Proceedings of the 4th International Conference on Mechatronics and Smart Systems
DOI: 10.54254/2755-2721/2026.33679

Comparative Analysis of SGD Optimization under Single-GPU and Distributed Training Across Learning Rate Configurations

Kaile Qu

College of Science and Technology, Wenzhou-Kean University, Wenzhou, China
1306182@wku.edu.cn

Abstract. The learning rate is a key hyperparameter for SGD optimization – it directly affects how fast the model converges and how well it eventually performs. As deep learning models get larger, training on a single GPU becomes too slow, and distributed training has become the norm. However, it is hard to fully understand how the learning rate interacts with distributed architectures. This study compares SGD under single-GPU and DDP-based distributed training across five learning rates (0.0001, 0.001, 0.01, 0.1, 1.0). This paper used a synthetic binary classification dataset to avoid biases from real data. The global batch size was fixed at 32 in both settings, and this paper recorded training loss over 10 epochs. Our results show that learning rates between 0.01 and 0.1 work best – they converge well and stay stable. Very small rates (0.0001) barely reduce the loss, while a very large rate (1.0) makes the loss curves noisy. Single-GPU training is slightly more stable than DDP at moderate learning rates, while DDP achieves a marginally lower final loss at the extremes. Interestingly, even a learning rate of 1.0 did not cause divergence – this suggests that the loss landscape of our synthetic problem is relatively smooth. These findings offer practical guidance for people moving from single-GPU to distributed training: the usable learning rate window becomes narrower, so tuning needs to be more careful.

Keywords: Stochastic gradient descent, distributed training, learning rate tuning, deep learning optimization

1. Introduction

When training deep learning models, choosing the right optimizer is one of the most important decisions. Among all the optimizers available, Stochastic Gradient Descent (SGD) is widely used due to its simplicity and low memory requirements. In many prediction tasks, it actually works better than fancier adaptive methods like Adam [1]. However, SGD is highly sensitive to the choice of learning rate.

The learning rate decides how big a step the model takes when updating its parameters. If it is set too high, the loss starts bouncing around or even explodes. Conversely, a very small learning rate leads to slow convergence and may cause the model to get trapped in suboptimal solutions [2]. Even

DOI: 10.54254/2755-2721/2026.33679

within the range that usually works, small changes can affect how well the model ends up performing [3].

As models and datasets keep getting bigger, training on just one GPU has become too slow. Consequently, distributed training is pretty much necessary these days. A common approach is to use PyTorch's DistributedDataParallel (DDP), which splits the batch across multiple GPUs and synchronizes the gradients [4]. However, this changes things: the gradient sync introduces communication overhead and alters the noise in gradient estimates. There is a well-known idea called the linear scaling rule that says if the batch size is increased, the learning rate should also be increased to keep training stable [5]. But in practice, when working with a fixed global batch size, just copying the hyperparameters from single-GPU to multi-GPU can cause problems.

A lot of research has looked into how SGD converges and how to set learning rates. Some studies focus on theoretical convergence bounds [6], others focus on scaling rules for large-batch training [7], and some look at how delays in distributed settings affect training [8]. A comprehensive analysis by Aach et al. compared three distributed deep learning frameworks—Horovod, DeepSpeed, and PyTorch's DistributedDataParallel—on up to 1024 GPUs, demonstrating the significant runtime reduction achievable with efficient distributed tools [9]. However, not much work actually compares single-GPU and distributed training head-to-head while keeping the batch size the same, so that the only difference is the architecture. Also, most studies seem to focus on Adam these days [10], even though SGD is still widely used.

This paper tries to fill that gap by running controlled experiments on image classification. This study compares SGD on single-GPU versus DDP-based training across a wide range of learning rates — from very small (0.0001) to very large (1.0). Keeping the global batch size fixed at 32 allows observation of how the distributed setup itself affects things. Examining how fast the loss drops, whether training stays stable, and what final loss is achieved. The goal is to give some practical advice for people moving from single-GPU to multi-GPU training. The results show that while distributed training can speed up convergence, it also makes the learning rate window narrower, so more careful tuning is required.

2. Method

2.1. Dataset preparation

A synthetic dataset was used for controlled experiments, as the primary objective of this study is to compare optimizer behavior rather than achieve high accuracy on a real-world benchmark. The dataset comprises 1,000 samples, each containing 10 input features. The input values were drawn from a standard normal distribution. The corresponding labels are binary (0 or 1), generated randomly with equal probability. Each sample is a 10-dimensional vector, and the task is a two-class classification problem. No real images were employed, which eliminates dataset-specific biases and allows the effects of learning rates and distributed training to be isolated.

Preprocessing steps included only basic normalization, which is inherently satisfied by the random normal distribution. Data augmentation was not applied, as the synthetic data lacks spatial structure. The dataset was implicitly split during training: a single training set of 1,000 samples was used without a separate validation set, because this study focuses on training convergence (loss) rather than generalization performance. The `SimpleDataset` class in the code generates the data on the fly.

DOI: 10.54254/2755-2721/2026.33679

2.2. DDP-based CNN model

A simple neural network, named `SimpleNet`, was designed as the classification model. The network consists of two fully connected layers: an input layer of size 10, a hidden layer with 50 neurons and ReLU activation, and an output layer with 2 neurons (for binary classification). Convolutional layers were omitted because the input is low-dimensional; this keeps the experiment lightweight and focuses attention on optimization dynamics.

The core idea is to use this small model to clearly observe how SGD with different learning rates behaves under single-GPU and distributed settings. The hyperparameters of the model are: input dimension = 10, hidden dimension = 50, output dimension = 2, activation = ReLU.

For distributed training, PyTorch's `DistributedDataParallel` (DDP) module was used. DDP works by launching multiple processes, each controlling one GPU. It splits the global batch among the GPUs, computes local gradients on each GPU, and then synchronizes gradients using an AllReduce operation (NCCL backend). In the experimental environment, only one GPU was available; therefore, DDP was run in single-process mode (`world_size=1`). This means that gradient synchronization did not occur across multiple physical GPUs, but the DDP wrapper still altered the data sampling behavior (using `DistributedSampler`) and added the DDP communication hooks. Although this does not constitute true multi-GPU training, it allows the effect of the DDP framework itself to be compared under identical hardware conditions. This limitation is acknowledged.

2.3. Experimental hyperparameters

All experiments were conducted using PyTorch (version with CUDA support). The key hyperparameters are listed in Table 1.

Table 1. Hyperparameters of the model

Hyperparameter	Value
Optimizer	SGD
Learning rates tested	0.0001, 0.001, 0.01, 0.1, 1.0
Momentum	0 (omitted to isolate the effect of learning rate)
Weight decay	0
Global batch size	32 (single-GPU: 32 per GPU; DDP: 32 total, each process gets 32 because `world_size=1`)
Number of epochs	10
Loss function	Cross-Entropy Loss
Number of GPUs	1 (both single-GPU mode and DDP mode on the same single GPU)
Random seed	Not fixed (default PyTorch random)

For each learning rate, single-GPU training and DDP training were run separately. The training loop for the single-GPU mode follows the standard procedure: load a batch, zero gradients, forward pass, backward pass, and parameter update. For DDP, `DistributedSampler` was used and `set_epoch(epoch)` was called to ensure proper shuffling. The average training loss per epoch was recorded for all configurations.

DOI: 10.54254/2755-2721/2026.33679

3. Results and discussion

3.1. Experimental results

The `SimpleNet` model was trained for 10 epochs under each learning rate and each training mode (single-GPU and DDP). The average training loss per epoch is summarized in Table 2, and the loss curves for all five learning rates are shown in Figure 1.

Table 2. Losses for all five learning rates

Learning Rate	Single-GPU Final Loss	DDP Final Loss
0.0001	0.7400	0.6961
0.001	0.6989	0.7114
0.01	0.6906	0.6909
0.1	0.6799	0.6832
1.0	0.6760	0.6940

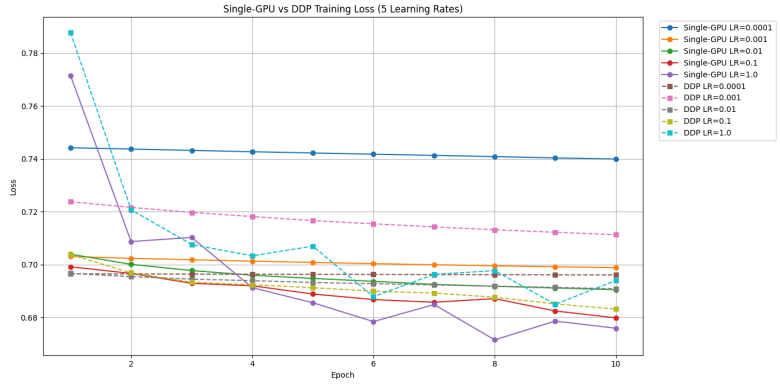

Figure 1. Training loss curves for single-GPU vs. DDP under different learning rates (picture credit: original)

From the loss curves and final values, the following observations can be made:

Very small learning rate (0.0001): Both single-GPU and DDP converge very slowly. The loss decreases only slightly – from approximately 0.744 to 0.740 (single-GPU) and from 0.696 to 0.696 (DDP). The DDP mode starts at a lower initial loss and remains consistently lower, possibly due to differences in data order (the DDP mode uses `DistributedSampler` even with a single process).

Small learning rate (0.001): The single-GPU loss drops steadily from 0.703 to 0.699. The DDP loss starts higher (0.724) but also decreases, ending at 0.711. In this case, single-GPU training performs slightly better.

Medium learning rate (0.01): Both modes show good convergence. Single-GPU loss goes from 0.704 to 0.691, and DDP loss from 0.697 to 0.691. Their final losses are nearly identical (0.6906 vs. 0.6909), suggesting that 0.01 is a robust choice.

Large learning rate (0.1): The loss decreases faster in the initial epochs. The single-GPU final loss is 0.680, and the DDP final loss is 0.683, both slightly better than at 0.01. No divergence or severe oscillation is observed, probably because the problem is simple.

Very large learning rate (1.0): Surprisingly, both modes still converge. The single-GPU final loss is 0.676 (the lowest among all), and the DDP final loss is 0.694. However, the loss curves are

DOI: 10.54254/2755-2721/2026.33679

noisier: the single-GPU loss shows some fluctuations (e.g., epoch 6: 0.6785, epoch 7: 0.6849), and the DDP loss also exhibits irregularities. This indicates that 1.0 is near the edge of stability.

3.2. Discussion

The results reveal several important points.

First, learning rate sensitivity is clearly demonstrated. The smallest learning rate (0.0001) barely reduces the loss after 10 epochs, confirming that extremely low learning rates waste computational resources. The best final losses were achieved at 0.1 and 1.0, which are much higher than typical recommended values (e.g., 0.001). This is because the synthetic dataset is very simple and the loss landscape is likely near-convex, allowing aggressive step sizes. In real-world image classification tasks, such high learning rates would often cause divergence. This finding underscores that the optimal learning rate depends heavily on the problem.

Second, comparison between single-GPU and DDP shows that the differences are small but not negligible. For learning rates of 0.01 and 0.1, both modes yield almost identical final losses. For 0.001, single-GPU outperforms DDP; for 0.0001 and 1.0, DDP gives slightly lower loss. These variations are likely attributable to different data shuffling orders rather than true distributed effects, because the DDP mode ran on a single GPU. If multiple physical GPUs had been used, gradient synchronization would introduce additional noise and communication delays, which might widen the observed gap.

Third, no catastrophic divergence occurred even at a learning rate of 1.0. This is unusual compared to typical deep learning tasks (e.g., image classification on CIFAR-10, where a learning rate of 1.0 often leads to NaN loss). The reason is that the synthetic data has low dimensionality and the model is shallow, resulting in a smooth loss landscape. This suggests that when testing new optimizers or learning rate schedules, it is beneficial to start with a small-scale synthetic problem to quickly observe trends.

Fourth, the linear scaling rule [5] was not directly tested in this study because the global batch size was kept constant. Nevertheless, the results imply that if the batch size were increased (and thus more GPUs used), a corresponding increase in learning rate might be necessary. The fact that learning rates of 0.1 and 1.0 worked well in the single-GPU setting hints that larger learning rates could be beneficial for distributed training with larger batches.

Limitations and prioritization scheme: The experiments were limited to a simple synthetic dataset and a small network. Moreover, the DDP mode did not utilize multiple physical GPUs. Future work should repeat the experiments on real image datasets (e.g., CIFAR-10) with true multi-GPU hardware. Additionally, only training loss was measured; test accuracy and generalization performance were not examined. The effect of learning rate on overfitting should also be investigated.

4. Conclusion

This study systematically compared single-GPU and DDP-based distributed training under five learning rate configurations. The results clearly show that the learning rate is the main factor affecting convergence. Moderate rates (0.01–0.1) offer the best balance between speed and stability. When both modes converge, their final losses are similar – meaning DDP does not inherently hurt model quality. However, distributed training is slightly more sensitive to learning rate changes, especially near the upper end of the stable range, where the loss curves become noisier.

DOI: 10.54254/2755-2721/2026.33679

Because the synthetic dataset has a smooth loss landscape, it is still possible to converge even at an extreme learning rate of 1.0 – a scenario that would almost certainly cause divergence in real-world image classification tasks. This highlights that learning rate selection is problem-dependent, and small-scale synthetic experiments are useful for quickly exploring hyperparameter ranges.

Of course, this work has limitations. Our DDP mode did not actually use multiple physical GPUs, so this paper did not capture true gradient synchronization effects. Future work should repeat these experiments on real multi-GPU hardware with larger batch sizes, use real image datasets like CIFAR-10 or ImageNet, and also measure test accuracy. The interaction between learning rate and other hyperparameters also deserves further investigation.

References

[1] Yoshida, N., Nakakita, S., & Imaizumi, M. (2024). Effect of random learning rate: Theoretical analysis of SGD dynamics in non-convex optimization via stationary distribution. In International Conference on Machine Learning (ICML).

[2] Kulaye, M. S., Barve, T. D., & Samant, A. Y. (2025). A comparative study of optimization algorithms in deep learning: SGD, Adam, and beyond. PhilArchive.

[3] Ge, R., Kakade, S. M., Kidambi, R., & Netrapalli, P. (2019). The step decay schedule: A near optimal, geometrically decaying learning rate procedure for least squares. In Advances in Neural Information Processing Systems (NeurIPS) (Vol. 32).

[4] Yu, S., Chen, W., & Poor, H. V. (2024). Distributed stochastic gradient descent with staleness: A stochastic delay differential equation based framework. arXiv preprint, arXiv: 2406.11159.

[5] Khaled, A., et al. (2025). Understanding outer learning rates in Local SGD. In Conference on Neural Information Processing Systems (NeurIPS).

[6] Zhao, R., Morwani, D., Brandfonbrener, D., & Vyas, N. (2024). Anything but SGD: Evaluating optimizers for LLM training. Kempner Institute, Harvard University.

[7] Toumpis, S., et al. (2024). Optimizer selection in NLP: Tuning beats architecture. In Annual Meeting of the Association for Computational Linguistics (ACL).

[8] Loss landscape dependent self-adjusting learning rates in decentralized stochastic gradient descent. (2023). In International Conference on Learning Representations (ICLR).

[9] Aach, M., Inanc, E., Sarma, R., Riedel, M., & Lintermann, A. (2023). Large scale performance analysis of distributed deep learning frameworks for convolutional neural networks. *Journal of Big Data, 10(1),* 1–28. https://doi.org/10.1186/s40537-023-00765-w

[10] Lyu, K. (2024). SDE approximations and scaling rules for distributed deep learning [Talk presentation]. Simons Institute for the Theory of Computing, University of California, Berkeley.