Matrices in Deep Neural Networks and How to Compute Them in Parallel

Tokyo Institute of Technology
Rio Yokota

IEEE CLUSTER 2022
Heidelberg, Germany
2022/9/6-9

What we were doing back in 2017

Data-parallel training of ImageNet on thousands of GPUs

Jun 2017

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

facebook

Priya Goyal Piotr Dollár Ross Girshick Pieter Noordhuis
Lukasz Wesolowski Aapo Kyrola Andrew Tulloch Yangqing Jia Kaiming He

Facebook

Sep 2017

ImageNet Training in Minutes

Berkeley UNIVERSITY OF CALIFORNIA

Yang You[1], Zhao Zhang[2], Cho-Jui Hsieh[3], James Demmel[1], Kurt Keutzer[1]
UC Berkeley[1], TACC[2], UC Davis[3]
{youyang, demmel, keutzer}@cs.berkeley.edu; zzhang@tacc.utexas.edu; chohsieh@ucdavis.edu

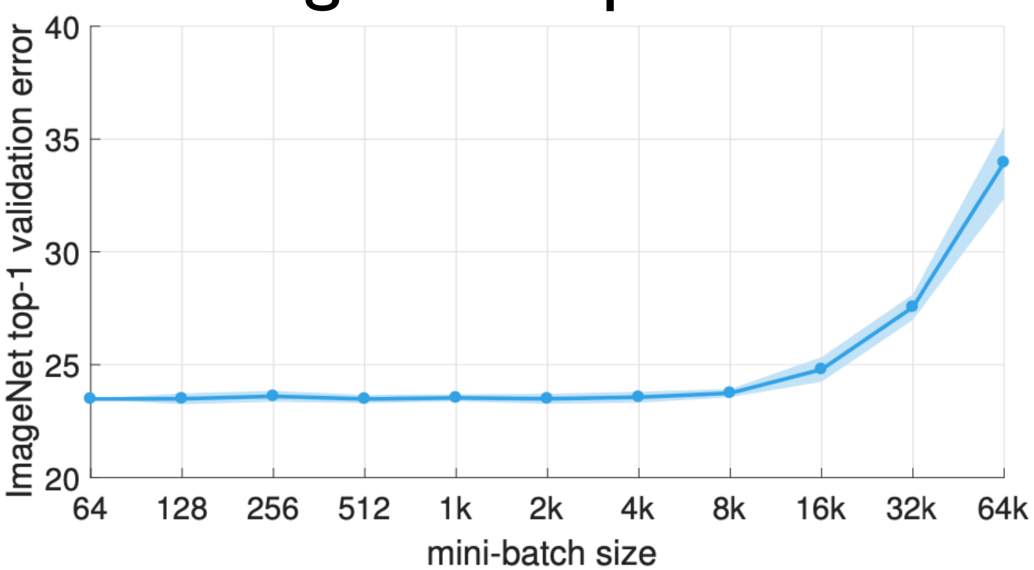
Nov 2017

Preferred Networks

Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes

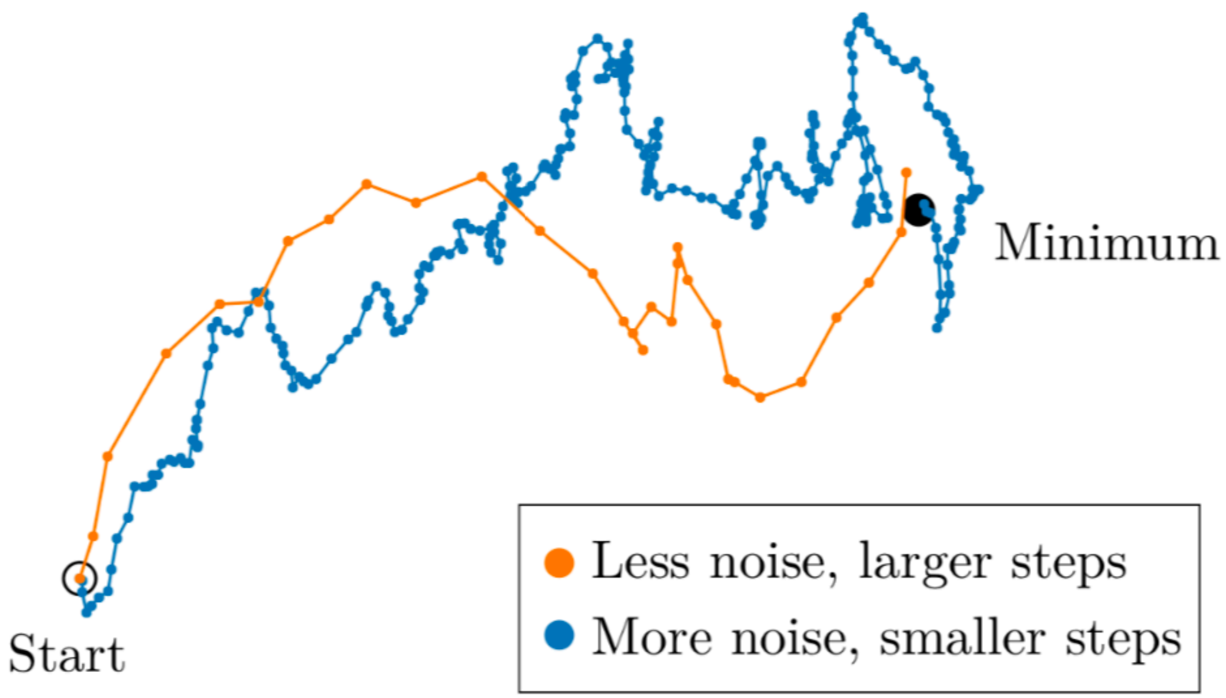
Takuya Akiba Preferred Networks, Inc. akiba@preferred.jp
Shuji Suzuki Preferred Networks, Inc. ssuzuki@preferred.jp
Keisuke Fukuda Preferred Networks, Inc. kfukuda@preferred.jp

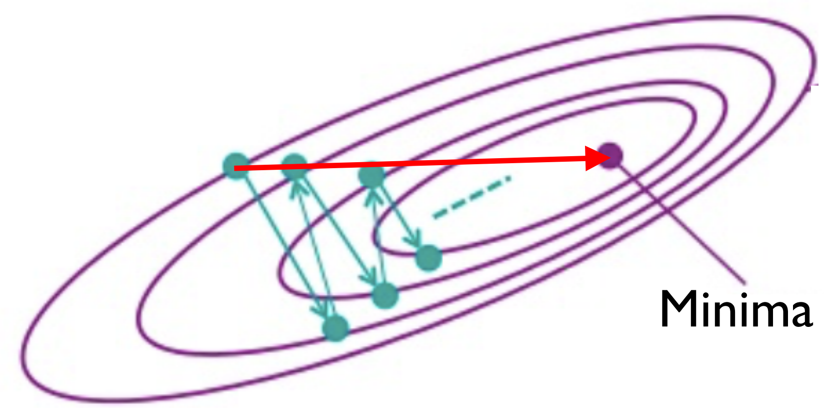
Large-batch problem



	Hardware	Software	Mini-batch size	Optimizer	Iteration	Time	Accuracy
Goyal *et al.* [9]	Tesla P100 × 256	Caffe2	8,192	SGD	14,076	1 hr	76.3%
You *et al.* [29]	KNL × 2048	Intel Caffe	32,768	SGD	3,519	20 min	75.4%
Akiba *et al.* [3]	Tesla P100 × 1024	Chainer	32,768	RMSprop/SGD	3,519	15 min	74.9%

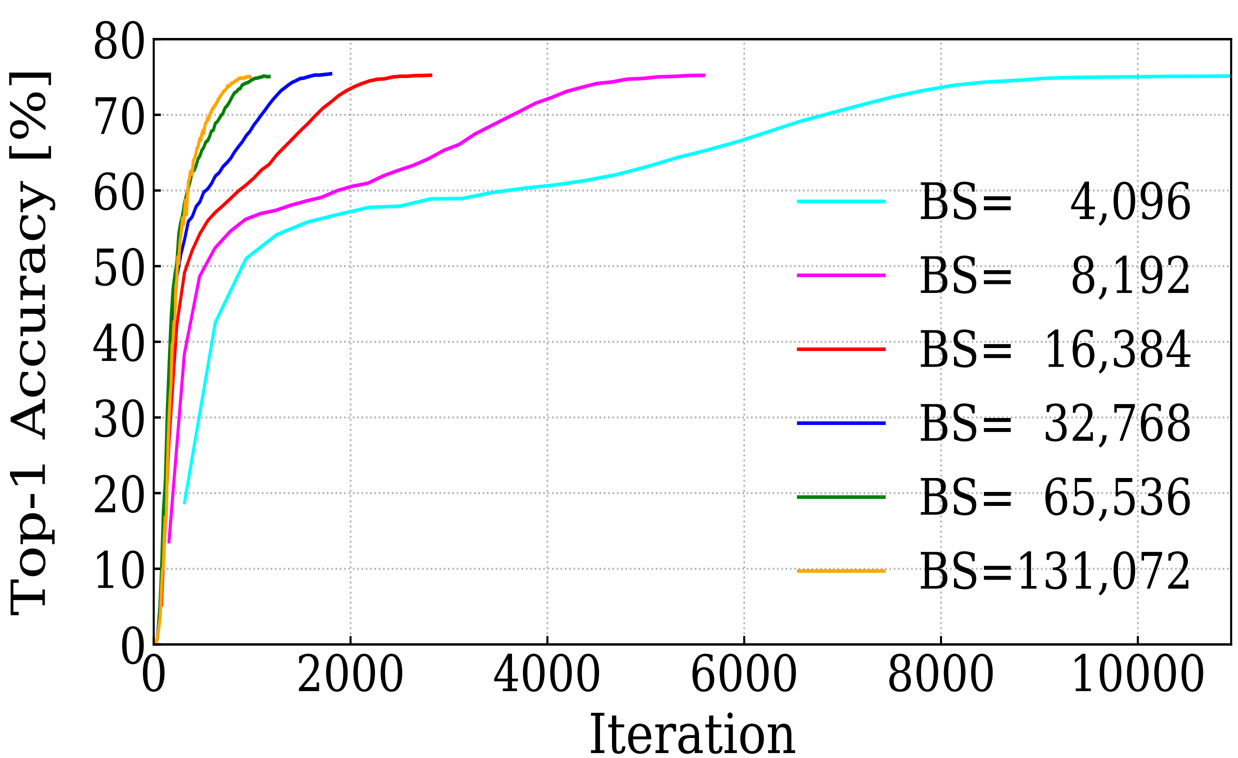
Our rationale at the time





Second order optimizers are good at taking a few large steps with less noise

Large batch training is about taking a few large steps with less noise



<u>Stochastic gradient descent (SGD)</u>

$$\theta_{t+1} = \theta_t - \eta \nabla \mathcal{L}(\theta_t)$$

<u>Natural gradient descent (NGD)</u>

$$\theta_{t+1} = \theta_t - \eta \underline{(F_{t+1} + \epsilon I)}^{-1} \nabla \mathcal{L}(\theta_t)$$

This is a dense matrix with P x P elements, where P is the number of parameters

Many big companies joined the race in 2018

Tencent

Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes

Xianyan Jia*[1], Shutao Song*[1], Wei He[1], Yangzihao Wang[1], Haidong Rong[1], Feihu Zhou[1], Liqiang Xie[1], Zhenyu Guo[1], Yuanzhou Yang[1], Liwei Yu[1], Tiegang Chen[1], Guangxiao Hu[1], Shaohuai Shi*[2], Xiaowen Chu[2]
Tencent Inc.[1], Hong Kong Baptist University[2]

Google

Image Classification at Supercomputer Scale

Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, Youlong Cheng
Google, Inc.

SONY

ImageNet/ResNet-50 Training in 224 Seconds

Hiroaki Mikami, Hisahiro Suganuma, Pongsakorn U-chupala, Yoshiki Tanaka and Yuichi Kageyama
Sony Corporation

FUJITSU

Yet Another Accelerated SGD: ResNet-50 Training on ImageNet in 74.7 seconds

Masafumi Yamazaki, Akihiko Kasagi, Akihiro Tabuchi, Takumi Honda, Masahiro Miwa, Naoto Fukumoto, Tsuguchika Tabaru, Atsushi Ike, Kohta Nakashima
Fujitsu Laboratories Ltd.

	#GPU/TPU	time	epochs
Facebook	512	30 min	90
UC Berkeley	2048	20 min	90
PFN	1024	15 min	90
Tencent	2048	6.6 min	90
Sony	2048	3.7 min	90
Google	1024	2.2 min	90
Our work	2048	2.0 min	45
Fujitsu	3456	1.2 min	90

Rich Information is Affordable: A Systematic Performance Analysis of Second-order Optimization Using K-FAC

Yuichiro Ueno
ueno.y.ai@m.titech.ac.jp
Tokyo Institute of Technology
AIST-Tokyo Tech RWBC-OIL, AIST
Tokyo, Japan

Kazuki Osawa
oosawa.k.ad@m.titech.ac.jp
Tokyo Institute of Technology
Tokyo, Japan

Yohei Tsuji
tsuji.y.ae@m.titech.ac.jp
Tokyo Institute of Technology
Tokyo, Japan

Akira Naruse
anaruse@nvidia.com
NVIDIA
Tokyo, Japan

Rio Yokota
rioyokota@gsic.titech.ac.jp
Tokyo Institute of Technology
AIST-Tokyo Tech RWBC-OIL, AIST
Tokyo, Japan

We were able to converge in fewer steps, but each step was more expensive

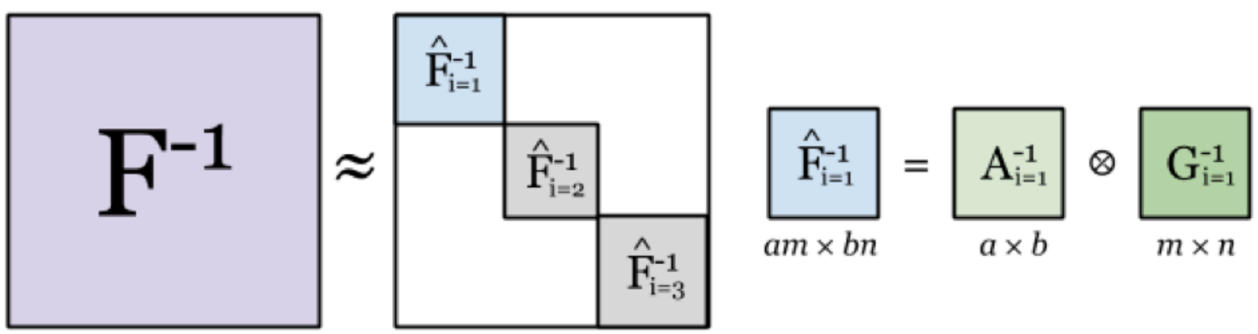
Followup work by Pauloski et al. at SC'20 and SC'21

Followup work by Pauloski et al.

SC'20

Convolutional Neural Network Training with Distributed K-FAC

J. Gregory Pauloski‡, Zhao Zhang*, Lei Huang*, Weijia Xu*, Ian T. Foster¶
*Texas Advanced Computing Center
Email: zzhang, huang, xwj@tacc.utexas.edu
‡University of Texas at Austin
Email: jgpauloski@utexas.edu
¶University of Chicago & Argonne National Laboratory
Email: foster@uchicago.edu



They use layer-wise block diagonalization and Kronecker factorization like we do

Instead of Cholesky factorization they use spectral decomposition

$$\mathbf{A} \otimes \mathbf{B} = (\mathbf{Q_A} \otimes \mathbf{Q_B})(\mathbf{D_A} \otimes \mathbf{D_B})(\mathbf{Q_A^\top} \otimes \mathbf{Q_B^\top})$$

SC'21

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

J. Gregory Pauloski
University of Chicago

Qi Huang
University of Texas at Austin

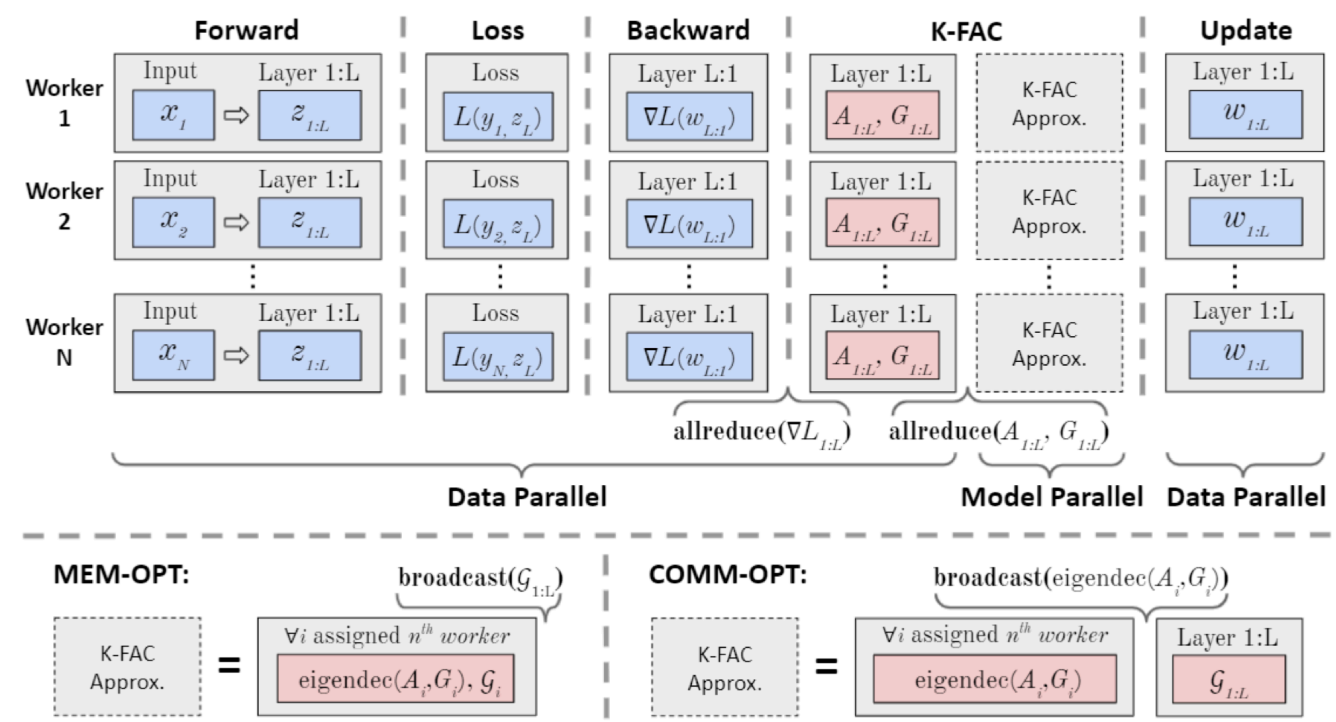
Lei Huang
Texas Advanced Computing Center

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

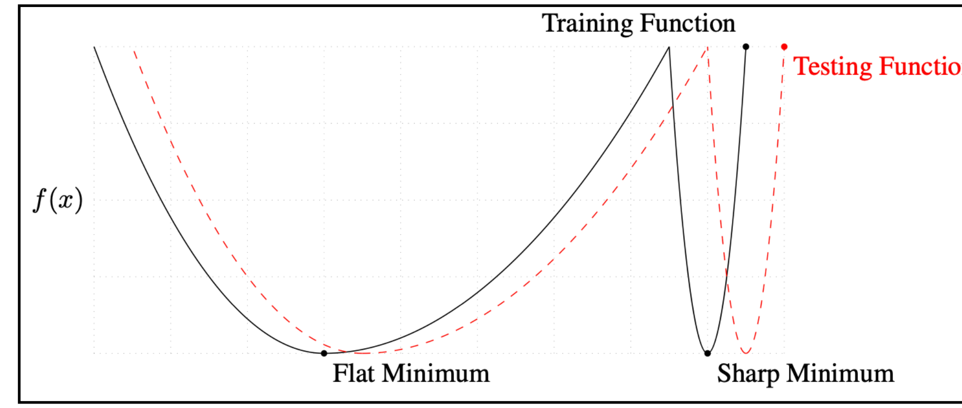


Minimize either memory usage or communication volume by choosing different decompositions of the matrix computations

Some myths regarding large-batch training

Large-batch training leads to sharp minima, which leads to poor generalization

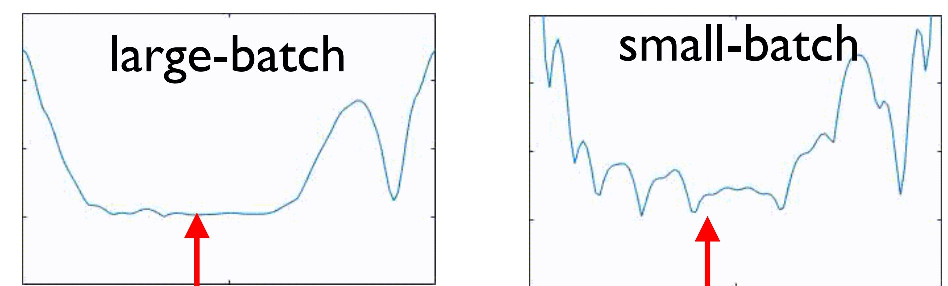
→Flat minima do not always generalize



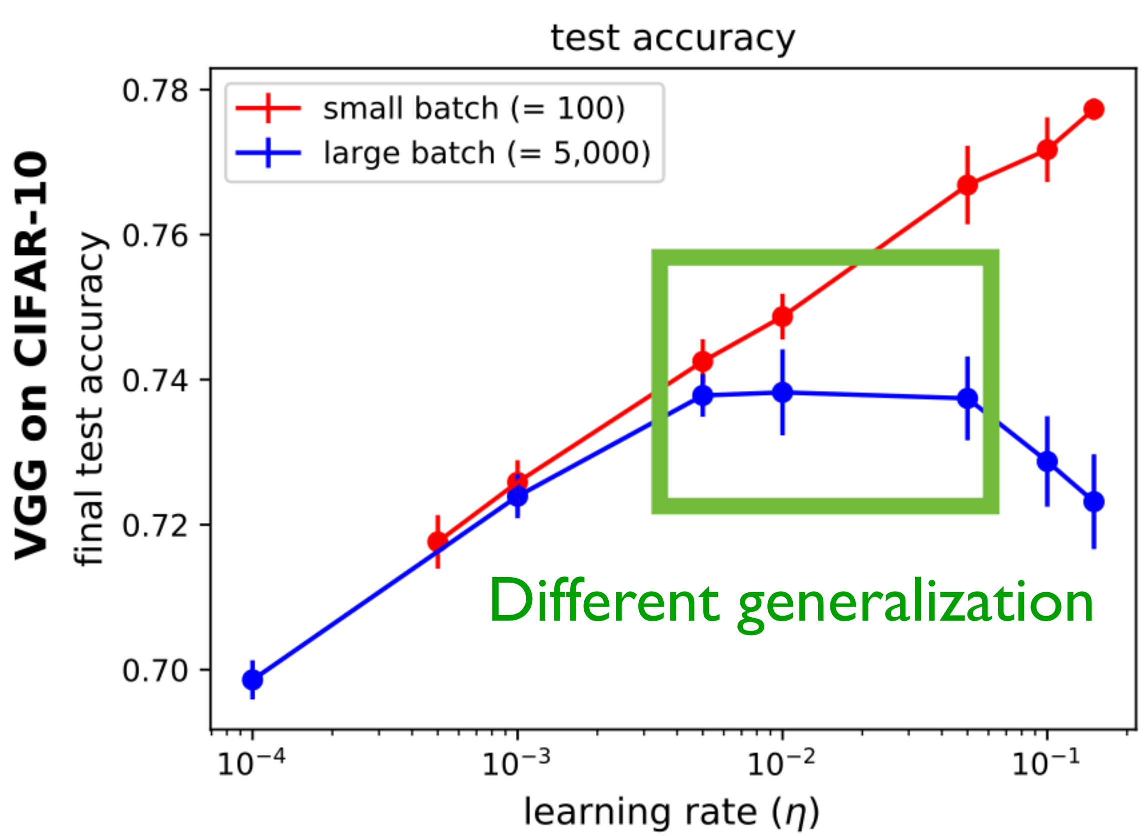
On the Maximum Hessian Eigenvalue and Generalization

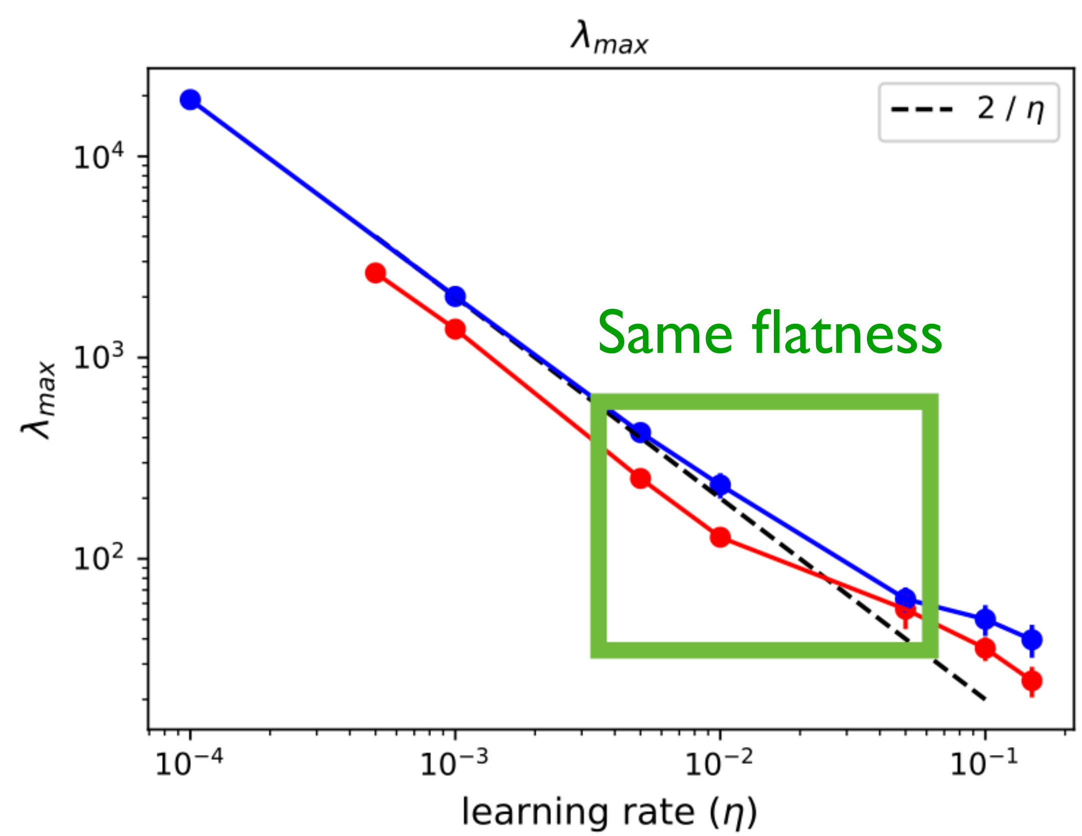
Simran Kaur†, Jeremy Cohen†, Zachary C. Lipton†

†Carnegie Mellon University



This flat minima is not actually flat





Some myths regarding large-batch training

Large-batch training leads to sharp minima,
which leads to poor generalization
→ Full-batch training can generalize as good as mini-batch training

Stochastic Training is Not Necessary for Generalization

Jonas Geiping
University of Siegen
jonas.geiping@uni-siegen.de

Micah Goldblum
University of Maryland, College Park
goldblum@umd.edu

Phillip E. Pope
University of Maryland, College Park
pepope@cs.umd.edu

Michael Moeller
University of Siegen
michael.moeller@uni-siegen.de

Tom Goldstein
University of Maryland, College Park
tomg@umd.edu

$$L(\theta) + \frac{\tau}{4}\underline{||\nabla L(\theta)||}^2$$

Regularize with L2-norm of gradient (not weight)

Similar to Sharpness Aware Minimization (SAM)

Full batch training on CIFAR-10

Experiment	ResNet-18	ResNet-50	Resnet-152	DenseNet-121
Baseline SGD	95.70	95.83	95.98	95.84
Baseline FB	75.42	54.32	58.62	76.87
FB train longer	87.36	83.31	91.02	82.06
FB clipped	93.85	94.15	91.41	93.44
FB regularized	95.36	95.51	95.82	95.47
FB strong reg.	95.67	96.05	96.01	95.81
FB in practice	95.91	96.56	96.76	95.86

H,F,C Matrices in deep learning

Critical Batch Size

An Empirical Model of Large-Batch Training

Sam McCandlish[*] OpenAI sam@openai.com
Jared Kaplan Johns Hopkins University, OpenAI jaredk@jhu.edu
Dario Amodei OpenAI damodei@openai.com
and the **OpenAI Dota Team**[†]

$$\mathcal{B}_{noise} = \frac{tr(\mathbf{H}\mathbf{C}^{-1})}{\mathbf{J}^T\mathbf{H}\mathbf{J}}$$

Preconditioned Optimizers

When Does Preconditioning Help or Hurt Generalization?

[*]Shun-ichi Amari[†], Jimmy Ba[‡], Roger Grosse[‡], Xuechen Li[§], Atsushi Nitanda[¶], Taiji Suzuki[¶], Denny Wu[‡], Ji Xu[‖]

Gauss-Newton

$$\mathbf{F}(\theta)^{-1}\nabla\mathcal{L}(\theta) = \left\{\mathbf{J}_{f,\theta}^\top \mathcal{H}_{\ell,f}\mathbf{J}_{f,\theta}\right\}^{-1}\mathbf{J}_{f,\theta}^\top \frac{\partial\mathcal{L}(\theta)}{\partial f}$$

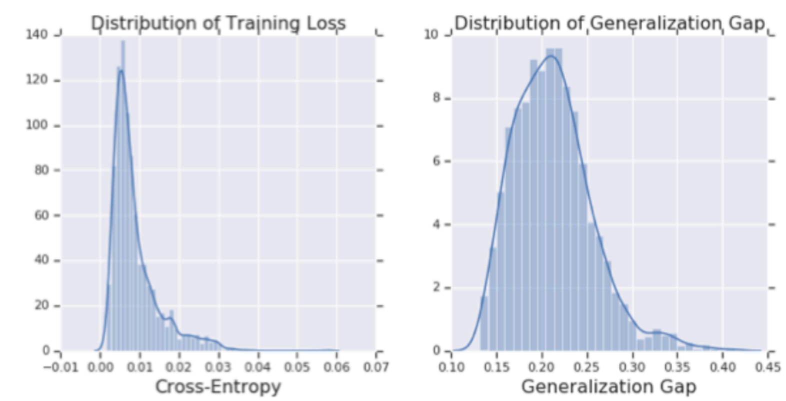
Gram-Gauss-Newton

$$\mathbf{F}(\theta)^{-1}\nabla\mathcal{L}(\theta) = \mathbf{J}_{f,\theta}^\top \left\{\mathcal{H}_{\ell,f}\mathbf{J}_{f,\theta}\mathbf{J}_{f,\theta}^\top\right\}^{-1}\frac{\partial\mathcal{L}(\theta)}{\partial f}$$

Generalization Metrics

Fantastic Generalization Measures and Where to Find Them

Yiding Jiang*, Behnam Neyshabur*, Hossein Mobahi
Dilip Krishnan, Samy Bengio
Google



- Spectral bound
- Path norm
- Fisher-Rao metric
- Variance of gradients
- Sharpness
- PAC-Baysian
- Takeuchi Information Criteria

$$\mathrm{TIC}(\theta) = -\log p(y|\theta) + \frac{1}{N}\mathrm{tr}\left(\boldsymbol{H}(\theta^*)^{-1}\boldsymbol{C}(\theta^*)\right)$$

Predicting Hyperparameters

Optimizing Millions of Hyperparameters by Implicit Differentiation

Jonathan Lorraine, Paul Vicol, David Duvenaud
University of Toronto, Vector Institute
{lorraine, pvicol, duvenaud}@cs.toronto.edu

$$\frac{\partial\theta}{\partial\lambda} = -\mathbf{H}^{-1}\frac{\partial^2\mathcal{L}}{\partial\theta\partial\lambda^\top}$$

Bayesian Inference Continual Learning

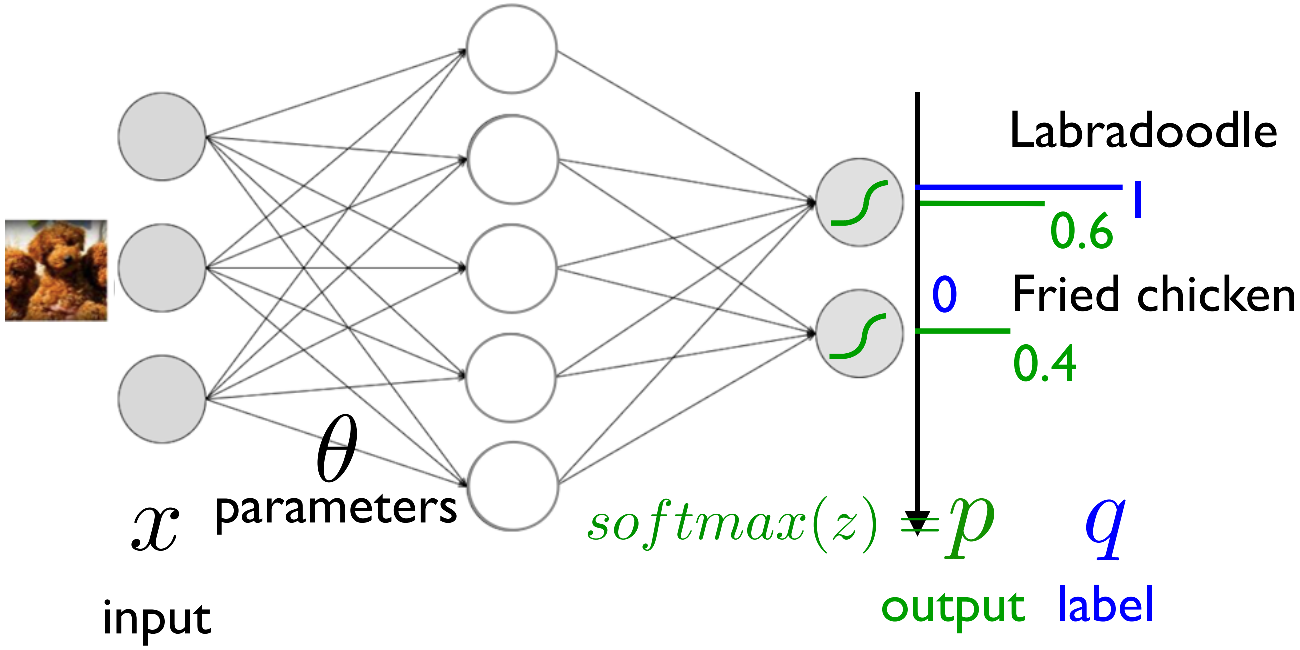
Noisy Natural Gradient as Variational Inference

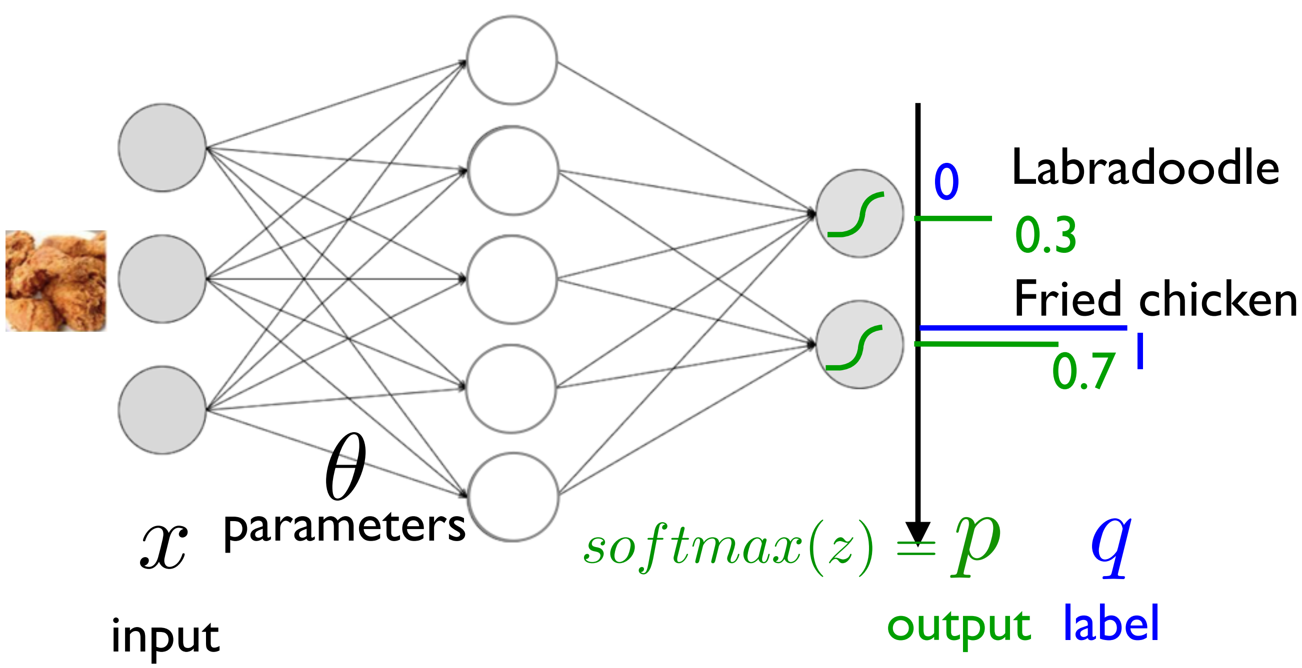
Guodong Zhang[*12] **Shengyang Sun**[*12] **David Duvenaud**[12] **Roger Grosse**[12]

$$\left\{\mathbf{F}(\theta) + \sigma^{-2}\mathbf{I}\right\}^{-1}\nabla\mathcal{L}(\theta)$$

$$= \left\{\mathbf{J}_{f,\theta}^\top\mathcal{H}_{\ell,f}\mathbf{J}_{f,\theta} + \sigma^{-2}\mathbf{I}\right\}^{-1}\mathbf{J}_{f,\theta}^\top\frac{\partial\mathcal{L}(\theta)}{\partial f}$$

What are H, G, F, C Matrices?





Negative log likelihood per class per data sample

$$l = -\log p$$

Overall loss

$$L = \sum^{data} \sum^{class} -q \log p = \sum^{data} -\log p$$

Hessian: Newton's method

$$H = \sum^{data} \sum^{class} q \frac{\partial^2 l}{\partial \theta^2}$$

Jacobian

$$J = \frac{\partial z}{\partial \theta}$$

Generalized Gauss-Newton: Gauss-Newton method

$$G = \sum^{data} \sum^{class} q \left(\frac{\partial z}{\partial \theta} \right) \left(\frac{\partial^2 l}{\partial z^2} \right) \left(\frac{\partial z}{\partial \theta} \right)^T$$

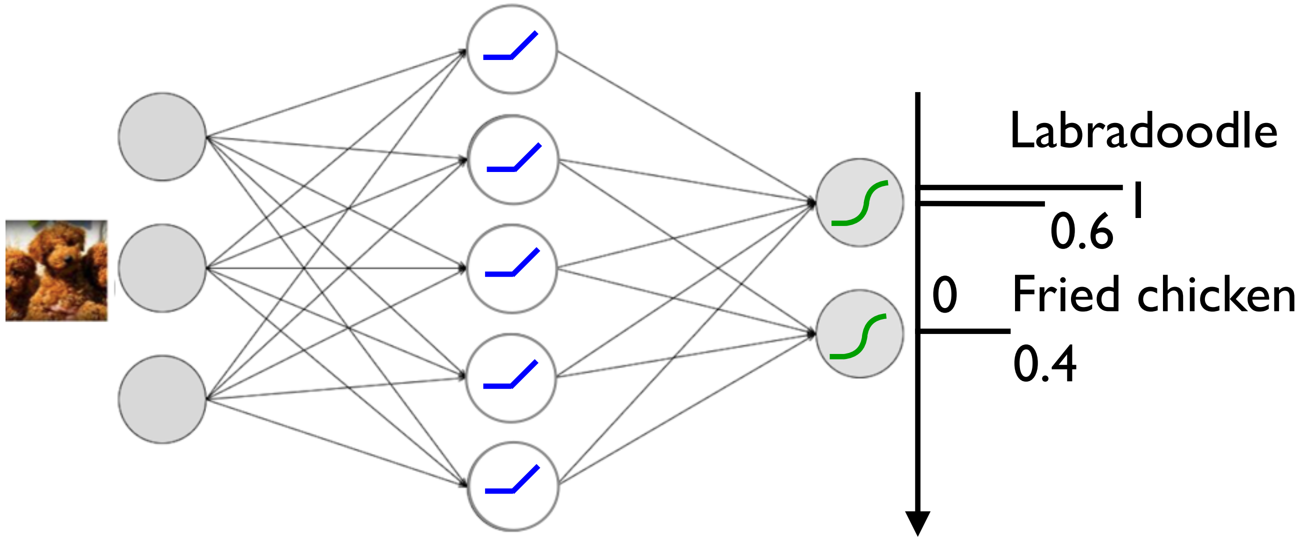
Fisher Information: Natural gradient descent

$$F = \sum^{data} \sum^{class} p \left(\frac{\partial l}{\partial \theta} \right) \left(\frac{\partial l}{\partial \theta} \right)^T$$

Uncentered covariance (empirical Fisher)

$$C = \sum^{data} \sum^{class} q \left(\frac{\partial l}{\partial \theta} \right) \left(\frac{\partial l}{\partial \theta} \right)^T$$

How matrices can be computed in PyTorch



$$x_0 \quad x_1 = ReLU(z_0) \quad p = softmax(z_1)$$

$$z_0 = W_0 x_0 \quad z_1 = W_1 x_1$$

Backward propagation

$$\frac{\partial x_1}{\partial z_0} * \frac{\partial z_1}{\partial x_1} * \frac{\partial l}{\partial z_1} = \frac{\partial l}{\partial \theta}$$

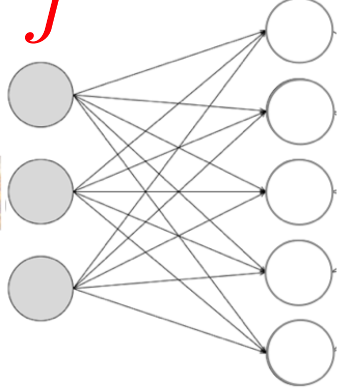
$$\otimes \qquad \otimes$$

$$\frac{\partial z_0}{\partial W_0} \qquad \frac{\partial z_1}{\partial W_1}$$

error signal

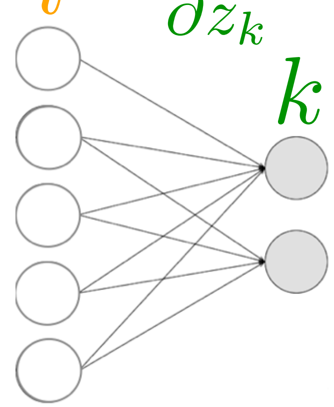
activation $x_j \frac{\partial l}{\partial z_i} = \frac{\partial l}{\partial W_{ij}}$

$j \quad i$



$l \quad x_l \frac{\partial l}{\partial z_k} = \frac{\partial l}{\partial W_{kl}}$

k



$$F = \sum^{data} \sum^{class} p \left(\frac{\partial l}{\partial \theta}\right) \left(\frac{\partial l}{\partial \theta}\right)^T$$

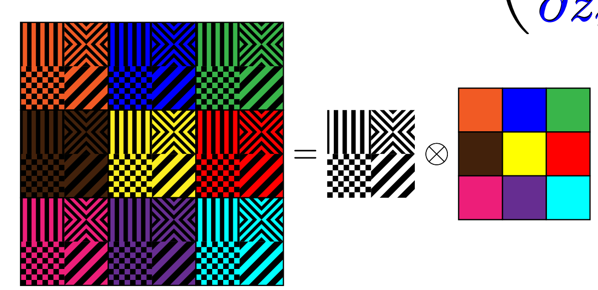
$$\frac{\partial l}{\partial \theta} = \begin{bmatrix} \frac{\partial l}{\partial W_0} \\ \frac{\partial l}{\partial W_1} \end{bmatrix}$$

$$\left(\frac{\partial l}{\partial \theta}\right) \left(\frac{\partial l}{\partial \theta}\right)^T = \begin{bmatrix} \frac{\partial l}{\partial W_0} \\ \frac{\partial l}{\partial W_1} \end{bmatrix} \begin{bmatrix} \frac{\partial l}{\partial W_0}^T & \frac{\partial l}{\partial W_1}^T \end{bmatrix}$$

$$= \begin{bmatrix} \left(\frac{\partial l}{\partial W_0}\right)\left(\frac{\partial l}{\partial W_0}\right)^T & \left(\frac{\partial l}{\partial W_0}\right)\left(\frac{\partial l}{\partial W_1}\right)^T \\ \left(\frac{\partial l}{\partial W_1}\right)\left(\frac{\partial l}{\partial W_0}\right)^T & \left(\frac{\partial l}{\partial W_1}\right)\left(\frac{\partial l}{\partial W_1}\right)^T \end{bmatrix}$$

$$\left(\frac{\partial l}{\partial W_{ij}}\right)\left(\frac{\partial l}{\partial W_{kl}}\right) = \left(x_j \frac{\partial l}{\partial z_i}\right)\left(x_l \frac{\partial l}{\partial z_k}\right)$$

$$= (x_j x_l)\left(\frac{\partial l}{\partial z_i} \frac{\partial l}{\partial z_k}\right)$$

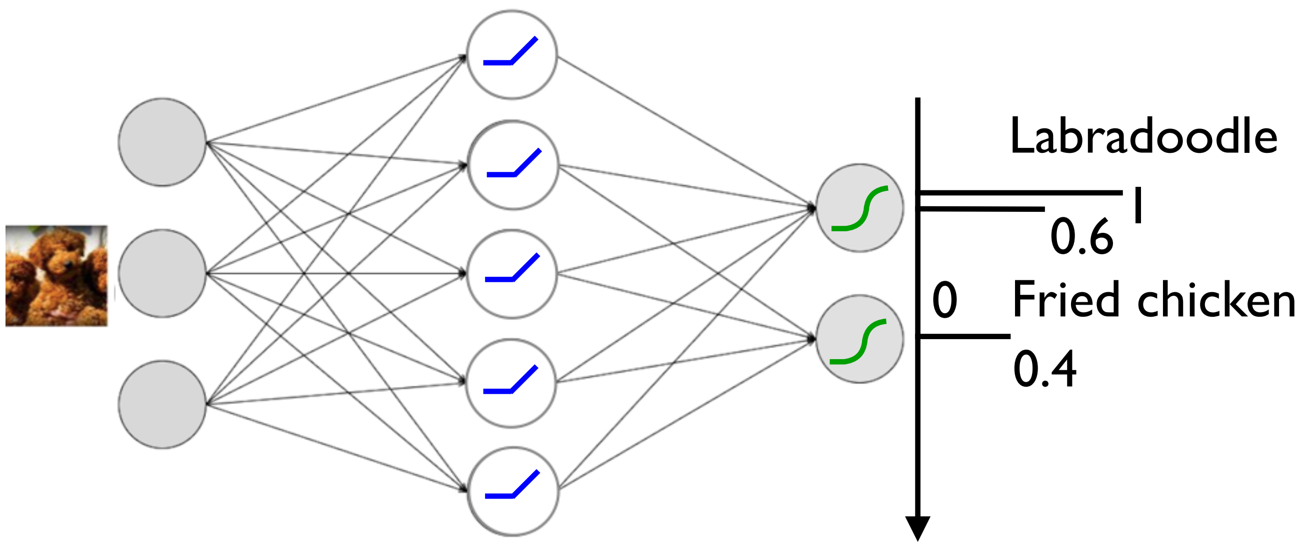


```
def forward_hook(self, in_data, out_data):
    in_data = in_data[0].clone().detach()
    def backward_hook(out_grads):
        self.in_data = in_data
        self.out_grads = out_grads
    out_data.register_hook(backward_hook)
for module in model.children():
    module.register_forward_hook(forward_hook)
```

$x_j \; x_l$

$\frac{\partial l}{\partial z_i} \; \frac{\partial l}{\partial z_k}$

Hessian = Generalized Gauss-Newton



Forward propagation

$$x_0 \searrow z_0 = W_0 x_0 \nearrow x_1 = ReLU(z_0) \searrow z_1 = W_1 x_1 \nearrow p = softmax(z_1)$$

Backward propagation

$$\frac{\partial x_1}{\partial z_0} * \frac{\partial z_1}{\partial x_1} * \frac{\partial l}{\partial z_1} = \frac{\partial l}{\partial \theta}$$

$$\otimes \frac{\partial z_0}{\partial W_0} \qquad \otimes \frac{\partial z_1}{\partial W_1}$$

Backprop is a Jacobian-vector product

$$\underbrace{\frac{\partial z_1}{\partial W_0}}_{15 \times 2} * \underbrace{\boxed{\frac{\partial l}{\partial z_1}}}_{2 \times 1} = \underbrace{\frac{\partial l}{\partial W_0}}_{15 \times 1}$$

Gradient of first layer per data sample

$$\frac{\partial l}{\partial W_0} = \frac{\partial z_0}{\partial W_0} * \frac{\partial x_1}{\partial z_0} * \frac{\partial z_1}{\partial x_1} * \frac{\partial l}{\partial z_1}$$

Hessian of first layer per data sample

$$\frac{\partial^2 l}{\partial W_0^2} = \frac{\partial^2 z_0}{\partial W_0^2} * \frac{\partial x_1}{\partial z_0} * \frac{\partial z_1}{\partial x_1} * \frac{\partial l}{\partial z_1} \rightarrow 0$$

$$+ \left(\frac{\partial z_0}{\partial W_0}\right)^2 * \frac{\partial^2 x_1}{\partial z_0^2} * \frac{\partial z_1}{\partial x_1} * \frac{\partial l}{\partial z_1} \rightarrow 0$$

$$+ \left(\frac{\partial z_0}{\partial W_0}\right)^2 * \left(\frac{\partial x_1}{\partial z_0}\right)^2 * \frac{\partial^2 z_1}{\partial x_1^2} * \frac{\partial l}{\partial z_1} \rightarrow 0$$

$$+ \left(\frac{\partial z_0}{\partial W_0}\right)^2 * \left(\frac{\partial x_1}{\partial z_0}\right)^2 * \left(\frac{\partial z_1}{\partial x_1}\right)^2 * \frac{\partial^2 l}{\partial z_1^2}$$

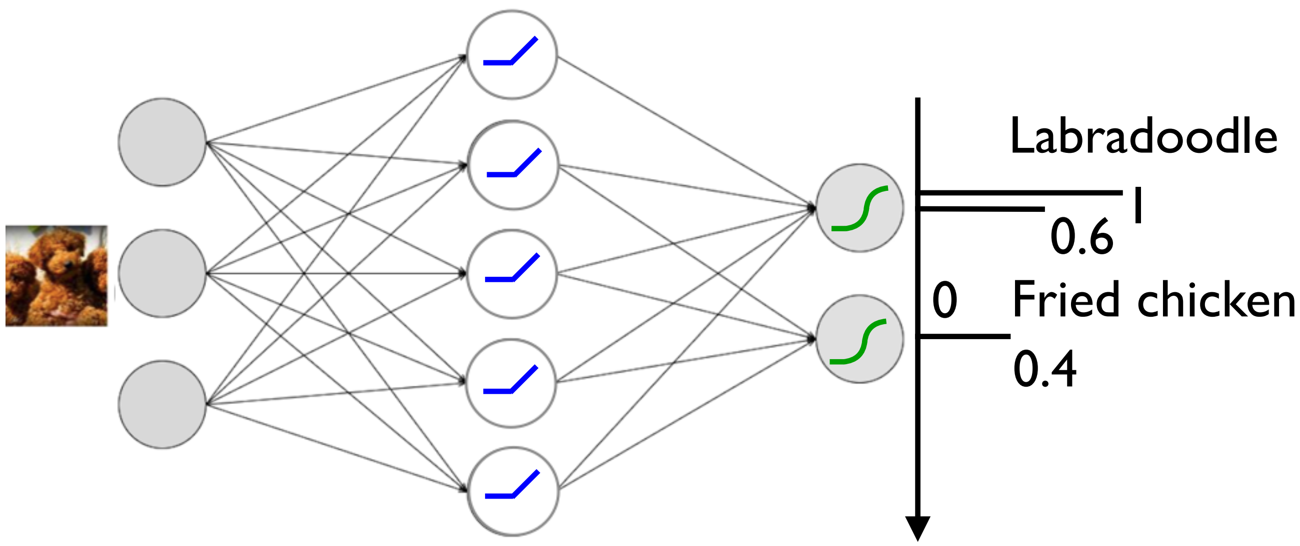
$$= \left(\frac{\partial z_1}{\partial W_0}\right) * \frac{\partial^2 l}{\partial z_1^2} * \left(\frac{\partial z_1}{\partial W_0}\right)^T = G$$

$$\underbrace{\frac{\partial z_1}{\partial W_0}}_{15 \times 2} * \underbrace{\boxed{\sqrt{\frac{\partial^2 l}{\partial z_1^2}}}}_{2 \times 2}$$

Jacobian-matrix product can be Backproped

Cholesky factorization

Generalized Gauss-Newton = Fisher



$$p = softmax(z)$$

Negative log likelihood per class per data sample

$$l_k = -\log p_k$$

$$\frac{\partial z}{\partial \theta} * \frac{\partial l}{\partial z} = \frac{\partial l}{\partial \theta}$$

Derivative with respect to p

$$\frac{\partial l_k}{\partial p_i} = -\frac{\delta_{ik}}{p_k}$$

$$G = \sum^{data} \left(\frac{\partial z}{\partial \theta}\right) q \frac{\partial^2 l}{\partial z^2} \left(\frac{\partial z}{\partial \theta}\right)^T$$

$$F = \sum^{data} \left(\frac{\partial z}{\partial \theta}\right) \sum^{class} p \left(\frac{\partial l}{\partial z}\right) \left(\frac{\partial l}{\partial z}\right)^T \left(\frac{\partial z}{\partial \theta}\right)^T$$

Thus, for softmax cross-entropy we have

$$G = F$$

Softmax cross entropy for the ith class

$$p_i = exp(z_i) / \sum_k^{class} exp(z_k)$$

Jacobian of that with respect to z

$$\frac{\partial p_i}{\partial z_j} = p_i(\delta_{ij} - p_j)$$

Jacobian of the softmax cross-entropy (loss per class)

$$\frac{\partial l_k}{\partial z_j} = \frac{\partial l_k}{\partial p_i}\frac{\partial p_i}{\partial z_j} = \left(-\frac{\delta_{ik}}{p_k}\right) p_i(\delta_{ij} - p_j)$$

$$= p_j - \delta_{kj}$$

GGN of the softmax cross-entropy

$$q_k \frac{\partial}{\partial z_i}\frac{\partial l_k}{\partial z_j} = \frac{\partial p_j}{\partial z_i} = p_j(\delta_{ij} - p_i)$$

Fisher of the softmax cross-entropy

$$\sum_k^{class} p_k \frac{\partial l_k}{\partial z_i}\frac{\partial l_k}{\partial z_j} = \sum_k^{class} p_k(p_i - \delta_{ki})(p_j - \delta_{kj})$$

$$= \sum_k^{class} (p_i p_j p_k + p_k \delta_{ki}\delta_{kj}) - 2 p_i p_j$$

$$= p_j(\delta_{ij} - p_i)$$

Einstein summation for k is explicit here

If weren't able to follow the equations

Loss function (negative log likelihood)

$$l = -\log p$$

Gradient

$$\frac{\partial l}{\partial p} = -\frac{1}{p}$$

Hessian

$$\frac{\partial^2 l}{\partial p^2} = \frac{1}{p^2}$$

Fisher

$$\left(\frac{\partial l}{\partial p}\right)^2 = \frac{1}{p^2}$$

Hessian

$$H = \sum^{data} \sum^{class} q \frac{\partial^2 l}{\partial \theta^2}$$

Gauss-Newton / Fisher (exact)

$$G = F = \sum^{data} \sum^{class} p \left(\frac{\partial l}{\partial \theta}\right) \left(\frac{\partial l}{\partial \theta}\right)^T$$

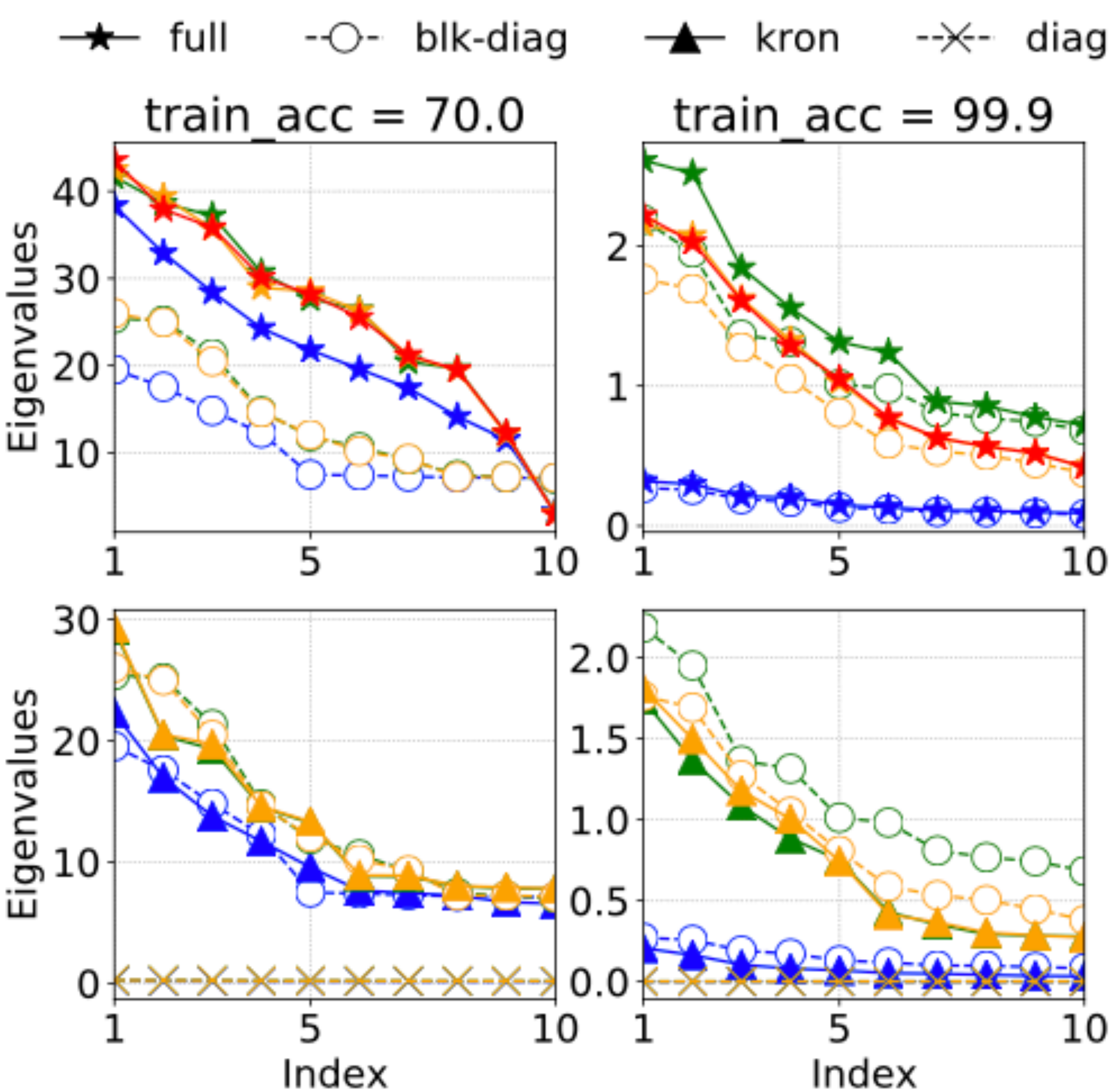
Fisher (Monte-Carlo sampling)

$$F_{mc} = \sum^{data} \sum^{sample} p \left(\frac{\partial l}{\partial \theta}\right) \left(\frac{\partial l}{\partial \theta}\right)^T$$

Uncentered covariance (empirical Fisher)

$$C = \sum^{data} \sum^{class} q \left(\frac{\partial l}{\partial \theta}\right) \left(\frac{\partial l}{\partial \theta}\right)^T$$

How good are these approximations?



MLP on MNIST

Hessian

$$H = \sum^{data} \sum^{class} q \frac{\partial^2 l}{\partial \theta^2}$$

Gauss-Newton / Fisher (exact)

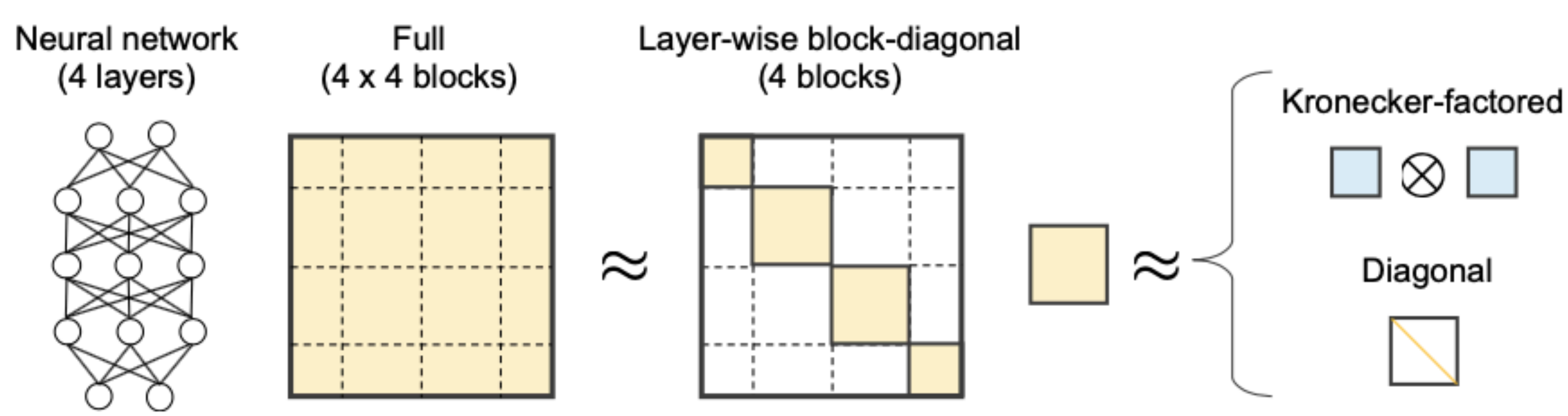
$$G = F = \sum^{data} \sum^{class} p \left(\frac{\partial l}{\partial \theta}\right) \left(\frac{\partial l}{\partial \theta}\right)^T$$

Fisher (Monte-Carlo sampling)

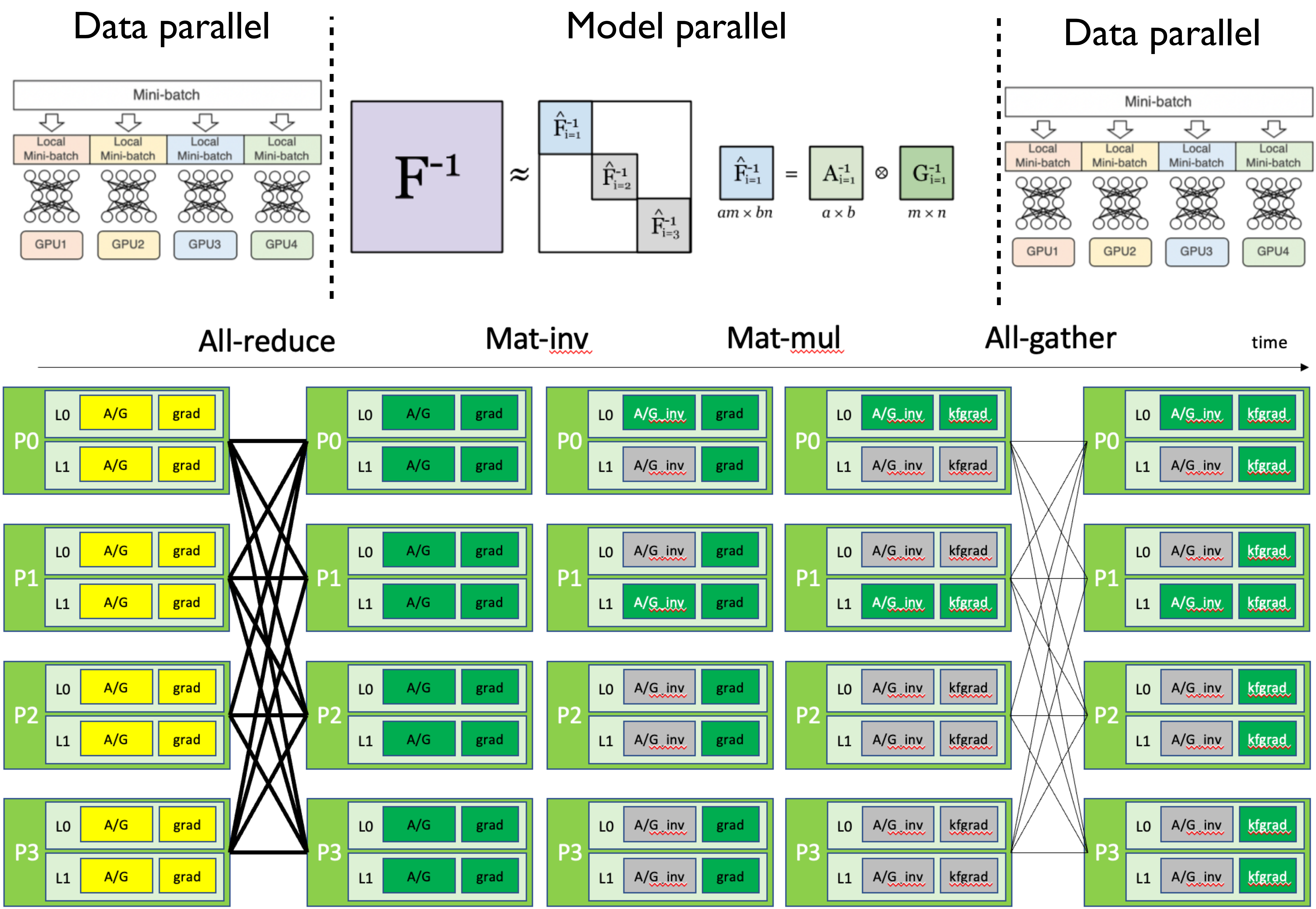
$$F_{mc} = \sum^{data} \sum^{sample} p \left(\frac{\partial l}{\partial \theta}\right) \left(\frac{\partial l}{\partial \theta}\right)^T$$

Uncentered covariance (empirical Fisher)

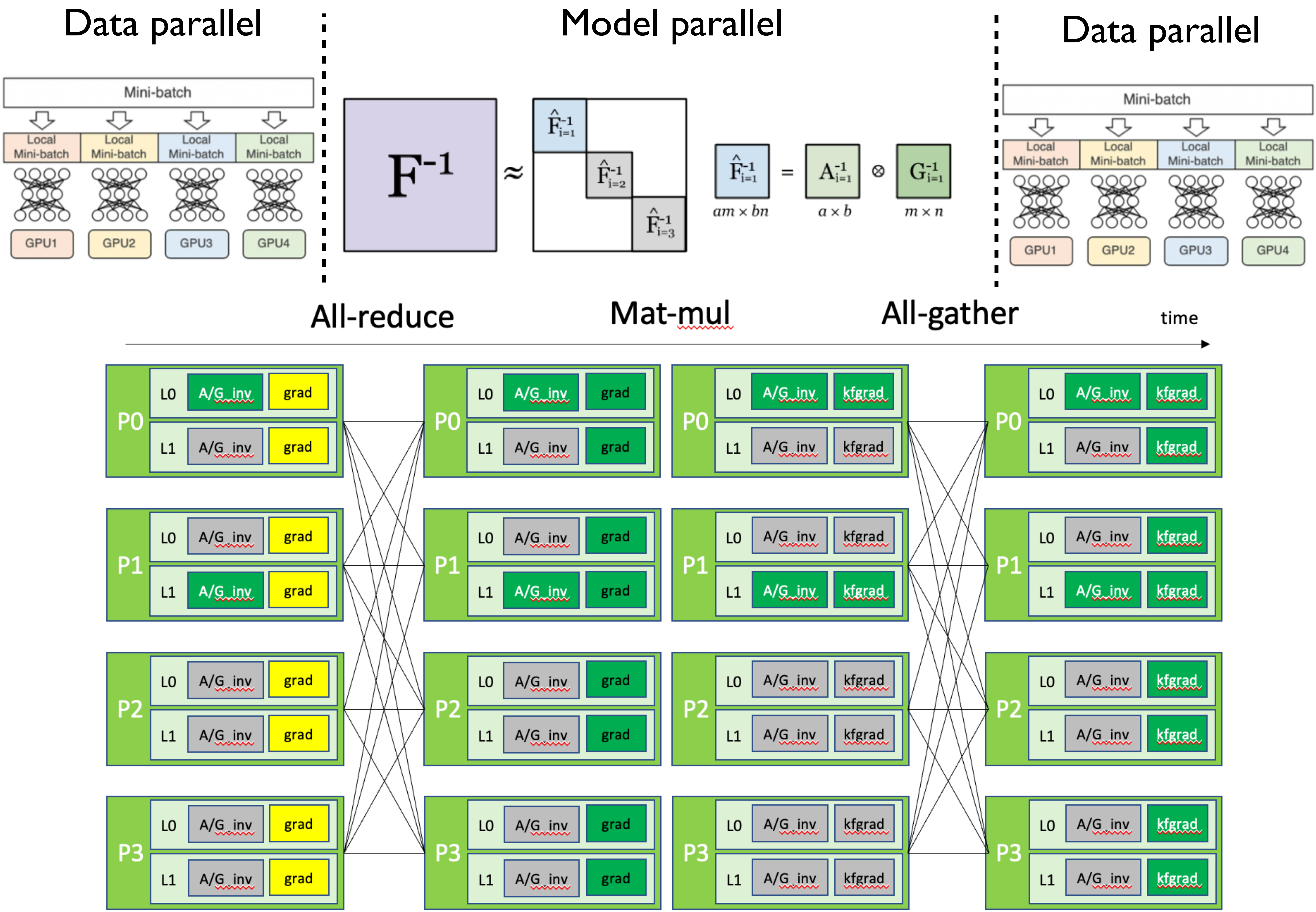
$$C = \sum^{data} \sum^{class} q \left(\frac{\partial l}{\partial \theta}\right) \left(\frac{\partial l}{\partial \theta}\right)^T$$



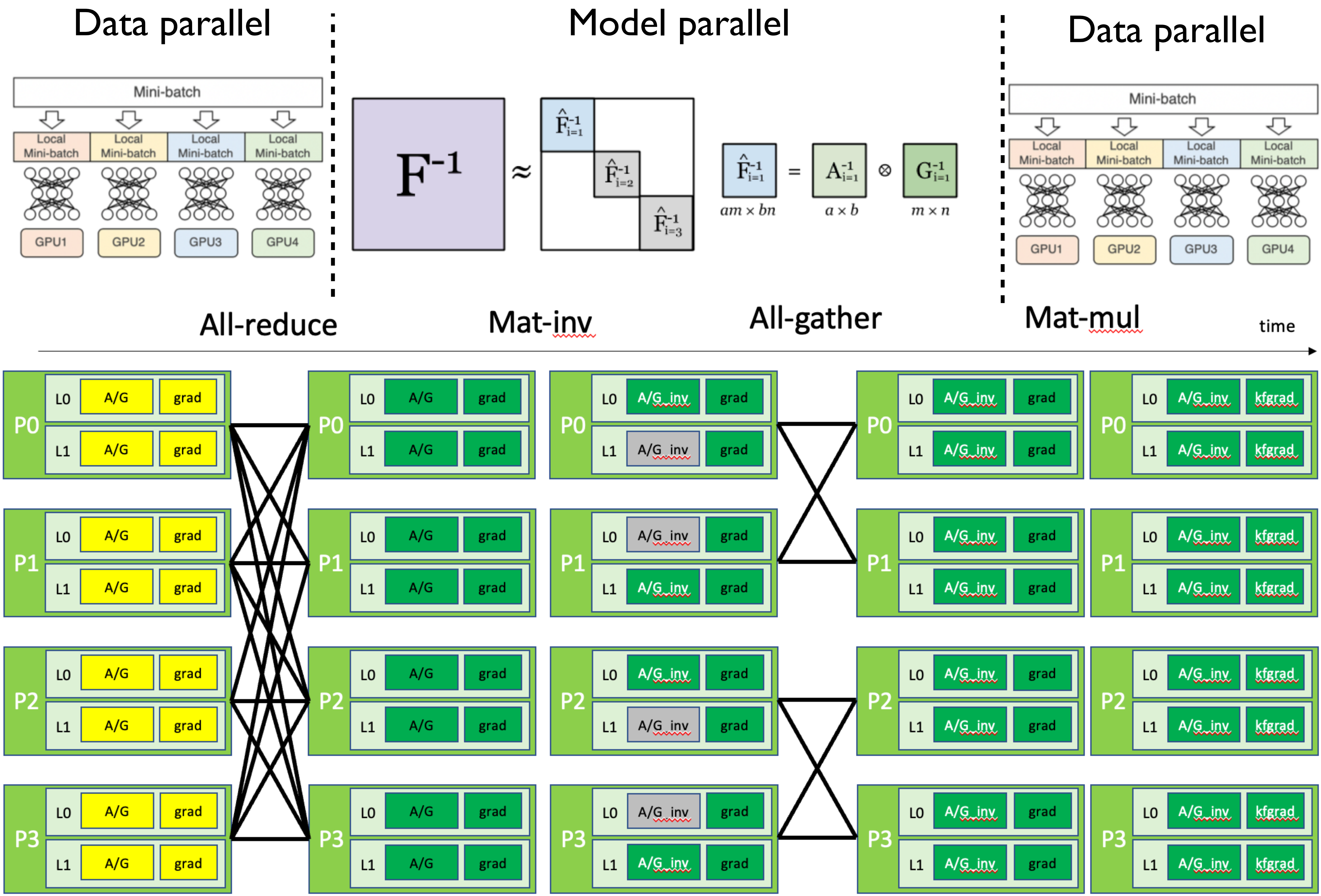
Distributed implementation



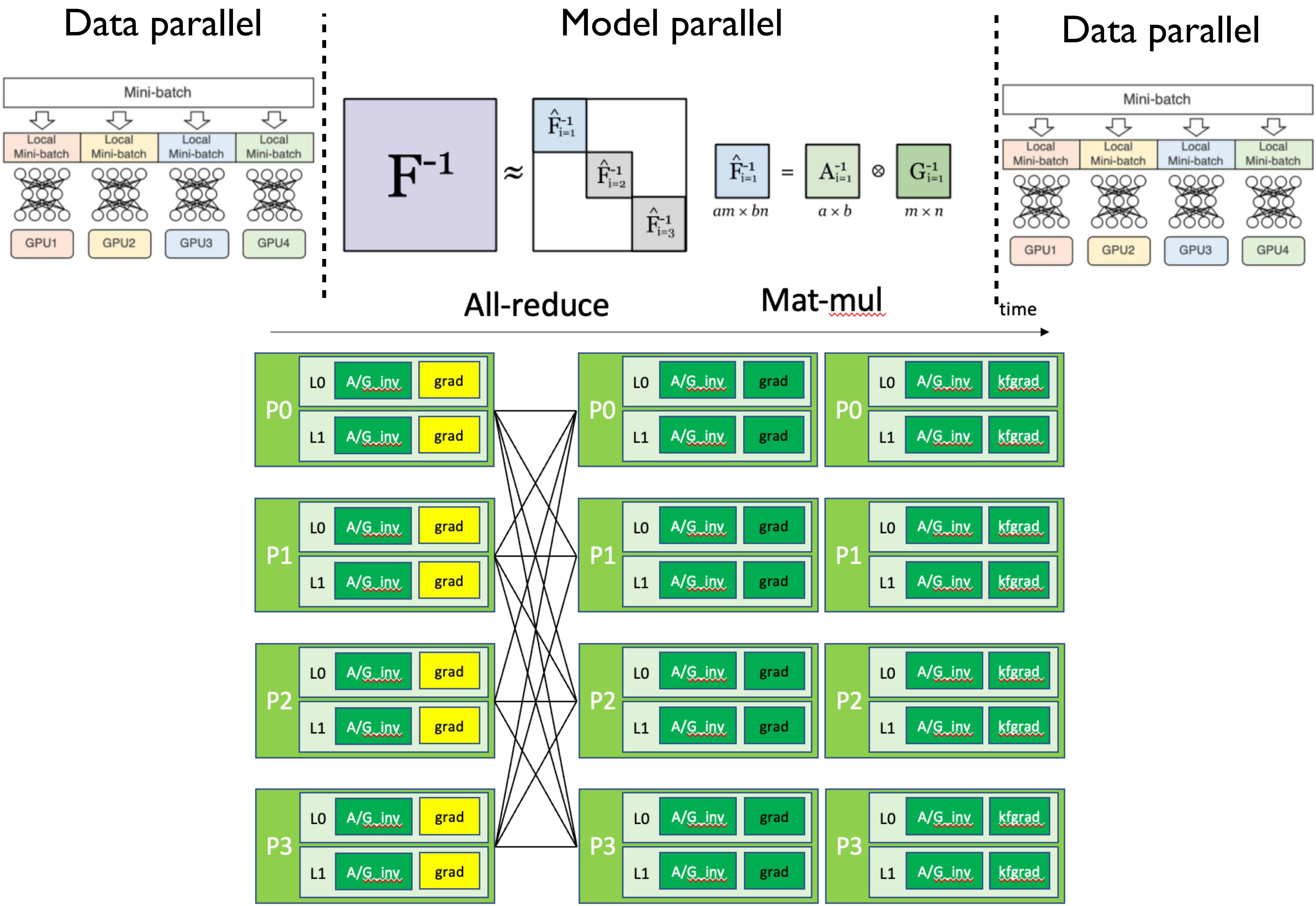
Use stale A/G_inv



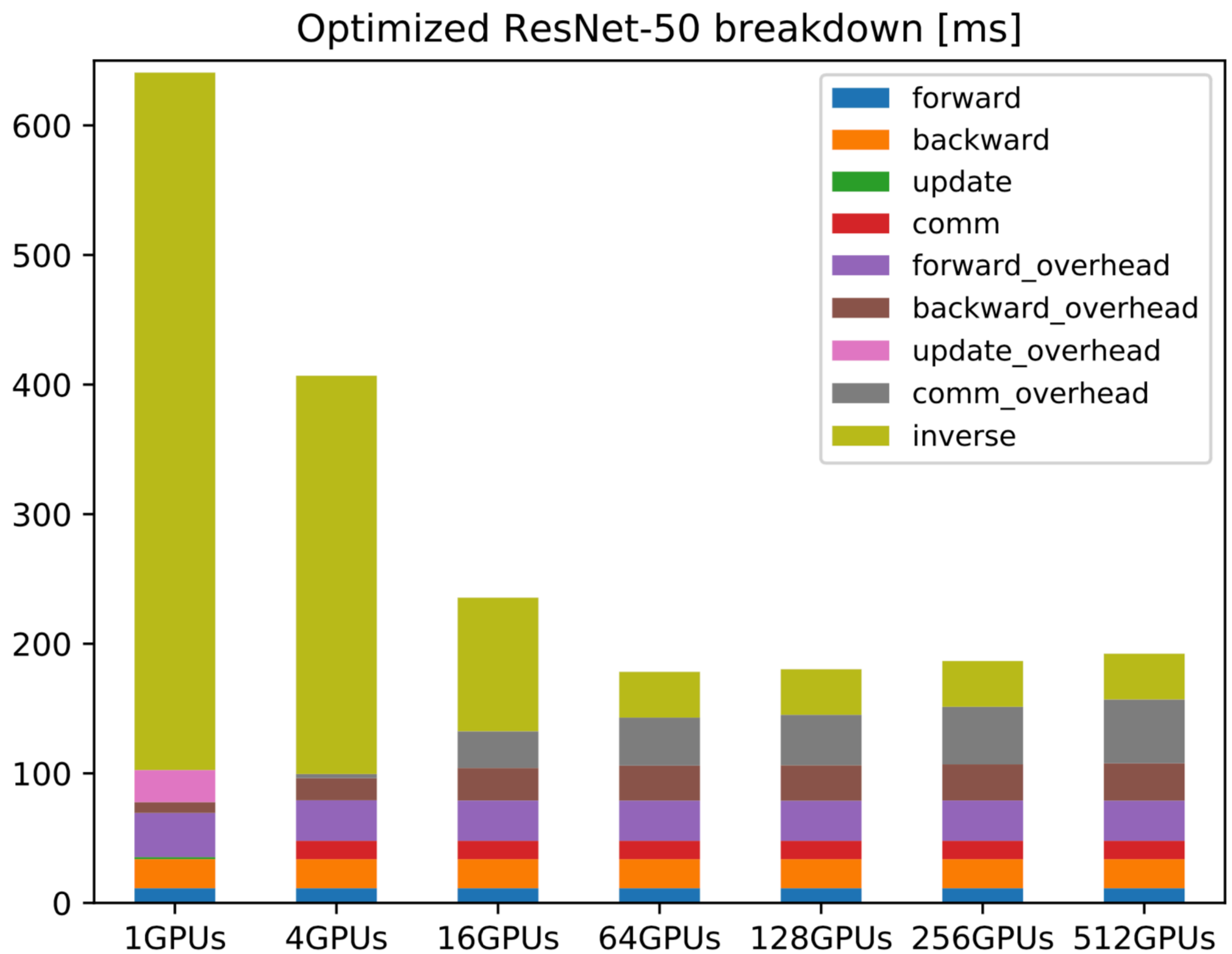
Distributed implementation 2

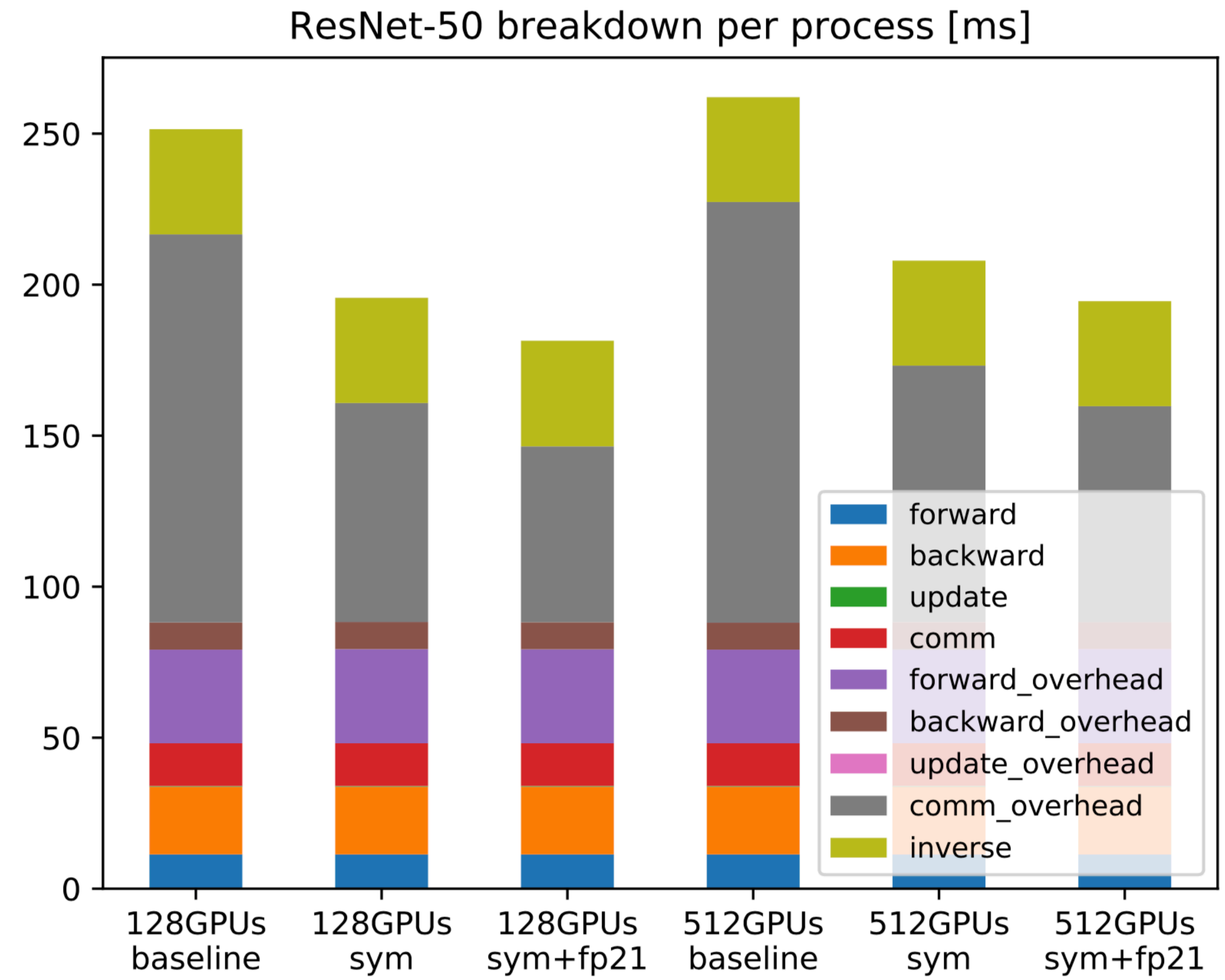


Use stale A/G_inv



Scalability of matrix computation in DNNs

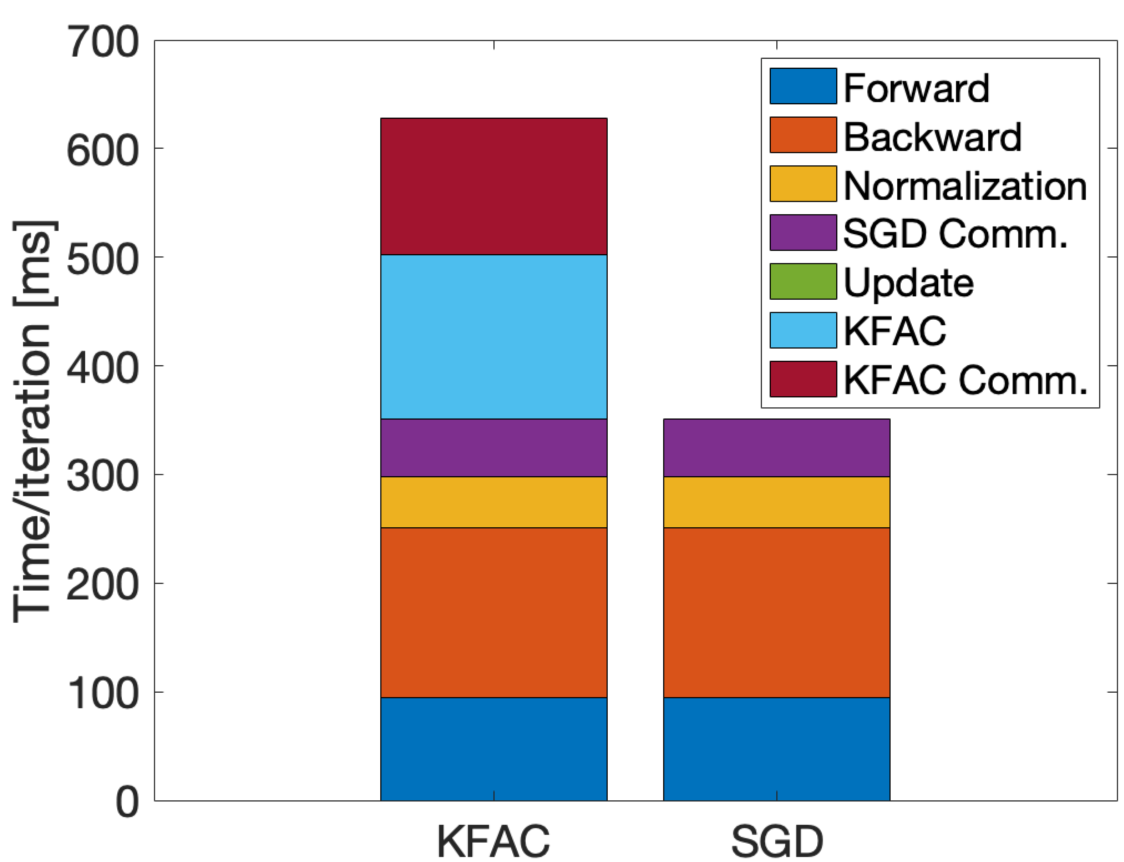




# of GPUs	Mini-batch size	Optimizer	Stale FIM	# of Updates	Time	Accuracy
128	4,096	SGD	-	28,151	26.7 min.	-
		K-FAC	not applied	10,920	38.6 min.	76.1 %
		K-FAC	✓	10,920	**21.5** min.	75.7 %
256	8,192	SGD	-	14,076	13.4 min.	-
		K-FAC	not applied	5,460	19.8 min.	76.0%
		K-FAC	✓	5,460	**9.4** min.	75.5%
512	16,384	SGD	-	7,038	6.7 min.	-
		K-FAC	not applied	2,730	10.5 min.	76.0 %
		K-FAC	✓	2,730	**5.5** min.	74.9 %
2048	65,536	K-FAC	✓	1,178	2.7 min.	75.6 %
	81,920		✓	795	2.0 min.	75.0 %

Summary

- Hessian, Gauss-Newton, and Fisher matrices play an important role in the theory of deep learning
- In many common DNNs Hessian = Gauss-Newton = Fisher
- K-FAC has the best balance between approximation accuracy and computation cost
- Distributed parallelism reduces the overhead of matrix computations drastically



Acknowledgements
Kazuki Osawa (ETH)
Yuichiro Ueno (PFN)
Yohei Tsuji (AWS)
Akira Naruse (NVIDIA)