Distributed Training in PyTorch

Deven Mahesh Mistry
demistry@iu.edu
Luddy School of Informatics, Computer Science and Engineering
May 8th, 2024

Abstract

We train a 38M-parameter TinyTransformer model from scratch on the WikiText-2 dataset using PyTorch's DistributedDataParallel (DDP) across 4x NVIDIA H100 GPUs. By varying batch sizes and sequence lengths, we study their impact on training convergence and gradient synchronization. This work offers practical insights into the dynamics of distributed language model training and serves as a reproducible reference for understanding small-scale transformer training in multi-GPU settings.

I. Introduction

Large language models (LLMs) like GPT-4 [1] and Claude [2] have rapidly transformed the landscape of AI applications, finding use across domains such as education, healthcare, and software development. Despite their success, most state-of-the-art models are closed source, offering little insight into their training regimes, dataset composition, or computational requirements.

In contrast, open-source models such as Gemma [3], Phi [4], and LLaMA [5] have enabled researchers and practitioners to delve deeper into the architecture and training dynamics of LLMs. However, building a model that competes with top-tier LLMs requires massive amounts of compute, high-quality data, and significant engineering resources—barriers that are prohibitive for most individuals and small research teams.

This project adopts a simplified yet instructive approach: training a 38-million-parameter Transformer-based [6] LLM from scratch on the WikiText-2 [7] dataset. The primary goal is not to achieve competitive performance, but to understand practical aspects of distributed training—such as multi-GPU synchronization, the role of gradient aggregation, and the impact of hyperparameters like batch size and sequence length on convergence behavior.

II. Methodology

The training configuration was intentionally kept simple to highlight the dynamics of distributed training. A custom TinyTransformer model was implemented with 5 Transformer layers, each using 5 attention heads and a hidden size of 2048. To reduce the number of trainable parameters, the word embedding dimensionality was lowered to 300 (as opposed to a more typical 1024) from GPT-2 [8].

The model was trained from scratch on the WikiText-2 [7] dataset, which contains approximately 2.45 million tokens in its training split. Tokenization was performed using the GPT-2 tokenizer [9], resulting in a vocabulary size of 50,257 tokens.

To study the effect of hyperparameters on model convergence, training was conducted across different batch sizes (8, 16, 32, 64, 128) and sequence lengths (64, 128, 512, 1024, 2048). Distributed training was enabled using PyTorch's DistributedDataParallel (DDP) framework [10]. All experiments were executed on a system with 4X 80GB NVIDIA H100 GPUs.

During distributed training, PyTorch's c10d backend (ProcessGroupNCCL) is used to initialize communication between the GPU processes. Each GPU (also referred to as a "replica" or "rank") receives a copy of the model and is assigned a distinct shard of the input data. For instance, with a configuration of batch size = 8 and sequence length = 64, each GPU processes $8 \times 64 = 512$ tokens per forward pass. With 4 GPUs, this results in a total of 2048 tokens processed in parallel per step.

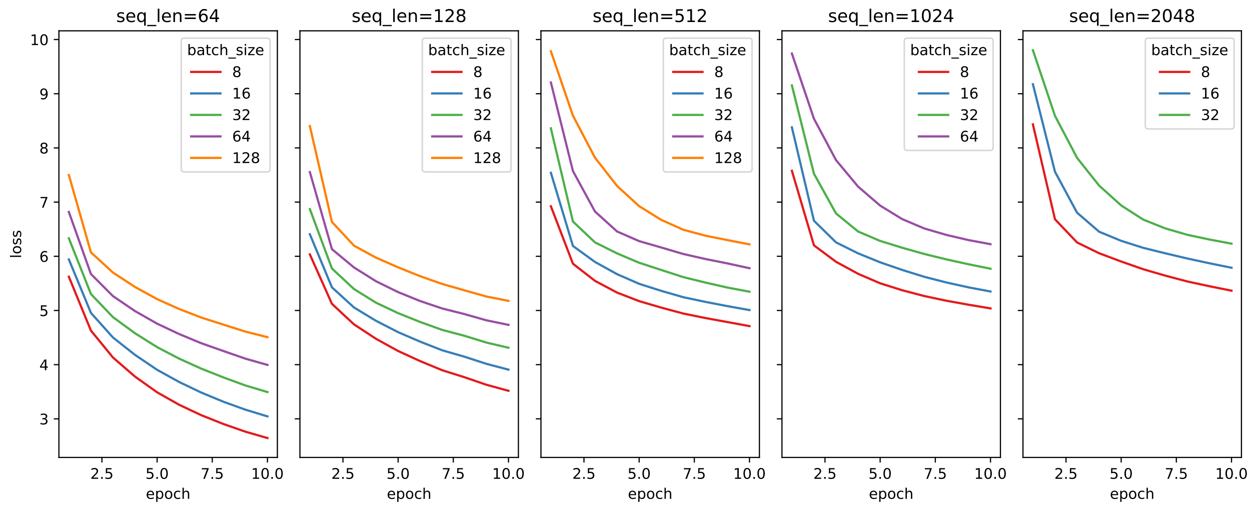
After the forward pass, each GPU computes gradients independently during the backward pass. At this point, DDP performs an all-reduce operation across the process group: each GPU broadcasts its gradients to all others and receives theirs in return. The gradients are then averaged (or summed and divided) across replicas to ensure consistent model updates. This synchronization step is performed layer by layer and is critical to maintaining model consistency during training.

This collective communication, handled via the ProcessGroup API (usually backed by NCCL in GPU-based training), introduces communication overhead, especially as model size or the number of participating GPUs increases. Understanding how this gradient synchronization scales — and how it is affected by factors like batch size and sequence length — is one of the key learning objectives of this project.

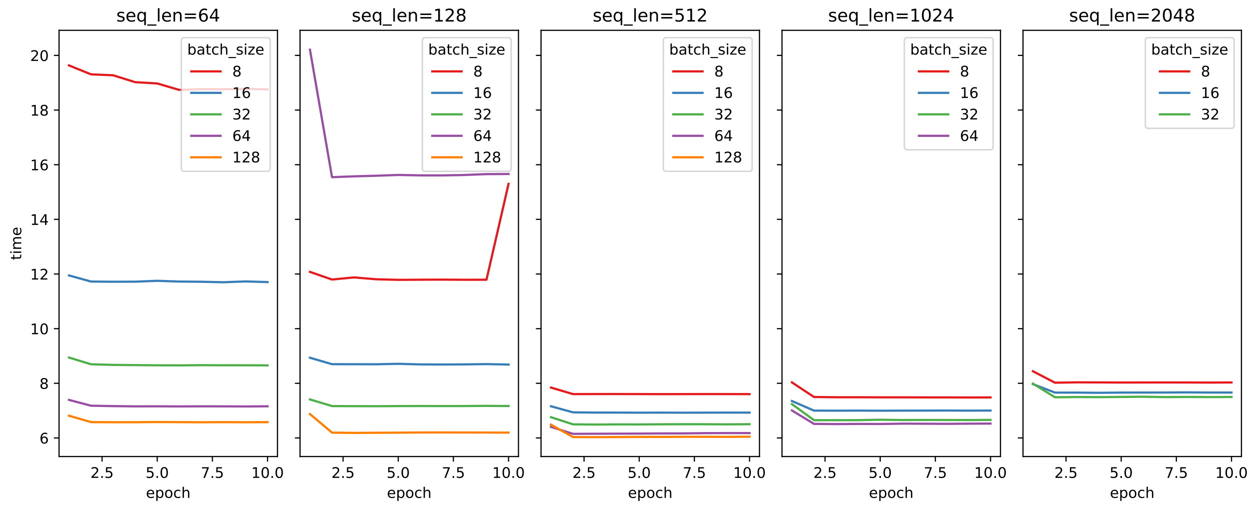
III. Results

The experiments systematically evaluated the effect of batch size and sequence length on model convergence and distributed training efficiency. The unit of the 'time' column in Table.I is seconds per epoch. The results for the configurations (128, 1024), (64, 2028) and (128, 2048), where the first value in the tuple is the batch size and the next value is sequence length, don't exist as that configuration ran into a CUDA Out of Memory error.

- Perplexity Trends
  - From Table I, the graphs indicate that perplexity increases with sequence length across all batch sizes. For example, with a batch size of 8, perplexity rises from approximately



(a) Loss vs Sequence Length



(b) Time vs Sequence Length

Fig. 1: Comparison of loss and time with sequence length

89 at a sequence length of 64 to over 1100 at a sequence length of 2048.

  - Larger batch sizes tend to yield lower perplexity at shorter sequence lengths, but this benefit diminishes as sequence length increases. At high sequence lengths (1024 and above), perplexity remains high across all batch sizes.

- Batch Size Effects
  - From Fig. Fig.1a and 1b, we observe a consistent trend where shorter sequences and shorter batch sizes yields in a lower loss. However, synchronizing smaller batches requires a greater communication overhead. Thus the combination with the lowest loss and perplexity, takes the longest time to run.
  - For longer sequences, the impact of batch size on perplexity lessens, with all batch sizes converging to higher perplexity values.
- Training Time
  - Training time per epoch generally decreases as batch size increases and as sequence length decreases, which is expected due to more efficient parallelization and smaller computational graphs as seen in Fig.1b.
  - However, the reduction in time is not always linear, reflecting the communication overhead introduced by distributed gradient synchronization.

IV. Discussion

- Scaling and Convergence

- The findings highlight the trade-off between batch size, sequence length, and model convergence in distributed training. Larger batch sizes facilitate faster convergence and lower perplexity for shorter sequences, likely due to more stable gradient estimates and efficient GPU utilization. However, as sequence length increases, the model struggles to maintain low perplexity, regardless of batch size. This suggests that for this model and dataset, there are diminishing returns to simply increasing batch size when also increasing sequence length. The model requires a lot of training samples to converge at larger sequence lengths.

- Distributed Communication Overhead
  - The observed training times reflect the cost of collective communication in PyTorch's DistributedDataParallel (DDP) framework. As batch size increases, the per-step communication cost is amortized over more data, but for longer sequences, the larger computational graphs and increased memory usage can offset these gains. The all-reduce operation, which synchronizes gradients across all GPUs, becomes a bottleneck at higher sequence lengths due to increased data transfer requirements.
- Practical Implications
  - For practitioners, these results suggest
    * Use larger batch sizes to reduce perplexity and training time, but only up to the point where GPU memory and communication overhead allow.
    * Avoid excessively long sequence lengths unless necessary for the task, as they significantly degrade perplexity and increase training time.
    * Monitor the scaling efficiency of distributed training, especially when increasing both batch size and sequence length, to ensure that the benefits of parallelism are not outweighed by communication costs.

Acknowledgment

This research was supported in part by Lilly Endowment, Inc., through its support for the Indiana University Pervasive Technology Institute.

V. Conclusion

The experiments demonstrate that both batch size and sequence length critically influence model convergence and distributed training efficiency. The interplay between these hyperparameters is nontrivial: larger batch sizes help, but cannot fully compensate for the challenges of training with long sequences. Communication overhead in distributed setups further complicates scaling, emphasizing the need for careful tuning and monitoring in large-scale language model training.

TABLE I: Training Results for Different Configurations

batch_size	seq_len	loss	perplexity	time
8	64	**3.63**	**88.97**	19.00
8	128	4.35	156.20	12.18
8	512	5.32	411.45	7.63
8	1024	5.68	660.87	7.54
8	2048	6.11	1129.89	8.07
16	64	4.02	135.58	11.74
16	128	4.71	238.01	8.73
16	512	5.66	649.03	6.95
16	1024	6.09	1097.90	7.04
16	2048	6.61	1891.85	7.69
32	64	4.43	210.38	8.69
32	128	5.08	372.61	7.19
32	512	6.08	1071.76	6.53
32	1024	6.60	1863.50	6.72
32	2048	7.26	3458.51	7.57
64	64	4.88	342.00	7.19
64	128	5.51	631.55	16.07
64	512	6.61	1947.08	6.19
64	1024	7.24	3296.77	6.57
128	64	5.37	589.94	6.60
128	128	5.99	1073.11	6.28
128	512	7.25	3411.32	**6.08**

References

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] Anthropic, "The claude 3 model family: Opus, sonnet, haiku," *Anthropic: Documentation*, 2024.

[3] G. Team, A. Kamath, J. Ferret, S. Pathak, N. Vieillard, R. Merhej, S. Perrin, T. Matejovicova, A. Ramé, M. Rivière *et al.*, "Gemma 3 technical report," *arXiv preprint arXiv:2503.19786*, 2025.

[4] M. Abdin, J. Aneja, H. Awadalla, A. Awadallah, A. A. Awan, N. Bach, A. Bahree, A. Bakhtiari, J. Bao, H. Behl *et al.*, "Phi-3 technical report: A highly capable language model locally on your phone," *arXiv preprint arXiv:2404.14219*, 2024.

[5] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[7] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[8] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[9] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-Art Natural Language Processing." Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[10] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. Luk, B. Maher, Y. Pan, C. Puhrsch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, M. Suo, P. Tillet, E. Wang, X. Wang, W. Wen, S. Zhang, X. Zhao, K. Zhou, R. Zou, A. Mathews, G. Chanan, P. Wu, and S. Chintala, "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation," in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024. [Online]. Available: https://pytorch.org/assets/pytorch2-2.pdf