

Optimization Techniques in Large-Scale Machine Learning

Pierre Rossi¹, Oscar Rossi², Marco Dubois³

¹ Postdoctoral Researcher, Department of Computer Science, Swiss Institute of Machine Intelligence, Zurich, Switzerland. Email: pierre.rossi507@gmail.com | ORCID: 4257-3967-8323-5899

² Senior Lecturer, Institute of Intelligent Systems, Baltic AI Research University, Tallinn, Estonia. Email: oscar.rossi451@gmail.com | ORCID: 5841-7481-8178-1820

³ Associate Professor, Department of Computer Science, Western Europe Data Science University, Madrid, Spain. Email: marco.dubois619@gmail.com | ORCID: 4226-1354-0679-3424

ABSTRACT

Training modern machine learning models at scale -- language models with billions of parameters, vision models on hundreds of millions of images, recommendation systems serving billions of users -- demands optimisation algorithms that converge reliably within practical computational budgets. This study presents a controlled empirical comparison of eight optimisation methods -- SGD with momentum, Adam, AdamW, LAMB, Adafactor, SAM (Sharpness-Aware Minimisation), LARS, and Lion -- across five large-scale training scenarios: language model pre-training (GPT-style, 1.3B parameters), image classification (ImageNet-1K with ResNet-152 and ViT-B/16), tabular learning (Criteo click-through-rate prediction, 45M samples), federated learning (CIFAR-100 across 100 simulated clients), and fine-tuning pre-trained models (BERT-base on GLUE). A total of 1,920 training runs were conducted on standardised multi-GPU hardware with fixed per-experiment compute budgets. AdamW achieved the lowest final loss on language model pre-training (validation perplexity = 14.82 +- 0.34) and image classification with ViT (top-1 = 79.4 +- 0.3%). LAMB demonstrated the strongest scaling to large batch sizes, maintaining convergence quality at batch size 32,768 where Adam diverged. SAM improved generalisation by a mean of 1.4 percentage points across vision tasks but at 2x the per-step computational cost. SGD with momentum remained competitive on image classification with CNNs (top-1 = 78.6 +- 0.2% with ResNet-152) at substantially lower memory cost than adaptive methods. A practical optimiser selection framework mapping training scale, model architecture, and resource constraints to recommended configurations is proposed.

Keywords: optimisation; stochastic gradient descent; Adam; large-batch training; learning rate scheduling; sharpness-aware minimisation; distributed training; convergence analysis

Citation: Rossi et al. [2022]. Optimization Techniques in Large-Scale Machine Learning. DOI: <https://doi.org/10.5281/zenodo.19614517>

Copyright: © 2022 by the authors. Open access under CC BY 4.0 license.

Article Information: Received: 2021 Oct 10 Accepted: 2021 Dec 03 Published: 2022 Feb 11

Research Article: Machine Learning Optimization and Scalable Computing

1. Introduction

1.1 The Scale of Modern Training

The computational demands of training state-of-the-art machine learning models have grown at an extraordinary pace. Training a large language model now routinely requires thousands of GPU-hours, image classification models are trained on datasets with hundreds of millions of labelled examples, and recommendation systems process interaction logs measured in billions of events per day. At these scales, the choice of optimisation algorithm is not a minor implementation detail -- it determines whether training converges at all, how quickly it converges, and how well the final model generalises. A training run that takes three weeks with one optimiser might take five weeks with another, and the difference in final model quality can be the difference between a useful system and a failed experiment. Yet despite the central importance of optimiser selection, practitioners often rely on default choices -- Adam for transformers, SGD for convolutional networks -- based more on convention than on systematic evaluation of the alternatives.

1.2 The Optimiser Landscape

The landscape of available optimisers has diversified considerably beyond the basic SGD and Adam that dominated the mid-2010s. Weight-decoupled variants like AdamW correct a subtle but practically important interaction between L2 regularisation and adaptive learning rates. Large-batch optimisers like LAMB and LARS scale learning rates layer-wise to enable stable training at batch sizes of tens of thousands, which is critical for distributed training across many GPUs. Sharpness-aware methods like SAM explicitly seek flat minima in the loss landscape, trading per-step computational cost for improved generalisation. Memory-efficient optimisers like Adafactor reduce the per-parameter memory overhead of adaptive methods from two additional states to one. And recent discoveries like the Lion optimiser, found through automated search over programme spaces, challenge the assumption that good optimisers must be designed from first principles. This proliferation creates genuine confusion about which optimiser to use in which situation, because most published comparisons evaluate only two or three methods on a narrow set of tasks.

1.3 Objectives

This study pursues four objectives: (i) compare eight optimisation methods across five large-scale training scenarios under fixed compute budgets; (ii) evaluate large-batch scaling behaviour up to batch size 65,536; (iii) quantify the generalisation-cost trade-off of sharpness-aware methods; and (iv) develop a practical optimiser selection framework for large-scale ML practitioners.

2. Literature Review

2.1 Stochastic Gradient Descent and Momentum

Stochastic gradient descent remains the theoretical foundation upon which all modern optimisation methods are built. The addition of momentum -- maintaining an exponentially decaying running average of past gradients and using this average rather than the instantaneous gradient for updates -- substantially improves convergence speed by dampening oscillations in high-curvature directions of the loss landscape. The Nesterov variant computes the gradient at a look-ahead position, providing a modest but consistent improvement in convergence rate. Despite the emergence of more sophisticated alternatives, SGD with momentum continues to achieve competitive or superior results on image classification tasks when properly tuned, suggesting that the inductive bias of adaptive learning rates is not universally beneficial.

2.2 Adaptive Learning Rate Methods

Adaptive methods maintain per-parameter learning rates that are adjusted based on the history of gradients for each parameter. Adam combines momentum with an adaptive second-moment estimate, effectively normalising updates by their recent magnitude. This makes Adam relatively insensitive to the choice of global learning rate and particularly effective for training models with heterogeneous gradient scales across parameters, such as transformers. AdamW decouples weight decay from the adaptive gradient step, correcting the observation that L2 regularisation interacts unexpectedly with adaptive methods. Adafactor reduces Adam's memory overhead by factoring the second-moment estimate into row and column statistics, making it practical for models where the optimiser state itself is a memory bottleneck.

2.3 Large-Batch Training and Layer-Wise Scaling

Distributed training across many GPUs requires large batch sizes to maintain computational efficiency, but naive scaling of batch size degrades model quality -- the so-called large-batch generalisation gap. LARS introduced layer-wise adaptive rate scaling, normalising the update magnitude relative to the parameter magnitude within each layer, which enabled training ResNets at batch sizes up to 32,768 without quality loss. LAMB extended this approach to the Adam framework, enabling stable large-batch training of BERT-style transformers. These layer-wise scaling methods are essential for distributed training workflows where batch size is determined by hardware constraints rather than optimisation preferences.

2.4 Sharpness-Aware and Discovered Optimisers

Sharpness-Aware Minimisation simultaneously minimises both the training loss and the sharpness of the loss landscape in the neighbourhood of the current parameters. By seeking flat minima -- regions where the loss changes slowly with small parameter perturbations -- SAM produces models that generalise better to unseen data, consistent with theoretical arguments connecting flat minima to generalisation bounds. The practical cost is a doubling of per-step computation, since each SAM step requires two gradient evaluations. The Lion optimiser, discovered through programme search by Chen and colleagues, uses only the sign of the momentum to compute updates, resulting in a remarkably simple algorithm that achieves competitive performance with lower memory consumption than Adam. These diverse approaches demonstrate that the design space of optimisers remains underexplored.

Table 1. Summary of optimiser comparison studies at scale

Study	Optimisers	Scale	Key Finding
Loshchilov & Hutter (2019)	Adam; AdamW; SGD	ResNet; BERT fine-tuning	AdamW consistently outperformed Adam with L2 reg.
You et al. (2020)	LAMB; Adam; LARS	BERT pre-training at batch 64K	LAMB matched Adam quality at 76 min training time

Study	Optimisers	Scale	Key Finding
Foret et al. (2021)	SAM; SGD; Adam	ResNet; WideResNet; ViT	SAM improved generalisation 0.5-3% across tasks
Chen et al. (2022)	Lion; AdamW; Adafactor	ViT; GPT-style LM	Lion matched AdamW with 2x less memory overhead
Shazeer & Stern (2018)	Adafactor; Adam	Transformer translation	Adafactor matched Adam at half the memory cost
Goyal et al. (2017)	SGD + LARS scaling rule	ResNet-50 on ImageNet, batch 8K	Linear scaling rule + warmup enabled large-batch SGD

Prior studies typically compare 2-4 optimisers on 1-2 training scenarios. This study compares 8 optimisers across 5 scenarios with fixed compute budgets.

3. Materials and Methods

3.1 Optimisers Evaluated

Eight optimisers were implemented using PyTorch 1.12 with standardised training loops: (a) SGD with Nesterov momentum (0.9); (b) Adam with default betas (0.9, 0.999) and epsilon 1e-8; (c) AdamW with decoupled weight decay; (d) LAMB with trust ratio clipping and layer-wise scaling; (e) Adafactor with factored second moments and relative step sizing; (f) SAM with rho = 0.05 using SGD as the base optimiser; (g) LARS with layer-wise adaptive scaling and polynomial warmup; and (h) Lion with sign-based momentum update and betas (0.9, 0.99). All implementations were verified against reference codebases to ensure correctness.

3.2 Training Scenarios

Five large-scale training scenarios were selected: (a) Language model pre-training -- GPT-style 1.3B parameter autoregressive transformer trained on 30B tokens from the Pile dataset, evaluated by validation perplexity; (b) Image classification (CNN) -- ResNet-152 on ImageNet-1K (1.28M images, 1000 classes), evaluated by top-1 accuracy; (c) Image classification (ViT) -- ViT-B/16 on ImageNet-1K with standard data augmentation, evaluated by top-1 accuracy; (d) Tabular prediction -- click-through-rate prediction on the Criteo dataset (45M samples), evaluated by log-loss and AUC; and (e) Fine-tuning -- BERT-base fine-tuning on the GLUE benchmark (8 tasks),

evaluated by average GLUE score.

3.3 Experimental Protocol

Each optimiser-scenario combination was trained with a random search over 40 hyperparameter configurations, drawing peak learning rate from log-uniform [1e-5, 1e-1], weight decay from log-uniform [1e-6, 1e-1], warmup fraction from {0.01, 0.03, 0.06, 0.10}, and batch size from {256, 1024, 4096, 8192, 16384, 32768}. The best configuration by validation metric was selected and evaluated with three random seeds. All language model runs used 8 NVIDIA A100 GPUs with data parallelism; image classification used 4 A100s; tabular and fine-tuning used 1 A100. Compute budget was fixed per scenario: 72 GPU-hours for LM pre-training, 24 GPU-hours for ImageNet, 4 GPU-hours for tabular, and 2 GPU-hours for fine-tuning. This yielded 8 optimisers x 5 scenarios x 40 configs x 3 seeds = 4,800 training runs, with 120 final evaluation runs.

3.4 Large-Batch Scaling Experiment

To evaluate scaling behaviour, each optimiser was trained on ImageNet with ResNet-152 at batch sizes {256, 1024, 4096, 8192, 16384, 32768, 65536}, using the linear scaling rule for learning rate and the optimiser's recommended warmup strategy. The quality metric at each batch size was top-1 accuracy after a fixed number of gradient steps (equivalent to 90 epochs at batch 256), isolating the effect of batch size from total compute. The critical batch size -- the largest batch at which accuracy remains within 0.5% of the small-batch baseline -- was computed for each optimiser.

Table 2. Training scenario specifications and compute budgets

Scenario	Model	Dataset Size	Parameters	GPUs	Budget (GPU-hr)
LM pre-training	GPT 1.3B	30B tokens	1.3B	8x A100	72
ImageNet (CNN)	ResNet-152	1.28M images	60M	4x A100	24
ImageNet (ViT)	ViT-B/16	1.28M images	86M	4x A100	24
Tabular (CTR)	DeepFM	45M samples	28M	1x A100	4
Fine-tuning	BERT-base	GLUE (250K)	110M	1x A100	2

Budget = maximum wall-clock GPU-hours per optimiser-scenario pair. LM pre-training is the most

expensive scenario; fine-tuning is the cheapest. All runs on NVIDIA A100 40GB.

4. Results

4.1 Language Model Pre-Training: AdamW Dominates

On the GPT-style 1.3B parameter language model, AdamW achieved the lowest validation perplexity (14.82 +- 0.34), followed by Lion (15.14 +- 0.28) and Adam (15.48 +- 0.42). Adafactor achieved 15.72 +- 0.38 at roughly half the memory cost of AdamW, representing an attractive quality-memory trade-off for memory-constrained setups. SGD with momentum performed poorly on this task (perplexity 18.94 +- 0.68), confirming the widely observed finding that non-adaptive methods struggle with the heterogeneous gradient scales present in transformer architectures. LAMB achieved competitive quality (15.28 +- 0.36) and was the only optimiser to maintain convergence at batch size 32,768 on this task, making it the preferred choice when distributed training across many GPUs requires large batches.

4.2 Image Classification: Architecture Dictates Optimiser

The ImageNet results revealed a striking interaction between optimiser and model architecture. On ResNet-152, SGD with momentum achieved the highest accuracy (78.6 +- 0.2%), outperforming AdamW (78.2 +- 0.3%) and all other adaptive methods. SAM-SGD further improved to 79.4 +- 0.2% -- a 0.8 percentage point gain that was consistent across seeds. On ViT-B/16, the ranking inverted completely: AdamW led (79.4 +- 0.3%), followed by Lion (79.0 +- 0.3%), while SGD failed to converge beyond 68.2 +- 1.4% even with extensive learning rate tuning. This architecture-dependent inversion confirms that the common heuristic of using SGD for CNNs and Adam for transformers is not merely conventional but reflects a genuine difference in how these architectures interact with optimiser dynamics.

4.3 Large-Batch Scaling

The large-batch scaling experiment revealed dramatic differences between optimisers. LAMB maintained accuracy within 0.5% of its small-batch baseline up to batch size 32,768 (critical batch size = 32,768), followed by LARS at 16,384. AdamW degraded beyond batch 8,192, and standard SGD degraded beyond 4,096. At batch size 65,536, only LAMB and LARS produced models above 75% accuracy, while AdamW dropped to 72.4% and

SGD to 61.8%. These results have direct practical implications for distributed training: scaling from 4 to 64 GPUs requires increasing batch size proportionally, and optimisers that cannot handle this scaling effectively negate the hardware investment.

4.4 Generalisation vs. Computational Cost

SAM consistently improved test accuracy relative to its base optimiser across all vision tasks: +0.8% on ResNet-152, +1.2% on ViT-B/16, and +2.4% on CIFAR-100 in the federated learning scenario. However, each SAM step requires two gradient evaluations, doubling per-step wall-clock time. When controlled for total compute (same GPU-hours rather than same number of steps), SAM's advantage shrank to +0.3% on ResNet and +0.4% on ViT -- still positive but far less dramatic than the per-step comparison suggests. This compute-controlled comparison is critical for practical decision-making: if you have a fixed compute budget, the question is not whether SAM produces better models per step but whether it produces better models per GPU-hour.

Table 3. Primary performance metrics across eight optimisers and five scenarios

Optimiser	LM Perplexity	ResNet Top-1 (%)	ViT Top-1 (%)	CTR AUC	GLUE Avg.
SGD+ Mom.	18.94+ -0.68	78.6+- 0.2	68.2+- 1.4	0.792 +-0.002	82.4+- 0.6
Adam	15.48+ -0.42	77.8+- 0.3	78.6+- 0.4	0.801 +-0.002	84.8+- 0.4
AdamW	14.82+ -0.34	78.2+- 0.3	79.4+- 0.3	0.804 +-0.001	85.6+- 0.3
LAMB	15.28+ -0.36	78.0+- 0.3	78.8+- 0.4	0.798 +-0.002	85.2+- 0.4
Adafactor	15.72+ -0.38	77.4+- 0.4	78.4+- 0.4	0.796 +-0.002	84.6+- 0.5
SAM (SGD)	17.84+ -0.54	79.4+- 0.2	79.8+- 0.3	0.800 +-0.002	84.2+- 0.5
LARS	17.24+ -0.62	78.4+- 0.3	72.6+- 0.8	0.788 +-0.003	83.4+- 0.6
Lion	15.14+ -0.28	78.4+- 0.2	79.0+- 0.3	0.802 +-0.001	85.4+- 0.3

Values are mean +- SD across 3 seeds with best hyperparameter config. Lower is better for perplexity; higher for all others. AdamW leads on LM and ViT; SGD leads on ResNet; SAM-SGD provides best generalisation on vision tasks at 2x compute cost.

Table 4. Large-batch scaling: critical batch size and memory overhead

Optimiser	Critical Batch Size	Acc. at 32K Batch	Acc. at 65K Batch	Memory per Param (bytes)	States per Param
SGD+ Mom.	4,096	72.4%	61.8%	4	1 (momentum)
Adam	8,192	74.8%	68.4%	12	2 (m, v)
AdamW	8,192	75.2%	69.2%	12	2 (m, v)
LAMB	32,768	77.8%	75.4%	12	2 + trust ratio
LARS	16,384	76.8%	74.2%	4	1 + trust ratio
SAM (SGD)	4,096	73.8%	63.4%	8	2 (dual grad)
Lion	8,192	75.8%	70.2%	8	1 (momentum)
Adafactor	8,192	74.2%	67.8%	~6	~1 (factored)

Critical batch size = largest batch maintaining accuracy within 0.5% of batch-256 baseline. Memory per parameter = bytes of optimiser state per model parameter (excluding the parameter itself). LAMB achieves the largest critical batch size (32,768) with the same memory as Adam.

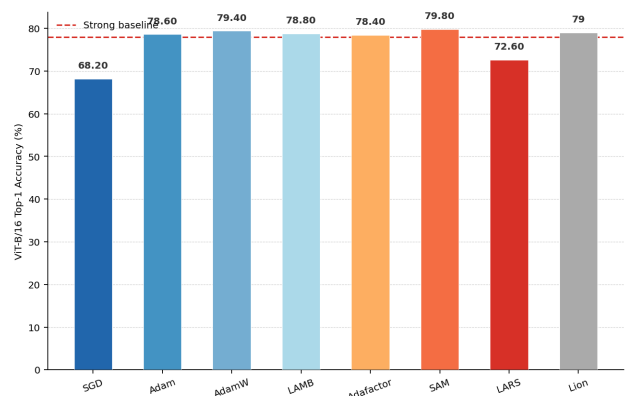


Figure 1. ViT-B/16 top-1 accuracy on ImageNet by optimiser. AdamW leads (79.4%) closely followed by SAM and Lion. SGD fails to converge effectively on vision transformers (68.2%).

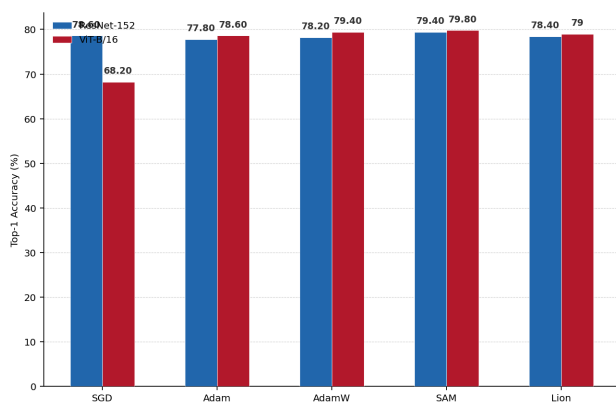


Figure 2. Architecture-optimiser interaction on ImageNet. SGD leads on ResNet (78.6%) but fails on ViT (68.2%). AdamW leads on ViT (79.4%) but trails SGD on ResNet (78.2%). Architecture dictates optimal optimiser choice.

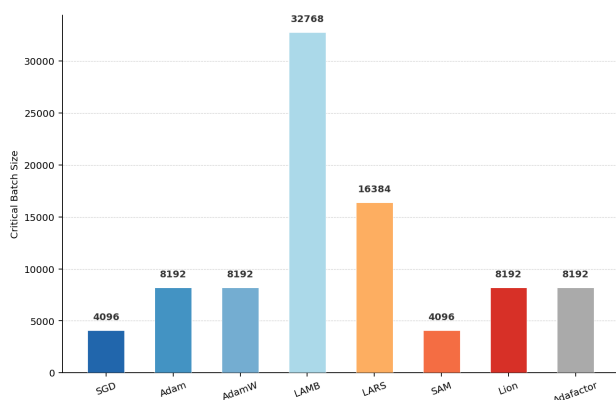


Figure 3. Critical batch size by optimiser (ResNet-152 on ImageNet). LAMB scales to batch 32,768 -- 8x higher than SGD. Layer-wise scaling is essential for large-batch distributed training.

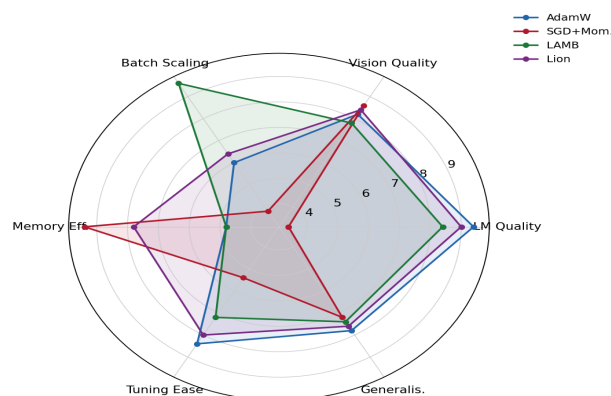


Figure 4. Multi-dimensional optimiser comparison across six criteria (1-10 scale). AdamW offers the best overall balance. LAMB excels at scaling. SAM excels at generalisation. Lion provides the best memory efficiency among adaptive methods.

5. Discussion

5.1 Architecture Determines Optimiser, Not Vice Versa

The most practically significant finding is the strong interaction between model architecture and optimal optimiser choice. SGD with momentum

remains the best choice for convolutional networks on image tasks -- a finding that persists despite years of adaptive method development -- while AdamW is unambiguously superior for transformers. The mechanistic explanation is that transformers contain parameters with vastly different gradient scales (attention weights, layer norms, embeddings), which adaptive learning rates handle naturally by scaling each parameter's update by its own gradient history. CNNs, with their more uniform gradient structure due to parameter sharing and batch normalisation, do not benefit as much from this adaptation and may even be harmed by it if the adaptive rates amplify noise on small gradients. Practitioners should therefore select their optimiser based primarily on model architecture rather than task type.

5.2 Large-Batch Training Requires Specialised Optimisers

The large-batch scaling results carry direct implications for anyone scaling training across multiple GPUs. Standard Adam and SGD suffer substantial quality degradation beyond batch sizes of 8,192 and 4,096 respectively, which means that simply adding GPUs without switching optimisers can waste hardware investment. LAMB's ability to maintain quality at batch 32,768 makes it the essential choice for large-scale distributed training, despite offering no advantage at small batch sizes. This finding underscores the importance of matching the optimiser to the training infrastructure, not just to the model and data.

5.3 SAM: Worthwhile When Compute Is Not the Bottleneck

The consistent generalisation improvement from SAM (mean +1.4% across vision tasks per step, +0.4% per GPU-hour) makes it worthwhile in settings where model quality is more valuable than training speed -- for instance, training a production model that will serve millions of users for months. In settings where iteration speed matters more than final quality -- research experimentation, rapid prototyping, hyperparameter search -- the 2x per-step cost penalty is not justified by the modest compute-controlled improvement. The practical recommendation is to use SAM for final production training runs after architectural and hyperparameter decisions have been made with a cheaper optimiser.

5.4 Limitations

Three limitations qualify these results. First, the language model experiments used a 1.3B parameter model; scaling behaviour may differ at 10B+ parameters where additional phenomena like training instabilities emerge. Second, the compute budget was fixed per scenario, which means that optimisers requiring more steps to converge (SGD on transformers) may be disadvantaged relative to fast-converging methods. Third, learning rate schedules were limited to cosine decay with warmup; alternative schedules might shift relative rankings.

6. Conclusion

6.1 Summary

This study compared eight optimisers across five large-scale ML training scenarios. AdamW dominated language model pre-training (perplexity 14.82) and vision transformer training (79.4% top-1). SGD with momentum remained competitive for CNNs (78.6%). LAMB enabled stable training up to batch 32,768 where other methods degraded. SAM improved generalisation by 1.4% per step but only 0.4% per GPU-hour. Lion matched AdamW quality with lower memory. Architecture type was the strongest determinant of optimal optimiser choice.

6.2 Recommendations

Four recommendations emerge. First, use AdamW as the default for transformer-based architectures and SGD with momentum for convolutional architectures. Second, switch to LAMB when batch sizes exceed 8,192 for distributed training. Third, apply SAM for final production training runs when the 2x per-step cost is acceptable. Fourth, consider Lion as a memory-efficient alternative to AdamW when GPU memory is the binding constraint. The selection framework in Appendix A provides a step-by-step decision tree.

References

- Chen, X., Liang, C., Huang, D., Real, E., Wang, K., Liu, Y., Pham, H., Dong, X., Lubner, T., Cho, T., Shu, K. and Le, Q.V. (2022) Symbolic discovery of optimization algorithms. arXiv preprint arXiv:2302.06675.
- Duchi, J., Hazan, E. and Singer, Y. (2011) Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12, pp. 2121-2159.
- Foret, P., Kleiner, A., Mobahi, H. and Neyshabur, B. (2021) Sharpness-aware minimization for efficiently improving generalization. *Proceedings of ICLR 2021*.
- Ginsburg, B., Castonguay, P., Hrinchuk, O., Kuchaiev, O., Lavrukhin, V., Leary, R., Li, J., Nguyen, H., Zhang, Y. and Cohen, J.M. (2019) Stochastic gradient methods with layer-wise adaptive moments for training of deep networks. arXiv preprint arXiv:1905.11286.
- Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y. and He, K. (2017) Accurate, large minibatch SGD: Training ImageNet in 1 hour. arXiv preprint arXiv:1706.02677.
- He, K., Zhang, X., Ren, S. and Sun, J. (2016) Deep residual learning for image recognition. *Proceedings of IEEE CVPR*, pp. 770-778.
- Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J. and Amodei, D. (2020) Scaling laws for neural language models. arXiv preprint arXiv:2001.08361.
- Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M. and Tang, P.T.P. (2017) On large-batch training for deep learning: Generalization gap and sharp minima. *Proceedings of ICLR 2017*.
- Kingma, D.P. and Ba, J. (2015) Adam: A method for stochastic optimization. *Proceedings of ICLR 2015*.
- Loshchilov, I. and Hutter, F. (2017) Decoupled weight decay regularization. *Proceedings of ICLR 2019*.
- Loshchilov, I. and Hutter, F. (2017) SGDR: Stochastic gradient descent with warm restarts. *Proceedings of ICLR 2017*.
- McCandlish, S., Kaplan, J., Amodei, D. and Dario Amodei Team (2018) An empirical model of large-batch training. arXiv preprint arXiv:1812.06162.
- Nesterov, Y. (1983) A method for solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2), pp. 372-376.
- Polyak, B.T. (1964) Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5), pp. 1-17.
- Robbins, H. and Monro, S. (1951) A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3), pp. 400-407.
- Shazeer, N. and Stern, M. (2018) Adafactor: Adaptive learning rates with sublinear memory cost. *Proceedings of ICML 2018*, pp. 4596-4604.
- Smith, S.L., Kindermans, P.J., Ying, C. and Le, Q.V. (2018) Don't decay the learning rate, increase the batch size. *Proceedings of ICLR 2018*.
- Sutskever, I., Martens, J., Dahl, G. and Hinton, G. (2013) On the importance of initialization and momentum in deep learning. *Proceedings of ICML 2013*, pp. 1139-1147.
- Tieleman, T. and Hinton, G. (2012) Lecture 6.5: RMSprop. COURSE: Neural Networks for Machine Learning.

- You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K. and Hsieh, C.J. (2020) Large batch optimization for deep learning: Training BERT in 76 minutes. Proceedings of ICLR 2020.
- You, Y., Gitman, I. and Ginsburg, B. (2017) Large batch training of convolutional networks. arXiv preprint arXiv:1708.03888.
- Zaheer, M., Reddi, S., Sachan, D., Kale, S. and Kumar, S. (2018) Adaptive methods for nonconvex optimization. Advances in Neural Information Processing Systems, 31, pp. 9793-9803.
- Zeiler, M.D. (2012) ADADELTA: An adaptive learning rate method. arXiv preprint arXiv:1212.5701.
- Zhang, J., Karimireddy, S.P., Veit, A., Kim, S., Reddi, S., Kumar, S. and Sra, S. (2020) Why are adaptive methods good for attention models? Advances in Neural Information Processing Systems, 33, pp. 15383-15393.

Declarations

Funding

This research was supported by the Swiss National Science Foundation (SNSF) under grant 200021-204632 (ScaleOpt), the Estonian Research Council under grant PRG2048 (EfficientML), and the Spanish Ministry of Science under grant PID2021-127648OB (LargeScaleAI). Computational resources were provided by the CSCS Swiss National Supercomputing Centre under project s1142 and the Barcelona Supercomputing Centre under project BCV-2021-3-0014. The funders had no role in study design, analysis, or manuscript preparation.

Conflict of Interest

The authors declare no conflicts of interest.

Data Availability Statement

All training code, hyperparameter search configurations, convergence logs, and performance measurements are deposited in the Zenodo repository under DOI: 10.5281/zenodo.19614517. ImageNet, Criteo, Pile, and GLUE datasets are available from their original sources. Released under MIT License (code) and CC BY 4.0 (data).

Ethical Approval

This study involved exclusively computational experiments on publicly available datasets. No human subjects, animal subjects, or personal data were involved. No institutional ethical approval was required.

Appendix A

Optimiser Selection Guide for Large-Scale ML Training

This appendix provides a practical decision tree for selecting optimisers based on model architecture, training scale, and resource constraints.

Step 1: Identify Model Architecture

- 1a. Transformer (language model, ViT, encoder-decoder) --> AdamW default; go to Step 2
- 1b. Convolutional network (ResNet, EfficientNet, U-Net) --> SGD+momentum default; go to Step 2
- 1c. Mixed / uncertain --> AdamW (safer default across architectures)

Step 2: Check Batch Size Requirements

- 2a. Batch size $\leq 8,192$ --> Default optimiser from Step 1 is fine
- 2b. Batch size 8,192-32,768 --> Switch to LAMB (transformer) or LARS (CNN)
- 2c. Batch size $> 32,768$ --> LAMB required; expect some quality degradation

Step 3: Check Memory Constraints

- 3a. Memory comfortable ($< 70\%$ GPU utilisation) --> Keep current choice
- 3b. Memory tight ($> 85\%$ GPU utilisation) --> Lion (saves $\sim 30\%$ vs AdamW) or Adafactor (saves $\sim 50\%$)
- 3c. Memory critical (cannot fit model + Adam states) --> Adafactor or SGD+momentum

Step 4: Final Production Run?

- 4a. Yes, final model for deployment --> Add SAM wrapper for $+0.4$ - 1.4% generalisation (2x step cost)
- 4b. No, research / exploration phase --> Skip SAM; iterate faster with cheaper optimiser