Distributed Data-Parallel Training of Neural Networks At-Scale Using Distributed Shampoo

Jose Gallego-Posada

Meta, Mila and University of Montreal

September 27, 2023

Meta AI

arXiv > cs > arXiv:2309.06497

Computer Science > Machine Learning

[Submitted on 12 Sep 2023]

A Distributed Data-Parallel PyTorch Implementation of the Distributed Shampoo Optimizer for Training Neural Networks At-Scale

Hao-Jun Michael Shi, Tsung-Hsien Lee, Shintaro Iwasaki, Jose Gallego-Posada, Zhijing Li, Kaushik Rangadurai, Dheevatsa Mudigere, Michael Rabbat

Shampoo is an online and stochastic optimization algorithm belonging to the AdaGrad family of methods for training neural networks. It constructs a block-diagonal preconditioner where each block consists of a coarse Kronecker product approximation to full-matrix AdaGrad for each parameter of the neural network. In this work, we provide a complete description of the algorithm as well as the performance optimizations that our implementation leverages to train deep networks at-scale in PyTorch. Our implementation enables fast multi-GPU distributed data-parallel training by distributing the memory and computation associated with blocks of each parameter via PyTorch's DTensor data structure and performing an AllGather primitive on the computed search directions at each iteration. This major performance enhancement enables us to achieve at most a 10% performance reduction in per-step wall-clock time compared against standard diagonal-scaling-based adaptive gradient methods. We validate our implementation by performing an ablation study on training ImageNet ResNet50, demonstrating Shampoo's superiority over standard training recipes with minimal hyperparameter tuning.

Check out our open-source implementation:

https://github.com/facebookresearch/optimizers/tree/main/distributed_shampoo

Main contributions

Characterization of Distributed Shampoo

- Complete algorithmic characterization, **consolidating insights from recent literature**
- Including LR grafting and other as well as important deep learning heuristics

Open-source PyTorch Implementation

Performance optimizations required to ensure **Shampoo is competitive in terms of wall-clock time** compared to popular diagonal adaptive methods like Adagrad/Adam

Experimental evidence in large models

- Corroborating Shampoo's improvement in convergence and model quality w.r.t. benchmark training recipes
- On ImageNet task with ResNet50 models, **Shampoo yields a 1.35x improvement in wall-clock time**

Collaborators

Hao-Jun Michael Shi

Tsung-Hsien Lee

Shintaro Iwasaki

Zhijing Li

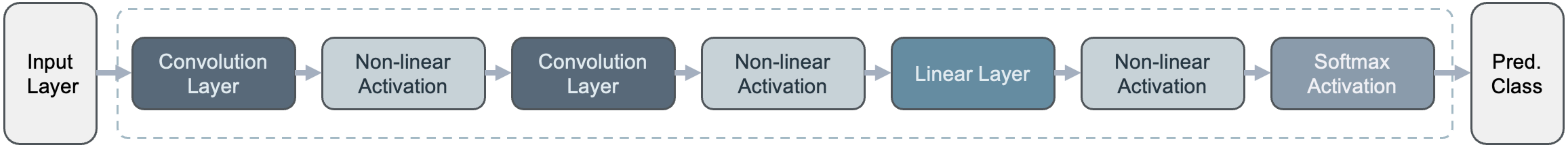
Kaushik Rangadurai

Dheevatsa Mudigere

Mike Rabbat

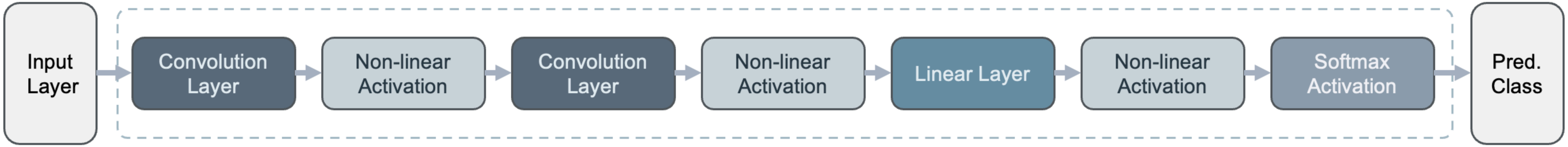
Acknowledgements to Vineet Gupta and Rohan Anil (and collaborators) for their algorithmic contributions in the original development of Shampoo.

Deep neural networks in one slide

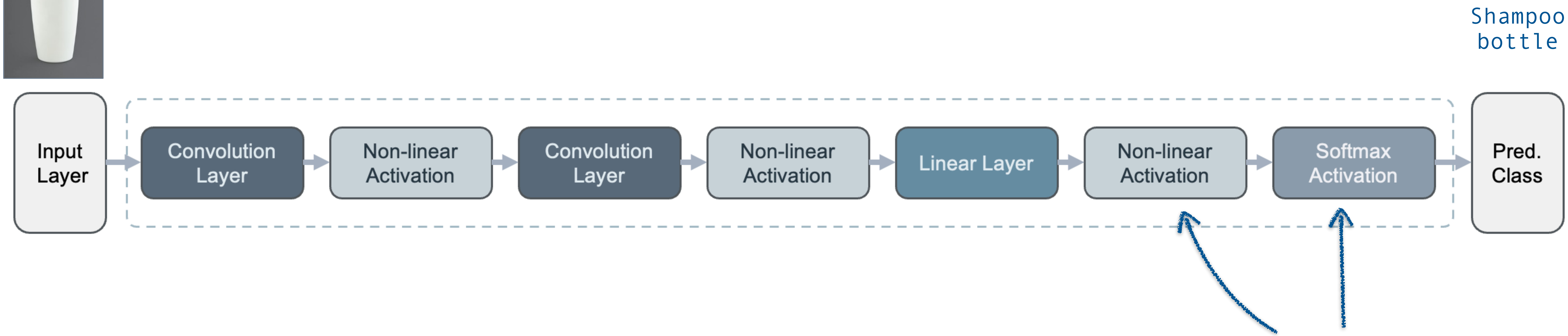


Deep neural networks in one slide

Shampoo
bottle



Deep neural networks in one slide



Modules are composable and differentiable

Deep neural networks in one slide

$$h(x; w) = \text{softmax}(W^{(l)}\sigma(W^{(n-1)}\sigma(\ldots\ \sigma(W^{(1)}x)\ \ldots)))$$

Shampoo bottle

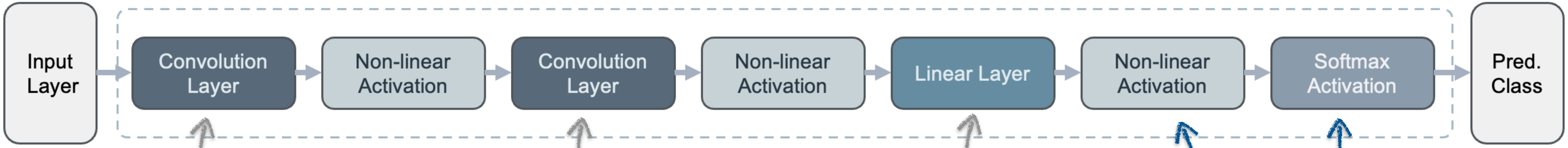
Input Layer → Convolution Layer → Non-linear Activation → Convolution Layer → Non-linear Activation → Linear Layer → Non-linear Activation → Softmax Activation → Pred. Class

Modules are composable and differentiable

Deep neural networks in one slide

$$h(x; w) = \text{softmax}(W^{(l)}\sigma(W^{(n-1)}\sigma(\ldots\ \sigma(W^{(1)}x)\ \ldots)))$$

Shampoo bottle



$W^{(1)} \in \mathbb{R}^{3\times5\times5\times64}$

$W^{(2)} \in \mathbb{R}^{64\times3\times3\times64}$

$W^{(3)} \in \mathbb{R}^{512\times1000}$

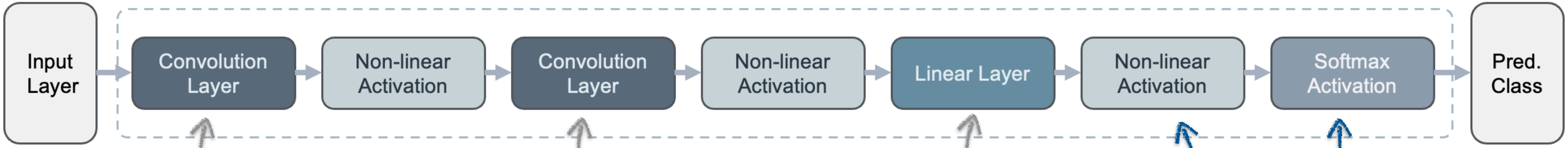
Modules are composable and differentiable

Trainable parameters: $w = \left(\text{vec}\left(W^{(1)}\right)^{\top}, \ldots, \text{vec}\left(W^{(l)}\right)^{\top}\right)^{\top} \in \mathbb{R}^{d}$

Deep neural networks in one slide

Shampoo bottle

$$h(x; w) = \mathrm{softmax}(W^{(l)}\sigma(W^{(n-1)}\sigma(\ldots\ \sigma(W^{(1)}x)\ \ldots)))$$



$W^{(1)} \in \mathbb{R}^{3\times5\times5\times64}$ $W^{(2)} \in \mathbb{R}^{64\times3\times3\times64}$ $W^{(3)} \in \mathbb{R}^{512\times1000}$

Modules are composable and differentiable

Trainable parameters: $w = \left(\mathrm{vec}\left(W^{(1)}\right)^{\top}, \ldots, \mathrm{vec}\left(W^{(l)}\right)^{\top}\right)^{\top} \in \mathbb{R}^{d}$

Modern architectures typically comprise between a few million to 100x billion parameters/variables!

Neural network training

Data

$$\{(x_i, y_i)\}_{i=1}^{N} = \{\xi_i\}_{i=1}^{N} \sim \mathcal{D}$$

Model

$$h(x; w) \mapsto \hat{y}$$

Loss function

$$\ell(\hat{y}, y)$$

Sampled function

$$f_\xi(w) = \ell(h(x; w), y)$$

- Large number of training samples, requiring the use of stochastic approximations
- Unlike traditional optimization, true goal is **generalization** to unseen examples
- DL optimization practice is dominated by *adaptive first-order methods* (like SGD+momentum, Adam, Adagrad)
- In the DL setting, complex optimization methods require engineering work to achieve performant implementations
- **Training faster is ideal as it allows to saves money and energy**

Preconditioned gradient methods

Generic PG Method

$$P_k = \text{UpdatePreconditioner}(P_{k-1}, g_k)$$
$$w_{k+1} = w_k - \alpha_k P_k g_k$$

Three core streams of work amongst the vast literature:

- **Newton and quasi-Newton methods ⇒ L-BFGS; K-BFGS**
- **Natural gradient ⇒ KFAC**
- **Adaptive gradient methods**
  - Most traction in DL practice
  - Widespread use of methods like Adagrad/Adam(W)
  - Strong pragmatic component behind our focus on this branch

Adaptive Gradient (Adagrad) Methods

Let us consider the online case where $g_k = \nabla f_k(w_k)$.

We implement Adagrad with element-wise operations (easy!)

Diagonal Adagrad

Initialize $v_0 = 0$. Then:

$$v_k = v_{k-1} + g_k^2$$
$$w_{k+1} = w_k - \alpha_k \frac{g_k}{\sqrt{v_k}}$$

This update rule is equivalent to using a diagonal scaling:

$$\begin{bmatrix} w_{k+1,1} \\ w_{k+1,2} \\ \vdots \\ w_{k+1,n} \end{bmatrix} = \begin{bmatrix} w_{k,1} \\ w_{k,2} \\ \vdots \\ w_{k,n} \end{bmatrix} - \alpha_k \begin{bmatrix} v_{k,1} & 0 & \ddots & 0 \\ 0 & v_{k,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v_{k,n} \end{bmatrix}^{-1/2} \begin{bmatrix} g_{k,1} \\ g_{k,2} \\ \vdots \\ g_{k,n} \end{bmatrix}$$

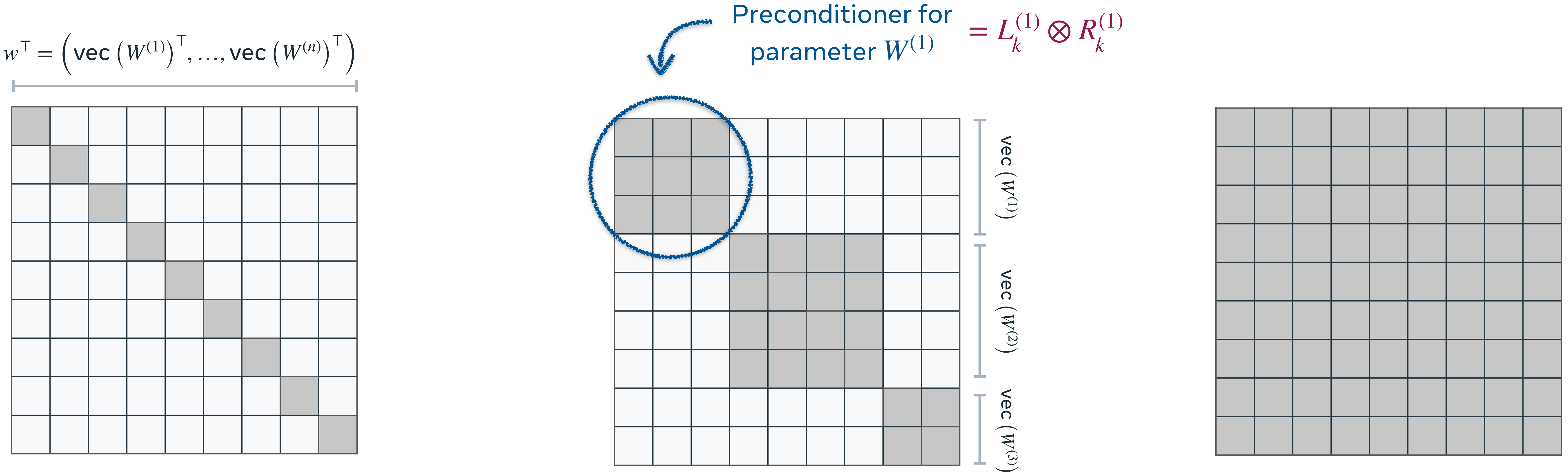
Does not capture gradient correlations

More generally, we can understand Adagrad as a PG method with $P_k = A_k^{-1/2}$, where

$$A_k = \begin{cases} \sum_{t=0}^{k} \mathrm{diag}(g_t^2) & \text{if diagonal Adagrad} \\ \sum_{t=0}^{k} g_t g_t^\top & \text{if full-matrix Adagrad (FMA)} \end{cases}$$

Duchi et al. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. JMLR'11

Shampoo Algorithm



Diagonal Adagrad — Shampoo — Full-matrix Adagrad (FMA)

Shampoo leverages two key approximations:

- Block-diagonal approximation to FMA; allows capturing weight correlations while reducing cost
- Kronecker product approximation to block-level preconditioner; exploits tensor structure in NN parameters

Gupta et al. Shampoo: Preconditioned Stochastic Tensor Optimization. ICML'18

Memory/computation spectrum of Adagrad-like methods

Consider applying an adaptive gradient method to a parameter matrix $W \in \mathbb{R}^{m \times n}$.

Identity

$A_k = \mathbb{I}$

No memory
No computation

Row-wise Adagrad, Adafactor, SM3row

$O(m+n)$ memory
$O(mn)$ computation

Diagonal Adagrad

$$A_k = \sum_{t \leq k} \text{diag}(g_t^2)$$

$O(mn)$ memory
$O(mn)$ computation

Shampoo

$O(m^2+n^2)$ memory
$O(m^3+n^3)$ computation

Full-matrix Adagrad

$$A_k = \sum_{t \leq k} g_t g_t^\top$$

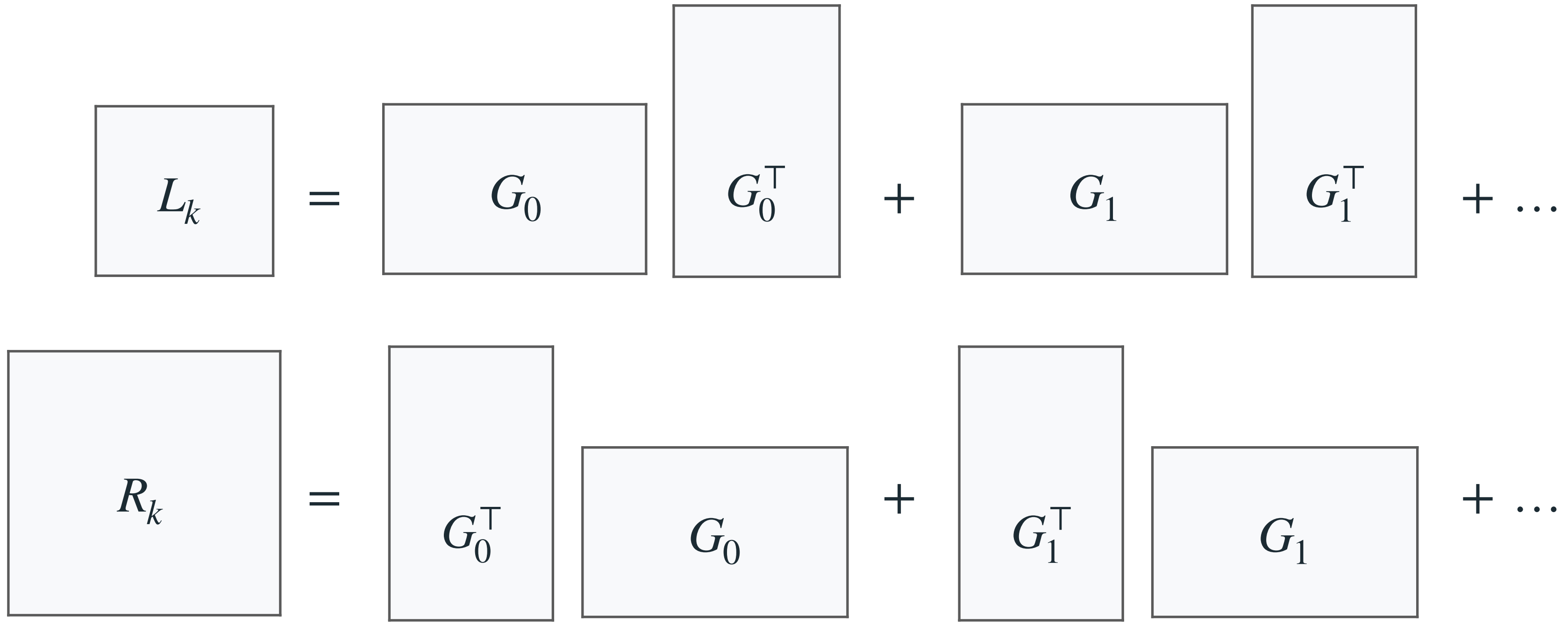
$O(m^2n^2)$ memory
$O(m^3n^3)$ computation

Shampoo (for matrices)

Shampoo can be applied to tensors of arbitrary order.

For simplicity, let us focus on a single fully-connected layer (without bias) with parameter matrix $W \in \mathbb{R}^{m \times n}$ and gradient $G \in \mathbb{R}^{m \times n}$.

Note that L_k and R_k are positive semi-definite square matrices.

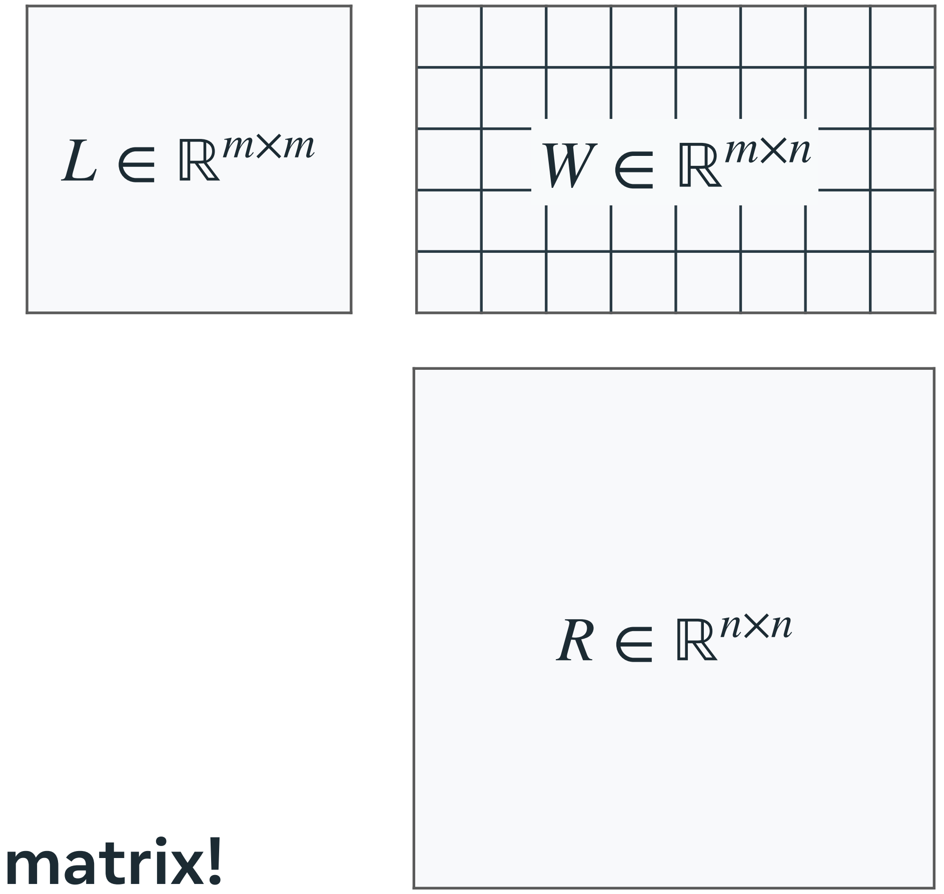


We need to invert and store $m \times m$ and $n \times n$ matrices, but never an $mn \times mn$ matrix!

Shampoo (Matrix)

Initialize $L_0 = 0 \in \mathbb{R}^{m \times m}$ and $R_0 = 0 \in \mathbb{R}^{n \times n}$.

$$L_k = L_{k-1} + G_k G_k^\top$$
$$R_k = R_{k-1} + G_k^\top G_k$$
$$W_{k+1} = W_k - \alpha_k L_k^{-1/4} G_k R_k^{-1/4}$$



Gupta et al. Shampoo: Preconditioned Stochastic Tensor Optimization. ICML'18

Two points of view

The *mixed Kronecker matrix-vector product* property yields:

Implementation POV

$$W_{k+1} = W_k - \alpha_k L_k^{-1/4} G_k R_k^{-1/4}$$

Theory POV

$$\mathrm{vec}(W_{k+1}) = \mathrm{vec}(W_k) - \alpha_k \left(L_k^{1/2} \otimes R_k^{\top/2}\right)^{-1/2} \mathrm{vec}(G_k)$$

Thus, we can interpret the Shampoo update as a Kronecker-factored block-diagonal preconditioner.

$$\underbrace{\begin{bmatrix} W_{k+1}^{(1)} \\ W_{k+1}^{(2)} \\ \vdots \\ W_{k+1}^{(l)} \end{bmatrix}}_{w_{k+1}} = \underbrace{\begin{bmatrix} W_k^{(1)} \\ W_k^{(2)} \\ \vdots \\ W_k^{(l)} \end{bmatrix}}_{w_k} - \alpha_k \underbrace{\begin{bmatrix} (L_k^{(1)})^{1/2} \otimes (R_k^{(1)})^{\top/2} & 0 & \cdots & 0 \\ 0 & (L_k^{(2)})^{1/2} \otimes (R_k^{(2)})^{\top/2} & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & (L_k^{(l)})^{1/2} \otimes (R_k^{(l)})^{\top/2} \end{bmatrix}^{-1/2}}_{A_k^{-1/2}} \underbrace{\begin{bmatrix} G_k^{(1)} \\ G_k^{(2)} \\ \vdots \\ G_k^{(l)} \end{bmatrix}}_{g_k}$$

Layer-wise LR grafting

- **Problem:** While Shampoo offers a good preconditioner for each layer, how do we "scale" or "equalize" the different blocks?
- **(Heuristic) Answer:** Layer-wise learning rate grafting
- **Key idea:** Use per-layer update size from base (aka *grafted*) optimizer
- **This is a key ingredient to make Shampoo work in practice.**

LR Grafting

$$
\begin{aligned}
u_{k,\text{Shampoo}} &= \text{ApplyPreconditioner}(L_k, R_k, g_k) \\
u_{k,\text{Grafted}} &= \text{GraftingUpdate}(g_k) \\
W_{k+1} &= W_k - \alpha_k \|u_{k,\text{Grafted}}\|_F \frac{u_{k,\text{Shampoo}}}{\|u_{k,\text{Shampoo}}\|_F}
\end{aligned}
$$

Agarwal et al. Disentangling Adaptive Gradient Methods from Learning Rates. arXiv'20

Performance optimizations

Multi-GPU training allows to accelerate training over large datasets

(Standard) Distributed Data-Parallel Training

- Model parameters are replicated across workers
- Each worker only computes a local subset of gradients
- Global mini-batch gradient is aggregated across workers
- Optimizer update is then carried out at each worker

Shampoo update is more complex — matrix ops. Naive replication would be sub-optimal!

1. **Distributed preconditioner storage and computation**
2. **Handling tensors of large dimensions**
3. **Periodic root-inverse computation**

Image classification experiments

ResNet50 model

- ~25M parameters
- Convolutional architecture
- Residual connections
- Batch-normalization layers

ImageNet dataset

- 1M+ labelled examples
- 1k classes

See full details in paper!

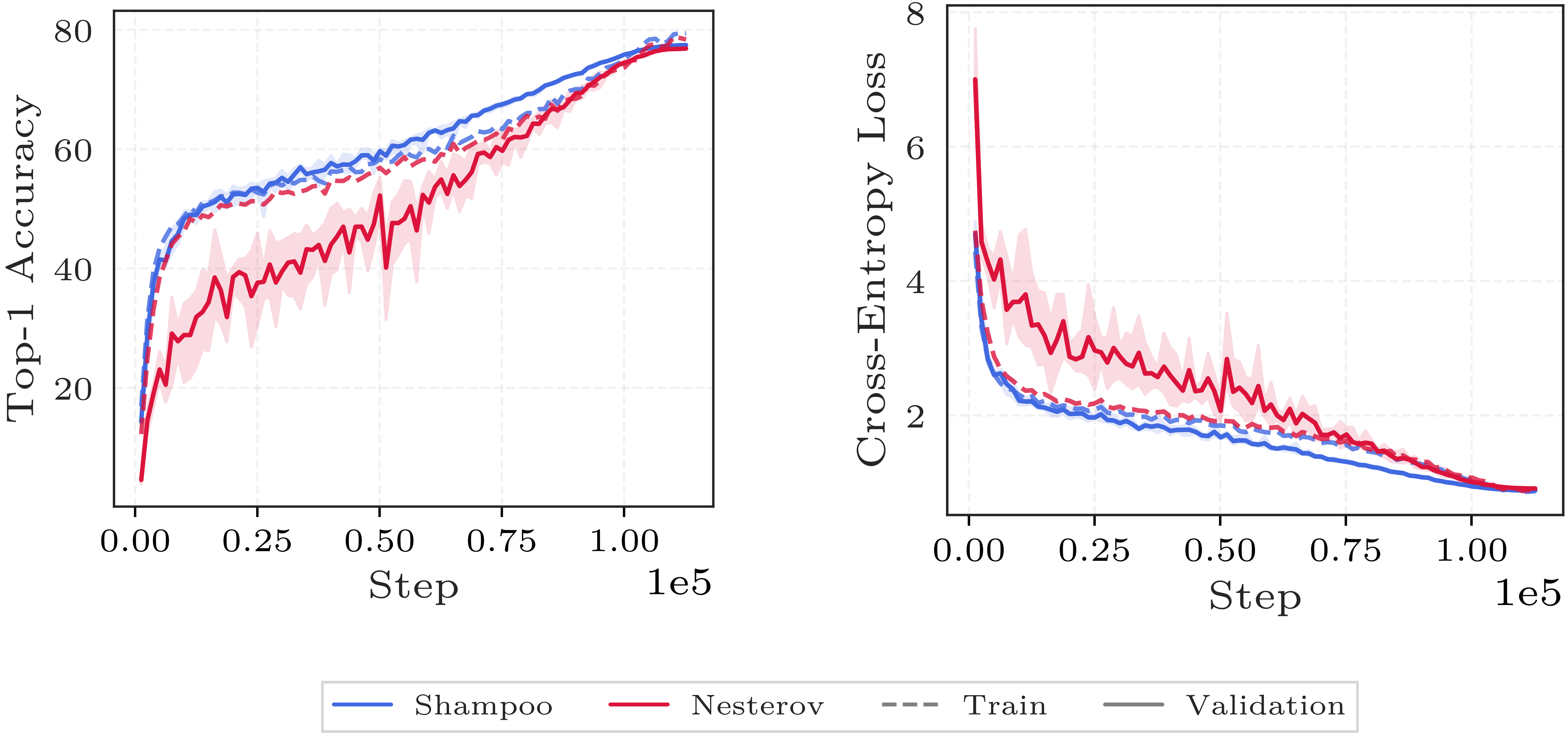
Experimental ablations

- Max. preconditioner dimension
- Preconditioner update frequency
- Restricted number of epochs
- Sensitivity to learning rate

Deng et al. ImageNet: A Large-Scale Hierarchical Image Database. CVPR'09

Image classification experiments

Both methods train for 90 epochs



Meta AI

Image classification experiments

Both methods train for 90 epochs

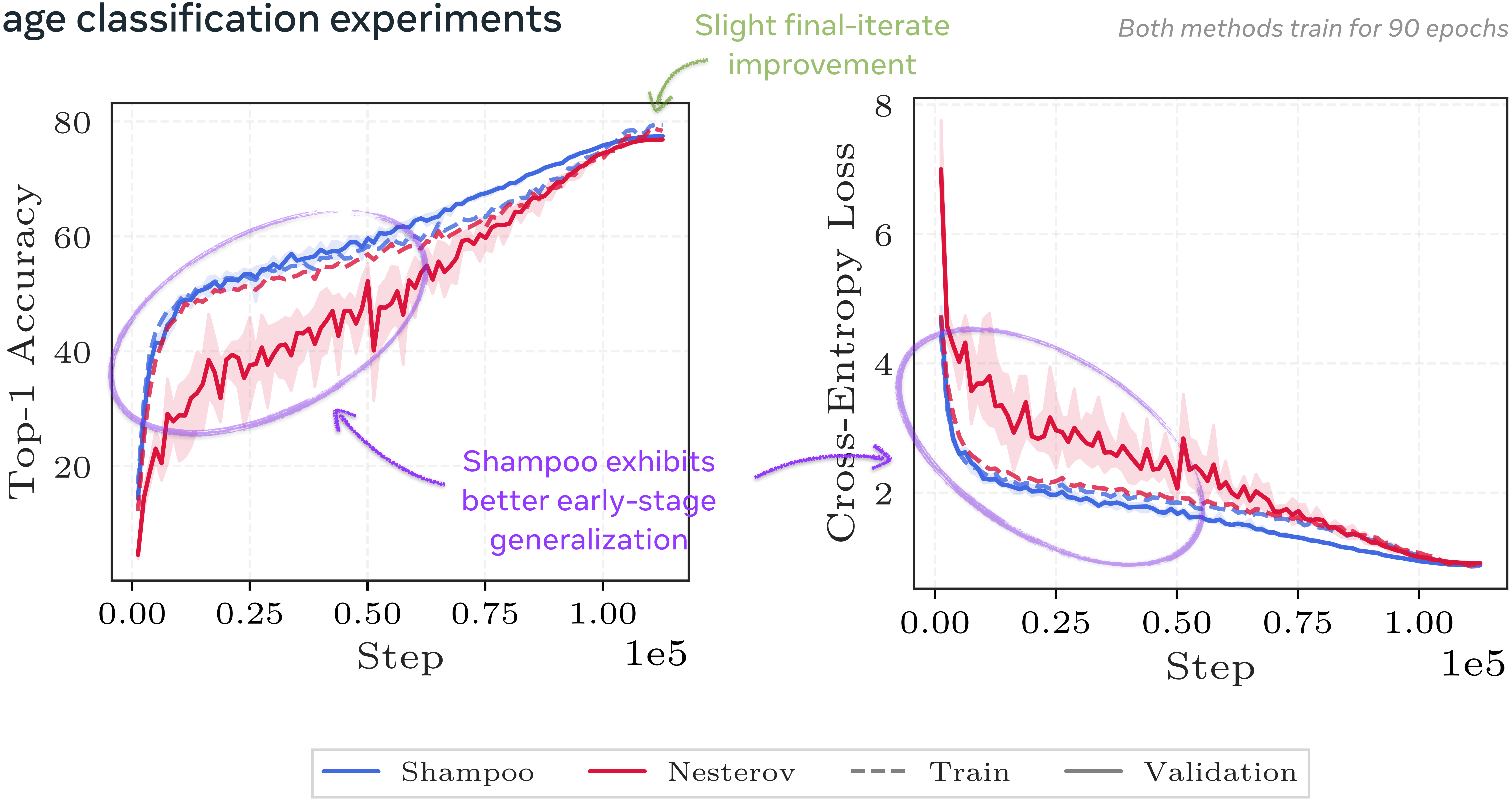


Image classification experiments

Shampoo overhead is competitive at ~10%

Both methods train for 90 epochs

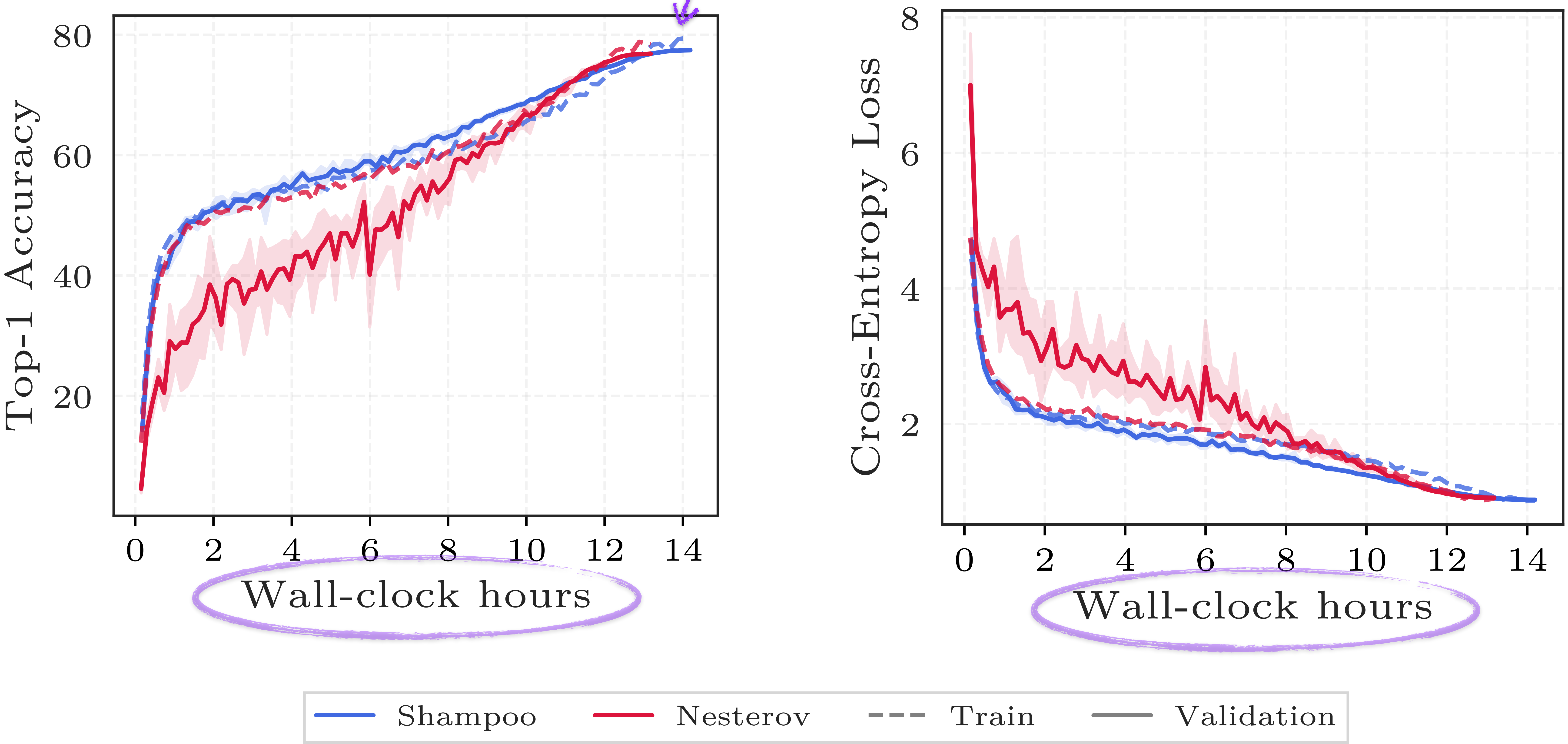
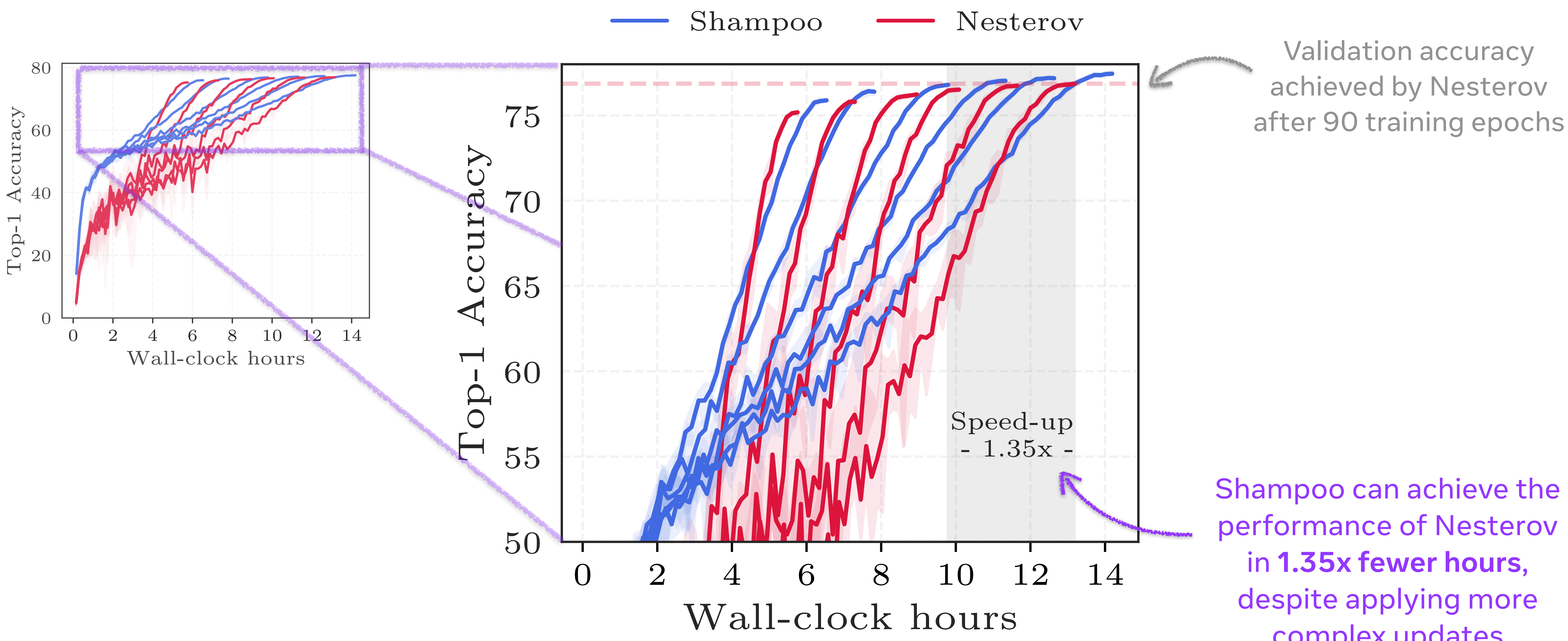


Image classification experiments



Shampoo can achieve the performance of Nesterov in **1.35x fewer hours,** despite applying more complex updates

Conclusion

1. **Well-engineered open-source implementation of Distributed Shampoo**
   https://github.com/facebookresearch/optimizers/tree/main/distributed_shampoo

2. **Our experiments corroborate improvements over popular baselines**

3. **Open questions on making heuristics like grafting rigorous**

4. **Full implementation and usage details available on preprint**
   https://arxiv.org/abs/2309.06497

Meta AI