abs/1811.10792
[14] Stich, S. U. (2018). Local SGD Converges Fast and Communicates Little. ArXiv. /abs/1805.09767
[15] Stich, S. U. (2018). Local SGD Converges Fast and Communicates Little. ArXiv. /abs/1805.09767
[16] Wang, J., & Joshi, G. (2018). Cooperative SGD: A unified Framework for the Design and Analysis of Communication-Efficient SGD Algorithms. ArXiv. /abs/1808.07576
[17] Wang, J., Sahu, A. K., Yang, Z., Joshi, G., & Kar, S. (2019). MATCHA: Speeding Up Decentralized SGD via Matching Decomposition Sampling. ArXiv. /abs/1905.09435

[18] 2013: Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In International Conference on Machine Learning, pp. 1139–1147, 2013.
[19] 2017: Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In Advances in Neural Information Processing Systems, pp. 5330–5340, 2017.
[20] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In Proceedings of the 35th International Conference on Machine Learning, pp. 3049–3058, 2018.
[21] Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Michael Rabbat. Stochastic gradient push for distributed deep learning, In International Conference on Machine Learning, 2019.
[22] 2019a: Anastasia Koloskova, Tao Lin, Sebastian U Stich, and Martin Jaggi. Decentralized deep learning with arbitrary communication compression. arXiv preprint arXiv:1907.09356, 2019a.
[23] Yu, H., Jin, R., & Yang, S. (2019). On the Linear Speedup Analysis of Communication Efficient Momentum SGD for Distributed Non-Convex Optimization. ArXiv. /abs/1905.03817
[24] K. Chen and Q. Huo, "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Shanghai, China, 2016, pp. 5880-5884, doi: 10.1109/ICASSP.2016.7472805.
[25] Angelia Nedic and Alex Olshevsky. Stochastic gradient-push for strongly convex functions on time-varying directed graphs. IEEE Trans. Automatic Control, 61(12):3936–3947, 2016.
[26] Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Michael Rabbat. Stochastic gradient push for distributed deep learning. In International Conference on Machine Learning, 2019.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY