Scalable Deep Neural Network Training with Distributed K-FAC

Greg Pauloski
23 March 2022

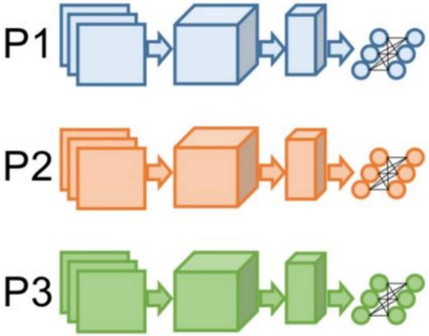
Committee: Kyle Chard, Ian Foster, Zhao Zhang

Outline

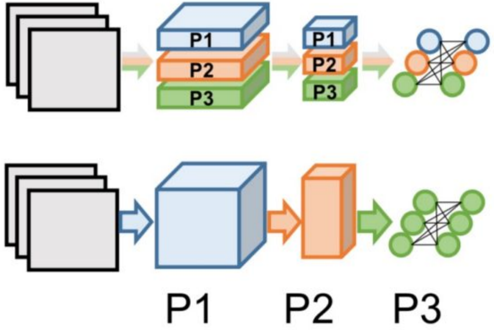
1. Challenges in large scale deep learning training
2. Optimization methods: first-order vs. second-order
3. Prior work in distributed K-FAC
4. Communication optimized distributed K-FAC [*SC ‘20, TPDS ‘22*]
5. KAISA: generalizing distributed second-order optimization [*SC ‘21*]
6. Implementation
7. Evaluation

Large Scale Training

Faster ➡ *Data Parallelism*



Larger Models ➡ *Model Parallelism*



Can we keep scaling the batch size to use more processors?

Published as a conference paper at ICLR 2017

ON LARGE-BATCH TRAINING FOR DEEP LEARNING: GENERALIZATION GAP AND SHARP MINIMA

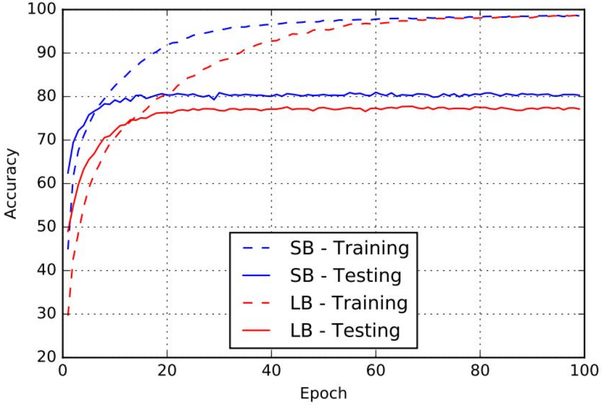
Nitish Shirish Keskar*
Northwestern University
Evanston, IL 60208
keskar.nitish@u.northwestern.edu

Dheevatsa Mudigere
Intel Corporation
Bangalore, India
dheevatsa.mudigere@intel.com

Jorge Nocedal
Northwestern University
Evanston, IL 60208
j-nocedal@northwestern.edu

Mikhail Smelyanskiy
Intel Corporation
Santa Clara, CA 95054
mikhail.smelyanskiy@intel.com

Ping Tak Peter Tang
Intel Corporation
Santa Clara, CA 95054
peter.tang@intel.com



HPC and Machine Learning

Large batch sizes (e.g., 100K to 1M)...

- ++ Used to **scale out** to more nodes.
- -- Leads to **worse generalization** performance and **higher communication** costs.

How can we better enable large batch training on HPC (generalization performance and scaling)?

Previous Efforts

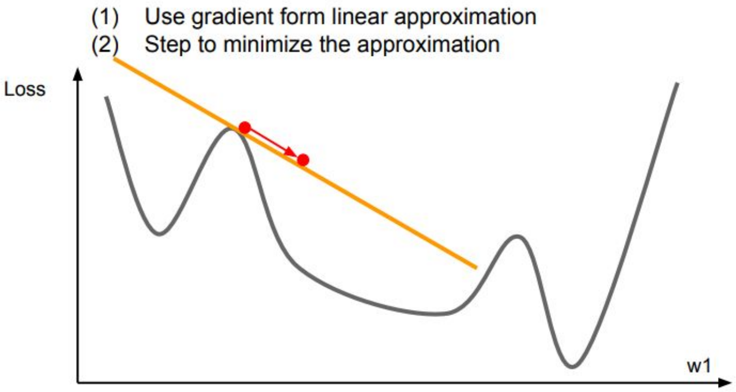
Generalization Gap

- Learning Rate Warmup
- Learning Rate Scaling
- Batch Size Warmup
- Layer-wise Adaptive Learning Rate (LARS/LAMB)
- Distributed Batch Normalization
- . . . and many more

Better Scaling

- Gradient Compression
- Lower Precision (FP16)
- Network Topology Aware Operations
- Layer Pipelining/Hybrid-Parallelism
- Asynchronous SGD
- Custom Fused Kernels
- Gradient Checkpointing
- . . . and many more

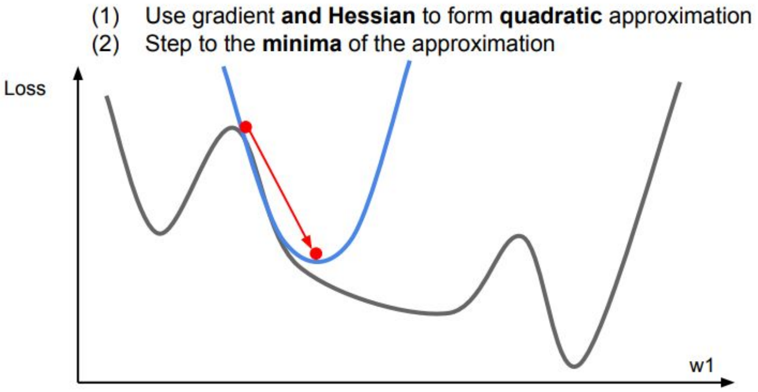
First-Order Optimization



1. Use gradient form linear approximation
2. Step to minimize the approximation

$$\text{SGD: } w_{k+1} = w_k - \alpha \nabla L(w_k)$$

Second-Order Optimization



1. Use gradient **and Hessian** to form **quadratic** approximation
2. Step to the **minima** of the approximation

$$\text{Newton: } w_{k+1} = w_k - \alpha \underbrace{H^{-1} \nabla L(w_k)}_{\text{Precondition}}$$

Hessian has $O(n^2)$ elements
Inversion is $O(n^3)$ operations

http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture07.pdf

Second-Order Optimization

A good candidate for large batch, distributed training!

1. Larger batches are more representative of the dataset's distribution.

   → *infrequent second-order information updates*

2. Gradient noise limits batch size and increases throughout training (McCandlish, 2018).

   → *second-order methods optimize noise-independent terms better* (Martens, 2014)

3. Higher computation-to-communication ratio in second-order methods.

   → *enables more advanced distribution schemes*

Kronecker-Factored Approximate Curvature

- Second-order methods incorporate the curvature of the parameter space.
  - ++ More progress optimizing the objective function per-iteration
  - -- **Expensive to compute!**

- K-FAC **efficiently approximates** the Fisher Information Matrix (FIM) for preconditioning the gradients (Martens+, 2015).

$$\text{SGD: } w^{(k+1)} = w^{(k)} - \frac{\alpha^{(k)}}{n} \sum_{i=1}^{n} \nabla L_i(w^{(k)}) \qquad \text{K-FAC: } w^{(k+1)} = w^{(k)} - \frac{\alpha^{(k)} \boxed{F^{-1}(w^{(k)})}}{n} \sum_{i=1}^{n} \nabla L_i(w^{(k)})$$

- Generalizes better with **large batch sizes** and **converges in fewer iterations** than first-order methods (Ba+, 2017)
  - Scales to extremely large batch sizes, e.g., 131k for ImageNet training (Osawa+, 2019)

Kronecker Product

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix}$$

$$m \times n \otimes p \times q \longrightarrow mp \times nq$$

$$\begin{bmatrix} 1 & \boxed{2} \\ 3 & 4 \end{bmatrix} \otimes \boxed{\begin{bmatrix} 5 & 6 \\ 7 & 8 \\ 9 & 0 \end{bmatrix}} = \begin{bmatrix} 1 \times 5 & 1 \times 6 & \boxed{2} \times 5 & \boxed{2} \times 6 \\ 1 \times 7 & 1 \times 8 & \boxed{2} \times 7 & \boxed{2} \times 8 \\ 1 \times 9 & 1 \times 0 & \boxed{2} \times 9 & \boxed{2} \times 0 \\ 3 \times 5 & 3 \times 6 & 4 \times 5 & 4 \times 6 \\ 3 \times 7 & 3 \times 8 & 4 \times 7 & 4 \times 8 \\ 3 \times 9 & 3 \times 0 & 4 \times 9 & 4 \times 0 \end{bmatrix}$$

$$2 \times 2 \otimes 3 \times 2 \longrightarrow 6 \times 4$$

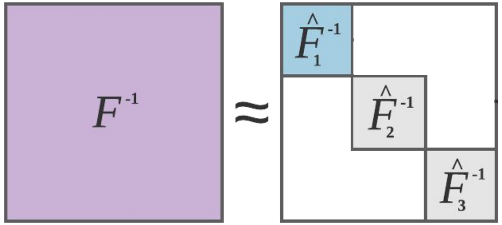
Efficient F Approximation

Step 1: Approximate the FIM as a block diagonal matrix*

$$\hat{F} = \mathtt{diag}(\hat{F}_1, \ldots, \hat{F}_i, \ldots, \hat{F}_L)$$

Step 2: Decompose each block as the Kronecker Product of the activations of the previous layer with the gradient w.r.t. the output of the current layer

$$\hat{F}_i = a_{i-1}a_{i-1}^\top \otimes g_i g_i^\top = A_{i-1} \otimes G_i$$



**Recall inverse of block diagonal matrix is composed of the inverses of each block*

Efficient Gradient Preconditioning

Step 3: Apply properties of Kronecker Product to weight update equation

Properties: $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$ $\quad (A \otimes B)\vec{c} = B^{\top}\vec{c}A$

Weight Update: $w_i^{(k+1)} = w_i^{(k)} - \alpha^{(k)} \boxed{\hat{F}_i^{-1} \nabla L_i(w_i^{(k)})}$

$$\begin{aligned}
\hat{F}_i^{-1} \nabla L_i(w_i^{(k)}) &= (A_{i-1} \otimes G_i)^{-1} \nabla L_i(w_i^{(k)}) \\
&= (A_{i-1}^{-1} \otimes G_i^{-1}) \nabla L_i(w_i^{(k)}) \\
&= G_i^{-1} \nabla L_i(w_i^{(k)}) A_{i-1}^{-1}
\end{aligned}$$

Preconditioned Gradient

Computing Inverses

. . . can be difficult

Tikhonov Regularization: $(\hat{F}_i + \gamma I)^{-1} = (A_{i-1} + \gamma I)^{-1} \otimes (G_i + \gamma I)^{-1}$

Use Eigen Decompositions:

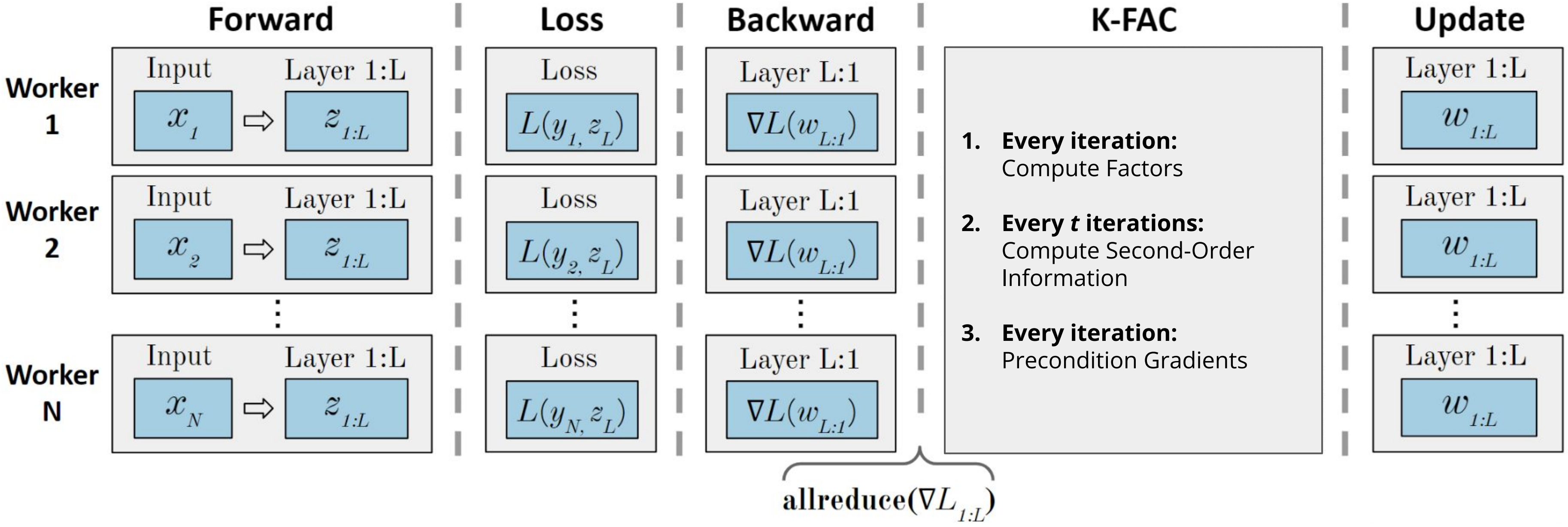
$$V_1 = Q_G^\top \nabla L(w_i^{(k)}) Q_A$$

$$V_2 = V_1 / (v_G v_A^\top + \gamma)$$

$$(\hat{F}_i + \gamma I)^{-1} \nabla L(w_i^{(k)}) = Q_G V_2 Q_A^\top$$

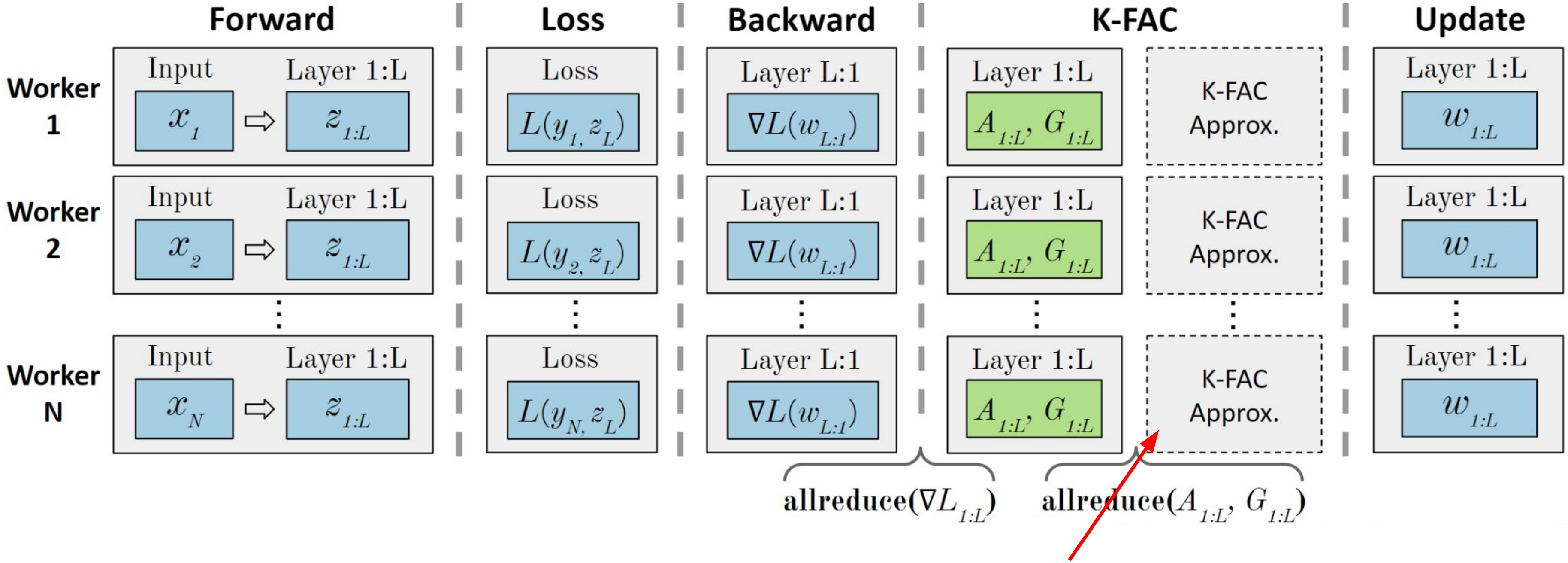
Batch Size	256	512	1024
SGD	92.77%	92.58%	92.69%
K-FAC with Inverse	92.58%	92.36%	91.71%
K-FAC with Eigen-decomp.	92.76%	92.90%	92.92%

Data Parallel Training with K-FAC



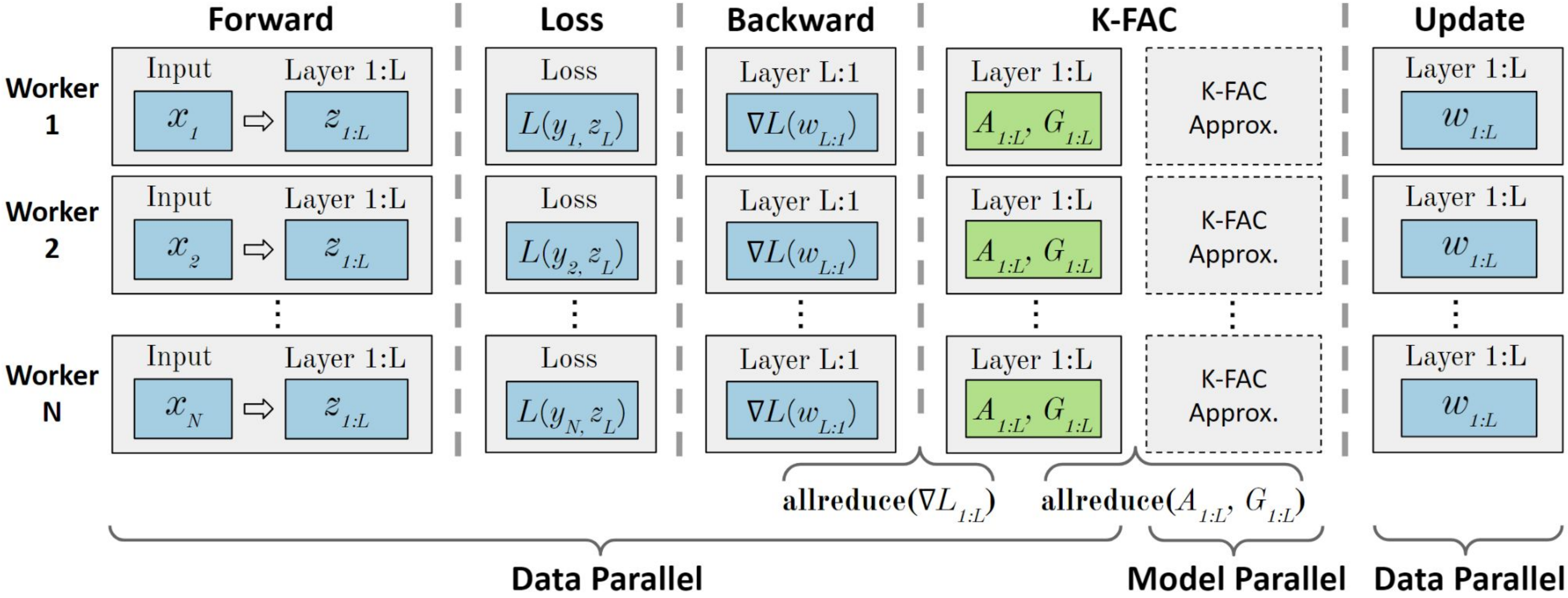
1. **Every iteration:** Compute Factors
2. **Every *t* iterations:** Compute Second-Order Information
3. **Every iteration:** Precondition Gradients

Data Parallel Training with K-FAC



Redundant expensive second-order computations

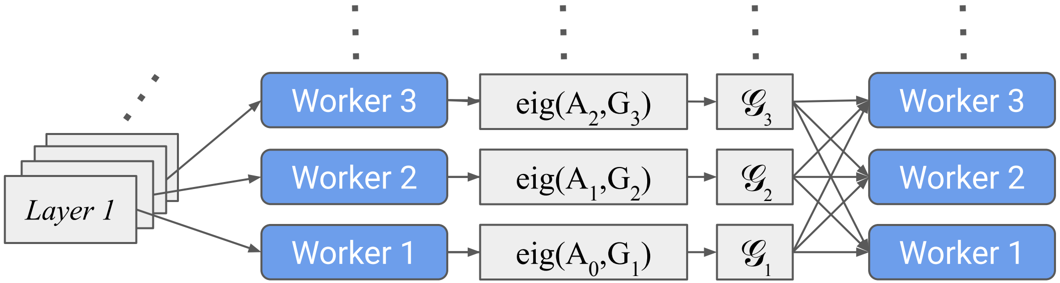
Data Parallel Training with K-FAC



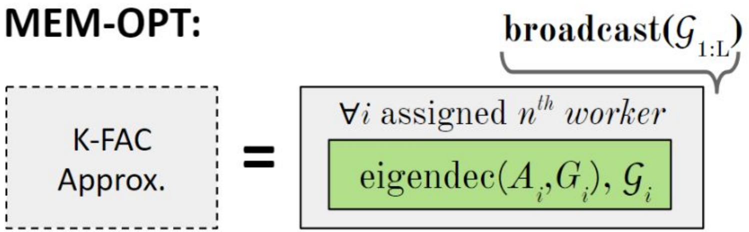
Model Parallel K-FAC Stage

MEM-OPT

Osawa et al. (CVPR 2019)



- ++ Lower memory usage
- -- Communication required every iteration



MEM-OPT:

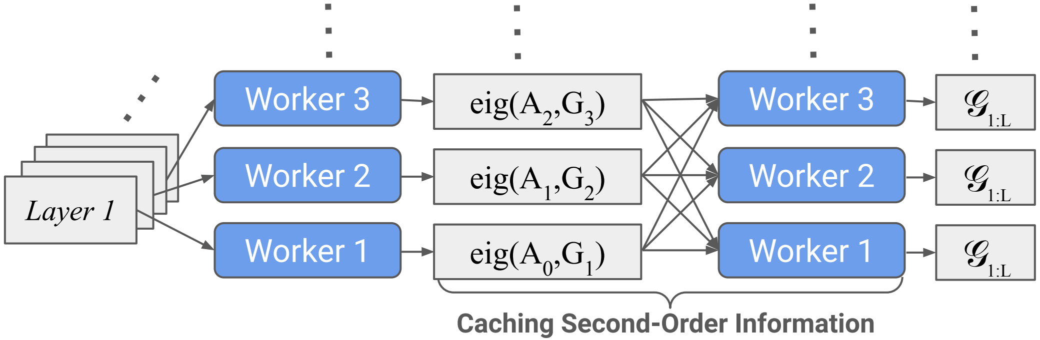
$$\boxed{\text{K-FAC Approx.}} = \overbrace{\forall i \text{ assigned } n^{th} \; worker \;\; \mathrm{eigendec}(A_i,G_i),\ \mathcal{G}_i}^{\mathbf{broadcast}(\mathcal{G}_{1:L})}$$

Can we make the communication frequency a function of the second-order computation frequency?

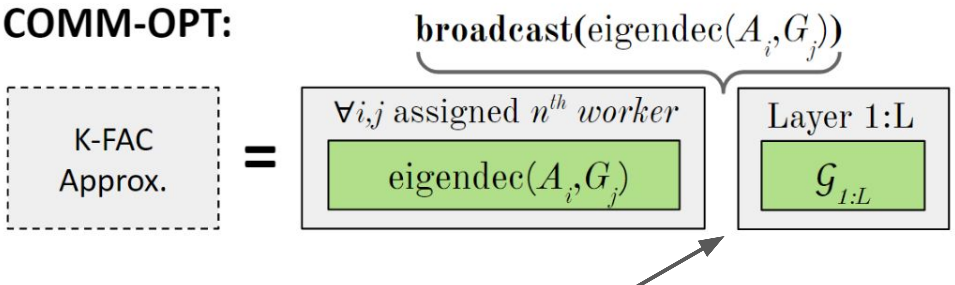
Model Parallel K-FAC Stage

COMM-OPT

Pauloski et al. (SC20)



- ++ Communication every *t* iterations
- -- Higher memory usage due to caching



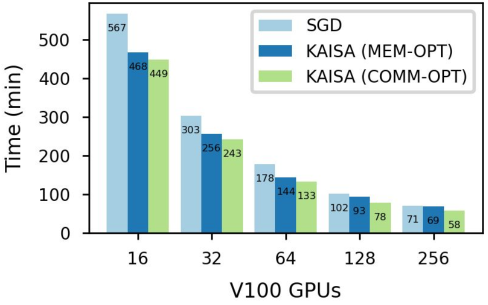
COMM-OPT:

$$\boxed{\text{K-FAC Approx.}} = \overbrace{\boxed{\forall i,j \text{ assigned } n^{th}\ \mathit{worker}\quad \boxed{\mathrm{eigendec}(A_i,G_j)}}}^{\mathbf{broadcast}(\mathrm{eigendec}(A_i,G_j))}\ \boxed{\text{Layer 1:L}\quad \boxed{\mathcal{G}_{1:L}}}$$

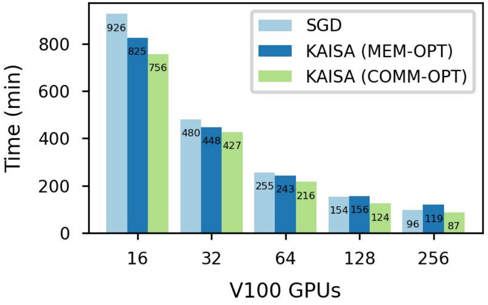
Preconditioning is cheap compared to eigen decomposition

MEM-OPT vs COMM-OPT

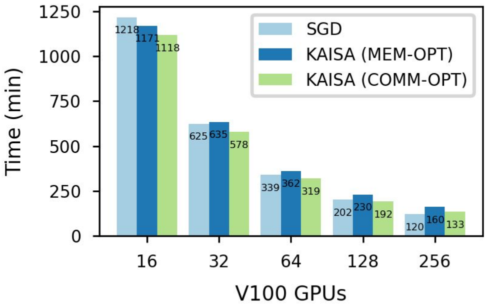
Communication Operation Interval				
	Gradients	**Factors**	**Preconditioned Gradients**	**Inverses/Eigen Decomps**
	`Allreduce`	`Allreduce`	`Broadcast/Allgather`	`Broadcast/Allgather`
MEM-OPT	**1**	***t***	**1**	**N/A**
COMM-OPT	**1**	***t***	**N/A**	***t***



(a) ***ResNet-50***

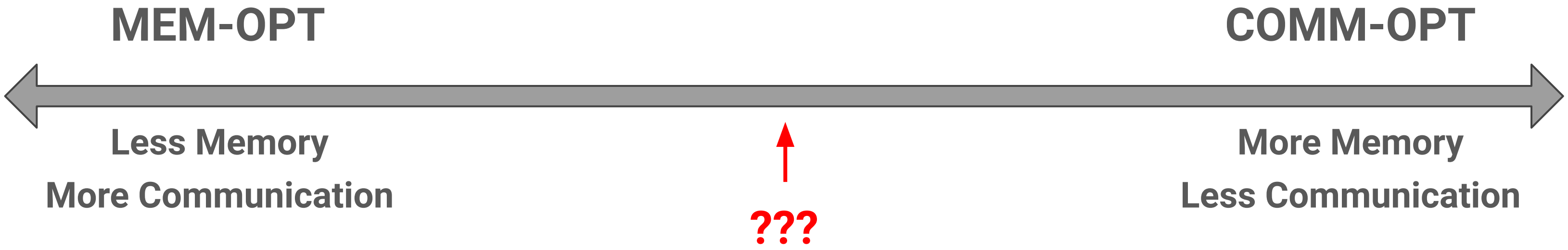


(a) ***ResNet-101***



(a) ***ResNet-152***

Communication vs Memory



MEM-OPT

COMM-OPT

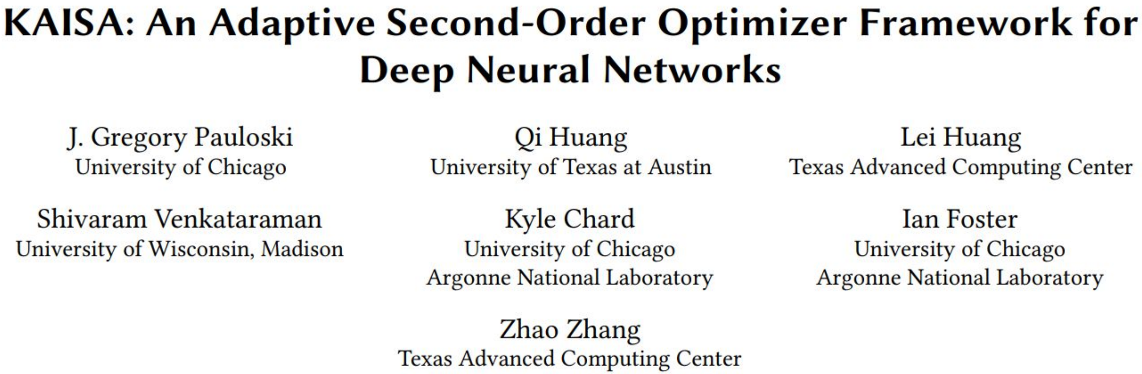
Less Memory

More Communication

???

More Memory

Less Communication



KAISA: An Adaptive Second-Order Optimizer Framework for Deep Neural Networks

J. Gregory Pauloski
University of Chicago

Qi Huang
University of Texas at Austin

Lei Huang
Texas Advanced Computing Center

Shivaram Venkataraman
University of Wisconsin, Madison

Kyle Chard
University of Chicago
Argonne National Laboratory

Ian Foster
University of Chicago
Argonne National Laboratory

Zhao Zhang
Texas Advanced Computing Center

Distribution of Work

Worker Types for a Layer (*N* workers)

Second-Order Worker	Gradient Worker	Gradient Receivers
Compute second-order information (one of the gradient workers)	Compute preconditioned gradient with second-order information	Receives preconditioned gradients from a gradient worker
1 per layer	***m*** per layer	***N* - *m*** per layer

Gradient Worker Fraction (*grad_worker_frac*) = ***m / N***

MEM-OPT: *grad_worker_frac* = **1/N**
COMM-OPT: *grad_worker_frac* = **1**

Gradient Worker Fraction

MEM-OPT

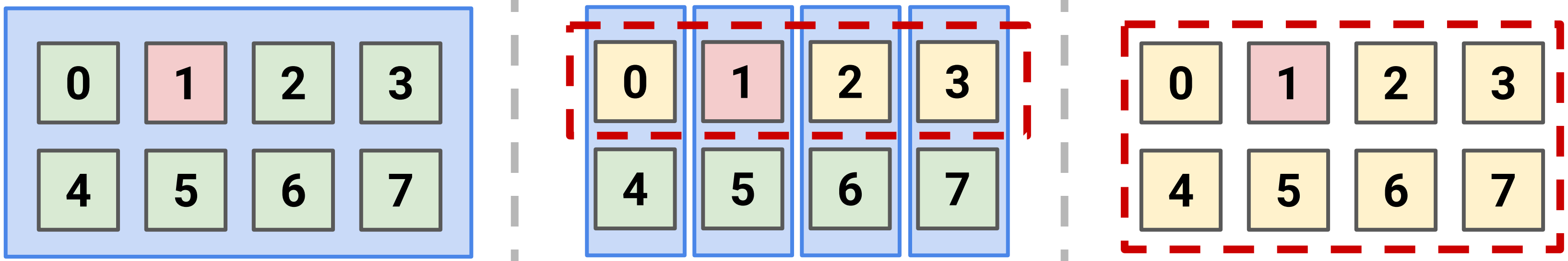
grad_worker_frac = 1/8

HYBRID-OPT

grad_worker_frac = 1/2

COMM-OPT

grad_worker_frac = 1



🟥 1 : second-order (gradient worker)

🟨 0 : gradient worker

🟩 0 : gradient receiver

🟦 : preconditioned gradient communication group (every iteration)

⬚ : second-order information communication group (every *t* iterations)

Gradient Worker Fraction

MEM-OPT

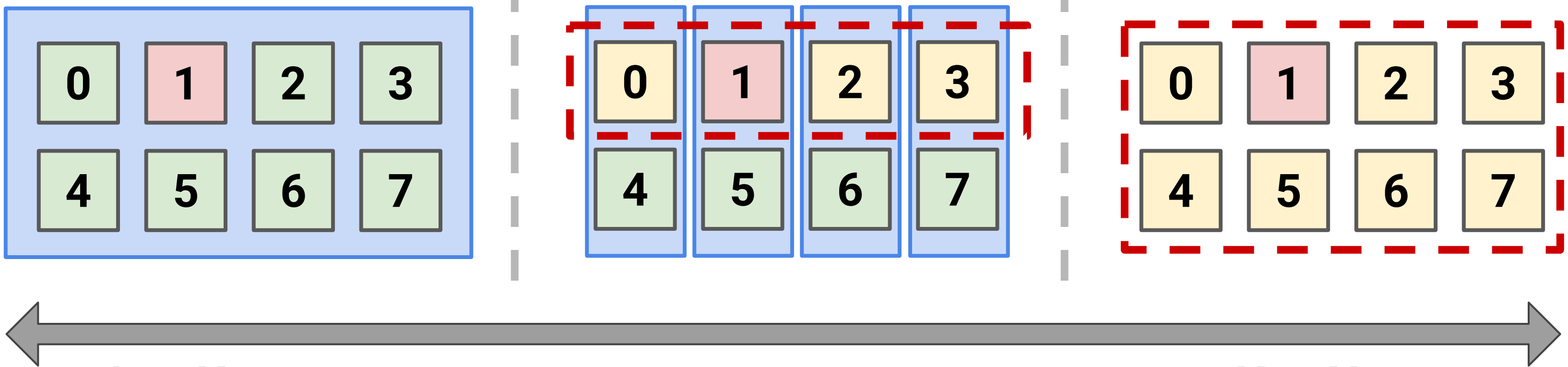
grad_worker_frac = 1/8

HYBRID-OPT

grad_worker_frac = 1/2

COMM-OPT

grad_worker_frac = 1

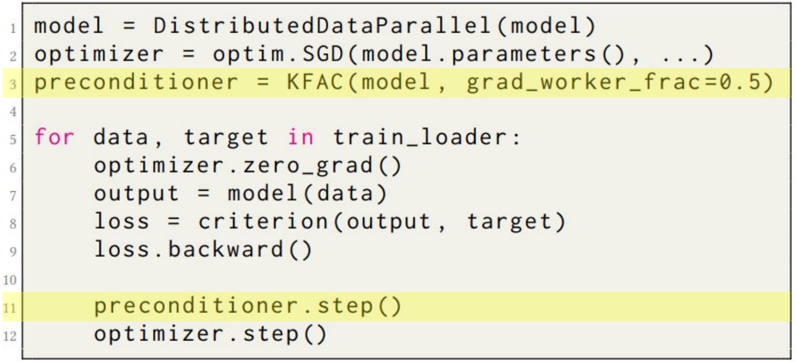


Less Memory
More Communication

More Memory
Less Communication

KAISA: Design Goals

- PyTorch **gradient preconditioner**
- **K-FAC** for second-order method
- **Adaptable** distribution scheme
- Understand the memory and communication **tradeoffs** in distributed second-order optimization
- Show KAISA is **faster** than default optimizers



```
model = DistributedDataParallel(model)
optimizer = optim.SGD(model.parameters(), ...)
preconditioner = KFAC(model, grad_worker_frac=0.5)

for data, target in train_loader:
    optimizer.zero_grad()
    output = model(data)
    loss = criterion(output, target)
    loss.backward()

    preconditioner.step()
    optimizer.step()
```

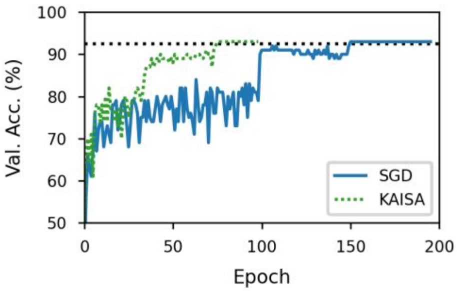
Listing 1: Example K-FAC usage.

https://github.com/gpauloski/kfac_pytorch

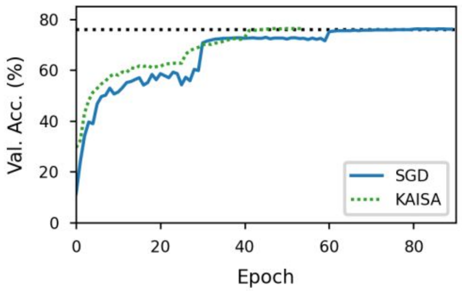
KAISA: Features

- Inverse or Eigen decomposition (default) K-FAC preconditioning
  - Linear and Conv2D layers
- Data Parallel Training Frameworks: PyTorch, DeepSpeed, NVIDIA Apex
- Adaptable distribution scheme (gradient worker fraction)
- Mixed Precision Training
- Gradient Accumulation
- Other Minor Optimizations:
  - Symmetry aware communication and communication bucketing
  - Preconditioning precomputation

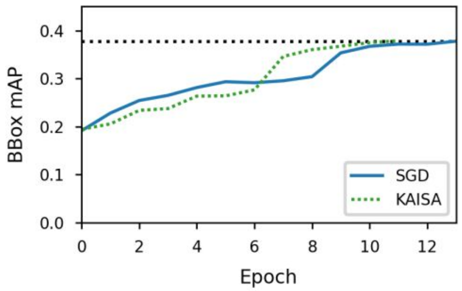
Evaluation: Convergence



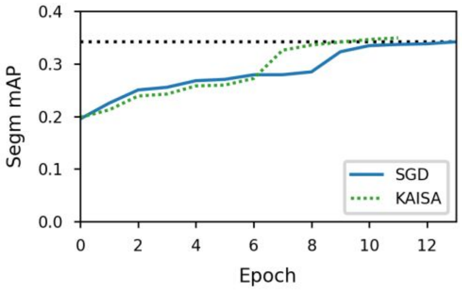
(a) ResNet-34 Accuracy



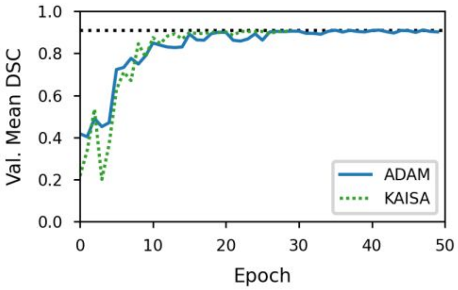
(b) ResNet-50 Accuracy



(c) Mask R-CNN BBox mAP



(d) Mask R-CNN Segm mAP



(e) U-Net Dice Score

Evaluation: Time-to-Convergence w/ Fixed Batch Size

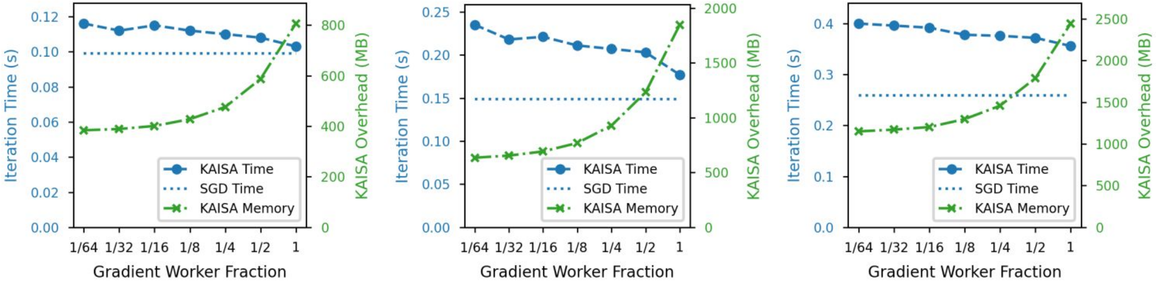
App	Default Optimizer	Baseline	# GPUs	Global Batch Size	Precision	KAISA Time-to-Convergence Improvement
ResNet-50	SGD	75.9% Val. Acc.	8 A100	2048	FP16	**24.3%**
Mask R-CNN	SGD	0.377 bbox mAP 0.342 segm mAP	64 V100	64	FP32	**18.1%**
U-Net	ADAM	91.0% Val. DSC	4 A100	64	FP32	**25.4%**
BERT-Large (Phase 2)	LAMB	90.8% SQuAD v1.1 F1	8 A100	65,636	FP16	**36.3%**

Evaluation: Time-to-Convergence w/ Fixed Memory Budget

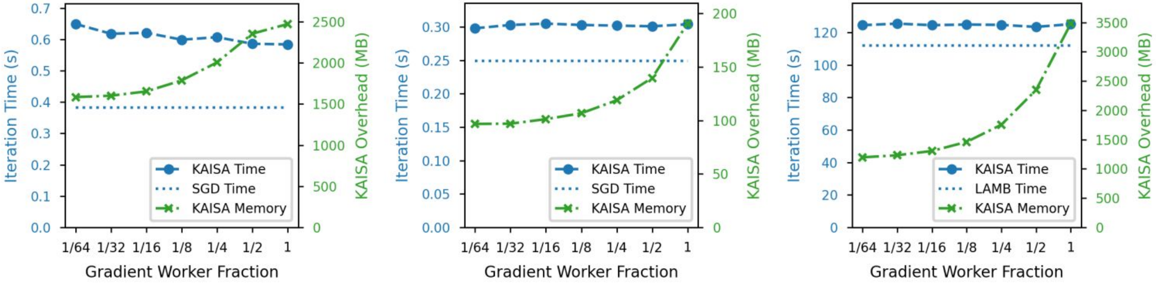
App	Optimizer	GPUs	Grad. Worker Frac.	Local Batch Size	Time-to-Convergence (minutes)
ResNet-50	SGD	64 V100	--	128	**123 (DNC)**
	KAISA		1/64	80	**96**
	KAISA		1/2	80	**83**
BERT-Large (Phase 2)	LAMB	8 A100	--	12	**2918**
	KAISA		1/2	8	**1703**
	KAISA		1	8	**1704**

* Use max possible local batch size for each experiment and measure time-to-convergence.

Evaluation: Memory vs. Communication



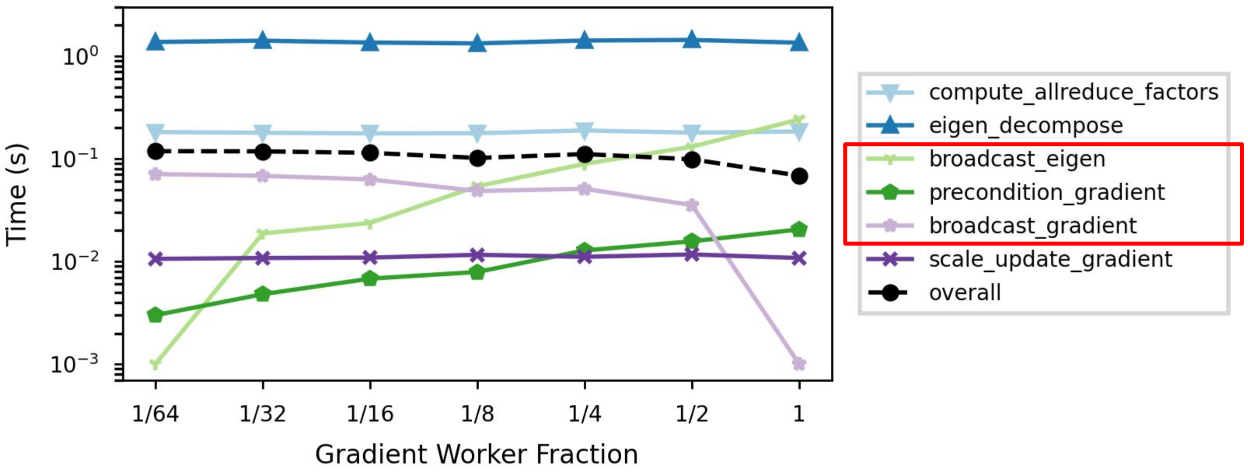
(a) ResNet-18 (FP32) (b) ResNet-50 (FP32) (c) ResNet-101 (FP32)



(d) ResNet-152 (FP32) (e) Mask R-CNN (FP32) (f) BERT-Large Phase 2 (FP16)

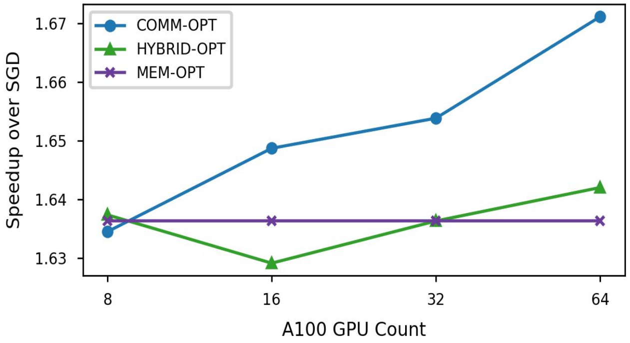
Figure 6.3: Average iteration time and K-FAC memory overhead across *grad-worker-frac* values on 64 V100 GPUs. Dotted lines are the baseline iteration times without K-FAC.

Evaluation: Memory vs. Communication

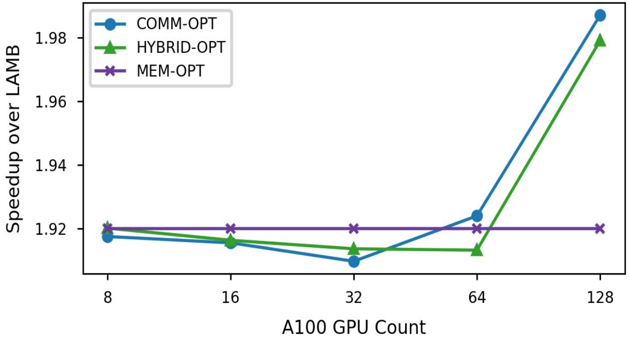


Evaluation: Scaling

- 27-29% **faster than SGD** with ResNet-50
- 41-44% **faster than LAMB** for BERT-Large
- MEM-OPT has constant speedup
- HYBRID/COMM-OPT **improve with scale**
- HYBRID-OPT **best balance** of memory usage and scaling with BERT-Large



(a) ResNet-50



(b) BERT-Large

Takeaways

Second-order optimization is **viable** for distributed training

- Converges to **same target** metrics as first-order methods
- Converges in **less wall time** with minimal configuration

Second-order optimization enables more **creative hybrid-parallel** schemes

KAISA's provides a framework for distributed training with **future second-order methods**

Questions?

`jgpauloski@uchicago.edu`

`github.com/gpauloski/kfac_pytorch`

J. Gregory Pauloski, Zhao Zhang, Lei Huang, Weijia Xu, and Ian T. Foster. 2020. *Convolutional Neural Network Training with Distributed K-FAC*. International Conference for High Performance Computing, Networking, Storage and Analysis (SC '20).

J. Gregory Pauloski, Qi Huang, Lei Huang, Shivaram Venkataraman, Kyle Chard, Ian Foster, and Zhao Zhang. 2021. *KAISA: An Adaptive Second-order Optimizer Framework for Deep Neural Networks*. International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21).

J. Gregory Pauloski, Lei Huang, Weijia Xu, Kyle Chard, Ian T. Foster, and Zhao Zhang. 2022. *Deep Neural Network Training with Distributed K-FAC*. To appear in Transactions on Parallel and Distributed Systems (TPDS).