# Scalable and Practical Natural Gradient for Large-Scale Deep Learning

Kazuki Osawa, *Student Member, IEEE*, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Chuan-Sheng Foo, and Rio Yokota

**Abstract**—Large-scale distributed training of deep neural networks results in models with worse generalization performance as a result of the increase in the effective mini-batch size. Previous approaches attempt to address this problem by varying the learning rate and batch size over epochs and layers, or *ad hoc* modifications of batch normalization. We propose scalable and practical natural gradient descent (SP-NGD), a principled approach for training models that allows them to attain similar generalization performance to models trained with first-order optimization methods, but with accelerated convergence. Furthermore, SP-NGD scales to large mini-batch sizes with a negligible computational overhead as compared to first-order methods. We evaluated SP-NGD on a benchmark task where highly optimized first-order methods are available as references: training a ResNet-50 model for image classification on ImageNet. We demonstrate convergence to a top-1 validation accuracy of 75.4 percent in 5.5 minutes using a mini-batch size of 32,768 with
1,024 GPUs, as well as an accuracy of 74.9 percent with an extremely large mini-batch size of 131,072 in 873 steps of SP-NGD.

**Index Terms**—Natural gradient descent, distributed deep learning, deep convolutional neural networks, image classification

## 1 Introduction

As the size of deep neural network models and the data which they are trained on continues to increase rapidly, the demand for distributed parallel computing is increasing. A common approach for achieving distributed parallelism in deep learning is to use the data-parallel approach, where the data is distributed across different processes while the model is replicated across them. When the mini-batch size per process is kept constant to increase the ratio of computation over communication, the effective mini-batch size over the entire system grows proportional to the number of processes.

Keskar *et al.* [1] report that when the mini-batch size is "large", generalization performance is worse than when a smaller mini-batch is used. Hoffer *et al.* [2] attribute this generalization gap to the limited number of updates, and suggest to train longer. Goyal *et al.* [3] adopt strategies such as scaling the learning rate proportional to the mini-batch size, while using the first few epochs to gradually warmup the learning rate. Such methods have enabled the training for mini-batch sizes of 8K, where ImageNet [4] with ResNet-50 [5] could be trained for 90 epochs with little reduction in generalization performance (76.3 percent top-1 validation accuracy) in 60 minutes. They also report that when the mini-batch size is increased beyond a certain point ( > 8K,) the generalization performance starts to degrade even with their strategies. Shallue *et al.* [6], with thorough hyper-parameter tuning for various models and datasets, empirically show that there is no evidence that larger mini-batch sizes degrade the generalization performance. However, for training ResNet-50 on ImageNet, their results agree that larger mini-batch sizes ( > 8K) degrade the generalization performance without the use of label smoothing [7].

Since Goyal *et al.* [3]'s results were presented, there have been many attempts to train ResNet-50 on ImageNet with mini-batch sizes larger than 16K. Combining the learning rate scaling with other techniques such as RMSprop [8] warm-up, BatchNorm [9] without moving averages, and a slow-start learning rate schedule, Akiba *et al.* [10] were able to train the same dataset and model with a mini-batch size of 32 K to achieve 74.9 percent accuracy in 15 minutes.

More complex approaches for manipulating the learning rate were proposed, such as LARS [11], where a different learning rate is used for each layer by normalizing them with the ratio between the layer-wise norms of the weights and gradients. This enabled the training with a mini-batch size of 32 K without the use of *ad hoc* modifications, which achieved 74.9 percent accuracy in 14 minutes (64 epochs) [11]. It has been reported that combining LARS with counter intuitive modifications to the batch normalization, can yield 75.8 percent accuracy even for a mini-batch size of 65 K [12].

The use of small batch sizes to encourage rapid convergence in early epochs, and then progressively increasing the batch size is yet another successful approach. Using such an adaptive batch size method, Mikami *et al.* [13] were able

- *Kazuki Osawa and Yohei Tsuji are with the Tokyo Institute of Technology, Tokyo 152-8550, Japan. E-mail: {oosawa.k.ad, tsuji.y.ae}@m.titech.ac.jp.*
- *Yuichiro Ueno and Rio Yokota are with the Tokyo Institute of Technology, Tokyo 152-8550, Japan, and also with the AIST-Tokyo Tech RWBC-OIL, AIST, Tokyo 152-8550, Japan.*
*E-mail: ueno.y.ai@m.titech.ac.jp, rioyokota@gsic.titech.ac.jp.*
- *Akira Naruse is with the NVIDIA, Tokyo 107-0052, Japan.*
*E-mail: anaruse@nvidia.com.*
- *Chuan-Sheng Foo is with the Institute for Infocomm Research, A*STAR, Singapore 138632. E-mail: foo_chuan_sheng@i2r.a-star.edu.sg.*

*Manuscript received 13 Oct. 2019; revised 11 Mar. 2020; accepted 19 June 2020.*
*Date of publication 23 June 2020; date of current version 3 Dec. 2021.*
*(Corresponding author: Kazuki Osawa.)*
*Recommended for acceptance by N. L. Roux.*
*Digital Object Identifier no. 10.1109/TPAMI.2020.3004354*

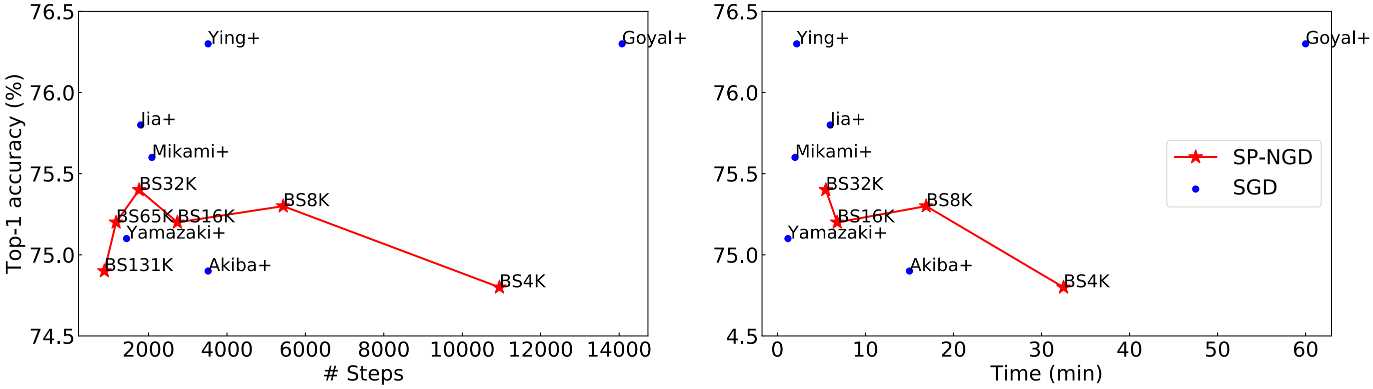

Fig. 1. Top-1 validation accuracy versus the number of steps to converge (left) and versus training time (right) of ResNet-50 on ImageNet (1000 class) classification by related work with SGD and this work with *Scalable and Practical NGD* (SP-NGD).

to train in 122 seconds with an accuracy of 75.3 percent, and Yamazaki *et al.* [14] were able to train in 75 seconds with a accuracy of 75.1 percent. The hierarchical synchronization of mini-batches have also been proposed [15], but such methods have not been tested at scale to the extent of the authors' knowledge.

While first-order stochastic gradient descent (SGD) has been the dominant approach for training deep neural networks, we explore the potential for second-order optimization methods such as Natural Gradient Descent (NGD) [17] that leverage curvature information to accelerate optimization in large mini-batch settings. We found that NGD enables training with a fewer number of steps than what was previously thought to be possible with SGD, while retaining competitive generalization performance with the help of stronger data augmentation i.e., *mixup* [18]. Note that this kind of data augmentation helps NGD much more than it does SGD, so we are not simply increasing the baseline accuracy. We minimize the extra per-step computational overhead associated with inverting the curvature matrices using an efficient distributed layer-wise NGD design that scales to massively parallel settings and large mini- batch sizes. In particular, we demonstrate scalability to batch sizes of 32,768 over 1,024 GPUs across 256 nodes. We adopt the Kronecker-Factored Approximate Curvature (K-FAC) method [19] for its ability to efficiently approximate the curvature, in contrast to iterative approaches like Hessian-free optimization [20], even though these may yield more accurate curvature approximation. The two main characteristics of K-FAC are that it converges faster than first-order stochastic gradient descent (SGD) methods, and that it can tolerate relatively large mini-batch sizes without any *ad hoc* modifications. K-FAC has been successfully applied to convolutional neural networks [21], distributed memory training of ImageNet [22], recurrent neural networks [23], Bayesian deep learning [24], reinforcement learning [25], and Transformer models [26].

A preliminary version of this manuscript was published previously [27]. Since then, the performance optimization of the distributed second-order optimization has been studied [28], and our distributed NGD framework has been applied to accelerate Bayesian deep learning with the natural gradient at ImageNet scale [29]. We extend the previous work and propose *Scalable and Practical Natural Gradient Descent* (SP-NGD) framework, which includes more detailed analysis on the FIM estimation and significant improvements on the performance of the distributed NGD. Although we adopt K-FAC to approximate the FIM, we believe that our SP-NGD framework can work in combination with any kind of NGD approximation that explicitly constructs curvature matrices (e.g., KFLR [30].)

Our contributions are:

- *Extremely large mini-batch training.* We were able to show for the first time that approximated NGD can achieve similar generalization capability compared to highly optimized SGD, by training ResNet-50 on ImageNet classification as a benchmark. We converged to over 75 percent top-1 validation accuracy for large mini-batch sizes of 4,096, 8,192, 16,384, 32,768 and 65,536. We also achieved 74.9 percent with an extremely large mini-batch size of 131,072, which took only 873 steps.
- *Scalable natural gradient.* We propose a distributed NGD design using data and model hybrid parallelism that shows *superlinear* scaling up to 64 GPUs.
- *Practical natural gradient.* We propose practical NGD techniques based on analysis of the FIM estimation in large mini-batch settings. Our practical techniques make the overhead of NGD compare to SGD almost negligible. Combining these techniques with our distributed NGD, we see an ideal scaling up to 1,024 GPUs as shown in Fig. 5.
- *Training ResNet-50 on ImageNet in 5.5 minutes.* Using 1,024 NVIDIA Tesla V100, we achieve 75.4 percent top-1 accuracy with ResNet-50 for ImageNet in 5.5 minutes (1,760 steps = 45 epochs, including a validation after each epoch). The comparison is shown in Fig. 1 and Table 1.

## 2 Related Work

With respect to large-scale distributed training of deep neural networks, there have been very few studies that use second-order methods. At a smaller scale, there have been previous studies that used K-FAC to train ResNet-50 on ImageNet [22]. However, the SGD they used as reference was not showing state-of-the-art Top-1 validation accuracy (only around 70 percent), so the advantage of K-FAC over SGD that they claim was not obvious from the results. In the present work, we compare the Top-1 validation

TABLE 1
Training Time and Top-1 Single-Crop Validation Accuracy of ResNet-50 for ImageNet Reported by Related Work and This Work

| | Hardware | Software | Batch size | Optimizer | # Steps | Time/step | Time | Accuracy |
|---|---|---|---|---|---|---|---|---|
| Goyal *et al.* [3] | Tesla P100 × 256 | Caffe2 | 8,192 | SGD | 14,076 | 0.255 s | 1 hr | **76.3** % |
| You *et al.* [11] | KNL × 2048 | Intel Caffe | 32,768 | SGD | 3,519 | 0.341 s | 20 min | 75.4 % |
| Akiba *et al.* [10] | Tesla P100 × 1024 | Chainer | 32,768 | RMSprop/SGD | 3,519 | 0.255 s | 15 min | 74.9 % |
| You *et al.* [11] | KNL × 2048 | Intel Caffe | 32,768 | SGD | 2,503 | 0.335 s | 14 min | 74.9 % |
| Jia *et al.* [12] | Tesla P40 × 2048 | TensorFlow | 65,536 | SGD | 1,800 | 0.220 s | 6.6 min | 75.8 % |
| Ying *et al.* [16] | TPU v3 × 1024 | TensorFlow | 32,768 | SGD | 3,519 | **0.037 s** | 2.2 min | **76.3** % |
| Mikami *et al.* [13] | Tesla V100 × 3456 | NNL | 55,296 | SGD | 2,086 | 0.057 s | 2.0 min | 75.3 % |
| Yamazaki *et al.* [14] | Tesla V100 × 2048 | MXNet | 81,920 | SGD | 1,440 | 0.050 s | **1.2 min** | 75.1 % |
| This work | Tesla V100 × 128 | Chainer | 4,096 | SP-NGD | 10,948 | 0.178 s | 32.5 min | 74.8 % |
| | Tesla V100 × 256 | | 8,192 | | 5,434 | 0.186 s | 16.9 min | 75.3 % |
| | Tesla V100 × 512 | | 16,384 | | 2,737 | 0.149 s | 6.8 min | 75.2 % |
| | Tesla V100 × 1024 | | 32,768 | | 1,760 | 0.187 s | **5.5 min** | 75.4 % |
| | n/a | | 65,536 | | 1,173 | n/a | n/a | 75.6 % |
| | n/a | | **131,072** | | **873** | n/a | n/a | 74.9 % |

accuracy with state-of-the-art SGD methods for large mini-batches mentioned in the introduction (Table 1).

The previous studies that used K-FAC to train ResNet-50 on ImageNet [22] also were not considering large mini-batches and were only training with mini-batch size of 512 on 8 GPUs. In contrast, the present work uses mini-batch sizes up to 131,072, which is equivalent to 32 per GPU on 4,096 GPUs, and we are able to achieve a much higher Top-1 validation accuracy of 74.9 percent. Note that such large mini-batch sizes can also be achieved by accumulating the gradient over multiple iterations before updating the parameters, which can mimic the behavior of the execution on many GPUs without actually running them on many GPUs.

The previous studies using K-FAC also suffered from large overhead of the communication since they used a parameter-server approach for their TensorFlow [31] implementation of K-FAC with a single parameter-server.[1] Since the parameter server requires all workers to send the gradients and receive the latest model's parameters from the parameter server, the parameter server becomes a huge communication bottleneck especially at large scale. Our implementation uses a decentralized approach using MPI/NCCL[2] collective communications among the processes. The decentralized approach has been used in high performance computing for a long time, and is known to scale to thousands of GPUs without modification. Although, software like Horovod[3] can alleviate the problems with parameter servers by working as a TensorFlow wrapper for NCCL, a workable realization of K-FAC requires solving many engineering and modeling challenges, and our solution is the first one that succeeds on a large scale task.

## 3 Notation and Background

### 3.1 Mini-Batch Stochastic Learning

Throughout this paper, we use $E[\cdot]$ to denote the empirical expectation among the samples in the mini-batch $\{(\boldsymbol{x}, \boldsymbol{t})\}$, and compute the *cross-entropy loss* for a supervised learning as

$$\mathcal{L}(\boldsymbol{\theta}) = E[-\log p_{\boldsymbol{\theta}}(\boldsymbol{t}|\boldsymbol{x})]\,, \tag{1}$$

where $\boldsymbol{x}, \boldsymbol{t}$ are the training input and label (one-hot vector), $p_{\boldsymbol{\theta}}(\boldsymbol{t}|\boldsymbol{x})$ is the likelihood of each sample $(\boldsymbol{x}, \boldsymbol{t})$ calculated by the probabilistic model using a feed-forward deep neural network (DNN) with the parameters $\boldsymbol{\theta} \in \mathbb{R}^N$.

For the standard mini-batch stochastic gradient descent (SGD), the parameters $\boldsymbol{\theta}$ are updated based on the gradient of the loss function at the current point

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta \nabla \mathcal{L}(\boldsymbol{\theta}^{(t)})\,, \tag{2}$$

where $\eta > 0$ is the learning rate.

### 3.2 Natural Gradient Descent in Deep Learning

*Natural Gradient Descent* (NGD) [17] is an optimizer which updates the parameters using the first-order gradient of the loss function preconditioned by the *Fisher information matrix* (FIM) of the probabilistic model

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta (\boldsymbol{F} + \lambda \boldsymbol{I})^{-1} \nabla \mathcal{L}(\boldsymbol{\theta}^{(t)})\,. \tag{3}$$

The FIM $\boldsymbol{F} \in \mathbb{R}^{N \times N}$ of a DNN with the learnable parameter $\boldsymbol{\theta} \in \mathbb{R}^N$ is defined as

$$\boldsymbol{F} := \mathbb{E}_{\boldsymbol{x} \sim q}\Big[\mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}}\Big[\nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) \nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})^\top\Big]\Big]\,. \tag{4}$$

$\mathbb{E}_v[\cdot]$ is an expectation w.r.t. the random variable $v$, and $q$ is the training data distribution. To limit the step size, a *damping* value $\lambda > 0$ is added to the diagonal of $\boldsymbol{F}$ before inverting it. In the training of DNNs, the FIM may be a curvature matrix in parameter space [17], [19], [30], [32].

To realize an efficient NGD training procedure for deep neural networks, we make the following approximations to the FIM that have also been used in previous work [19]:

- *Layer-wise block-diagonal approximation.* We assume that the correlation between parameters in different layers (Fig. 2) is negligible and can be ignored. This assumption significantly reduces the computational cost of inverting $\boldsymbol{F}$ especially when $N$ is large.

[1] The current version of the TensorFlow K-FAC implementation (https://github.com/tensorflow/kfac) supports other methods for distributing computations, including a decentralized approach for TPUs.
[2] https://developer.nvidia.com/nccl
[3] https://github.com/horovod/horovod

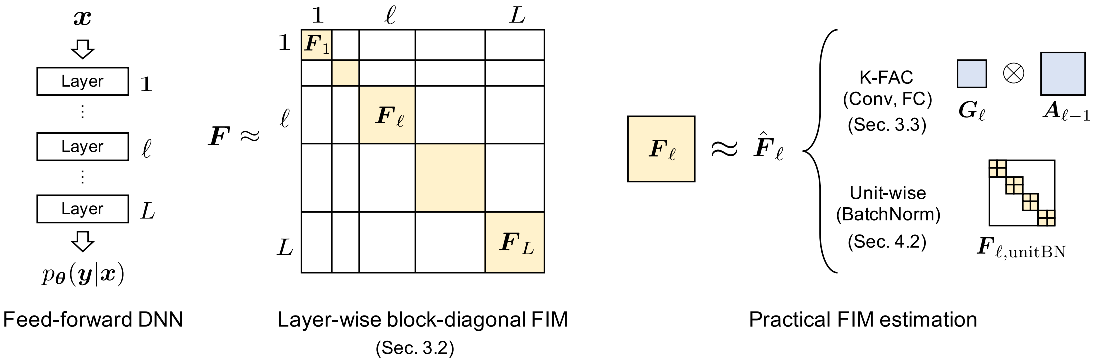
$x$
Layer 1
Layer $\ell$
Layer $L$
$p_{\theta}(y|x)$
Feed-forward DNN
$F \approx$
1 $\ell$ $L$
1 $\ell$ $L$
$F_1$
$F_\ell$
$F_L$
Layer-wise block-diagonal FIM
(Sec. 3.2)
$F_\ell \approx \hat{F}_\ell$
K-FAC (Conv, FC) (Sec. 3.3)
$G_\ell \otimes A_{\ell-1}$
Unit-wise (BatchNorm) (Sec. 4.2)
$F_{\ell,\mathrm{unitBN}}$
Practical FIM estimation

Fig. 2. Illustration of Fisher information matrix approximations for feed-forward deep neural networks used in this work.

- *Stochastic natural gradient.* We approximate the expectation over the input data distribution $\mathbb{E}_{x\sim q}[\cdot]$ using the empirical expectation over a mini-batch $E[\cdot]$. This enables estimation of $F$ during mini-batch stochastic learning.
- *Monte Carlo estimation.* We approximate the expectation over the model predictive distribution $\mathbb{E}_{y\sim p_{\theta}}[\cdot]$ using a single Monte Carlo sample (a single backward-pass). We note that for a $K$-class classification model, $K$ backward-passes are required to approximate $F$.

Using these approximations, we estimate the FIM $F_\ell \in R^{N_\ell \times N_\ell}$ for the $\ell$th layer using a Monte Carlo sample $y \sim p_{\theta}(y|x)$ for each input $x$ in a mini-batch as

$$F_\ell \approx \hat{F}_\ell := E\Big[\nabla_{w_\ell}\log\ p_{\theta}(y|x)\nabla_{w_\ell}\log p_{\theta}(y|x)^{\top}\Big]. \tag{5}$$

With this $\hat{F}_\ell$, the parameters $w_\ell \in \mathbb{R}^{N_\ell}$ for the $\ell$th layer are then updated using the FIM preconditioned gradients

$$w_\ell^{(t+1)} \leftarrow w_\ell^{(t)} - \eta\Big(\hat{F}_\ell^{(t)} + \lambda I\Big)^{-1}\nabla_{w_\ell}\mathcal{L}^{(t)}. \tag{6}$$

Here $\nabla_{w_\ell}\mathcal{L}^{(t)} \in \mathbb{R}^{N_\ell}$ denotes the gradient of the loss function w.r.t. $w_\ell$ for $w_\ell = w_\ell^{(t)}$.

### 3.3 K-FAC

Kronecker-Factored Approximate Curvature (K-FAC) [19] is a second-order optimization method for deep neural networks, that is based on an accurate and mathematically rigorous approximation of the FIM. Using K-FAC, we further approximate the FIM $F_\ell$ for $\ell$th layer as a Kronecker product of two matrices (Fig. 2)

$$F_\ell \approx \hat{F}_\ell = G_\ell \otimes A_{\ell-1}. \tag{7}$$

This is called *Kronecker factorization* and $G_\ell, A_{\ell-1}$ are called *Kronecker factors*. $G_\ell$ is computed from the gradient of the loss w.r.t. the output of the $\ell$th layer, and $A_{\ell-1}$ is computed from the activation of the $(\ell-1)$th layer (the input of $\ell$th layer).

The definition and the sizes of the Kronecker factors $G_\ell/A_{\ell-1}$ depend on the dimension of the output/input and the type of layer [19], [21], [23], [26].

#### 3.3.1 K-FAC for Fully-Connected Layers

In a fully-connected (FC) layer of a feed-forward DNN, the output $s_\ell \in \mathbb{R}^{d_\ell}$ is calculated as

$$s_\ell \leftarrow W_\ell a_{\ell-1}, \tag{8}$$

where $a_{\ell-1} \in \mathbb{R}^{d_{\ell-1}}$ is the input to this layer (the activation from the previous layer), and $W_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ is the weight matrix (the bias is ignored for simplicity). The Kronecker factors for this FC layer are defined as

$$\begin{aligned} G_\ell &:= E\Big[\nabla_{s_\ell}\log p_{\theta}(y|x)\nabla_{s_\ell}\log p_{\theta}(y|x)^{\top}\Big], \\ A_{\ell-1} &:= E\big[a_{\ell-1}a_{\ell-1}^{\top}\big], \end{aligned} \tag{9}$$

and $G_\ell \in \mathbb{R}^{d_\ell \times d_\ell}, A_{\ell-1} \in \mathbb{R}^{d_{\ell-1}\times d_{\ell-1}}$ [19]. From this definition, we can consider that K-FAC is based on an assumption that the input to the layer and the gradient w.r.t. the layer output are statistically independent.

#### 3.3.2 K-FAC for Convolutional Layers

In a convolutional (Conv) layer of a feed-forward DNN, the output $\mathcal{S}_\ell \in \mathbb{R}^{c_\ell \times h_\ell \times w_\ell}$ is calculated as

$$\begin{aligned} M_{\mathcal{A}_{\ell-1}} &\leftarrow \mathrm{im2}col(\mathcal{A}_{\ell-1}) \in \mathbb{R}^{c_{\ell-1}k_\ell^2 \times h_\ell w_\ell}, \\ M_{\mathcal{S}_\ell} &\leftarrow W_\ell M_{\mathcal{A}_{\ell-1}} \in \mathbb{R}^{c_\ell \times h_\ell w_\ell}, \\ \mathcal{S}_\ell &\leftarrow \mathrm{reshape}(M_{\mathcal{S}_\ell}) \in \mathbb{R}^{c_\ell \times h_\ell \times w_\ell}, \end{aligned} \tag{10}$$

where $\mathcal{A}_{\ell-1} \in \mathbb{R}^{c_{\ell-1}\times h_{\ell-1}\times w_{\ell-1}}$ is the input to this layer, and $W_\ell \in \mathbb{R}^{c_\ell \times c_{\ell-1}k_\ell^2}$ is the weight matrix (the bias is ignored for simplicity). $c_\ell, c_{\ell-1}$ are the number of output, input channels, respectively, and $k_\ell$ is the kernel size (assuming square kernels for simplicity). For each example in the mini-batch, the `im2col` operator[4] converts the input tensor to a matrix so that we can get the result of a convolution by a matrix multiplication. The Kronecker factors for this Conv layer are defined as

$$\begin{aligned} G_\ell &:= E\Big[\nabla_{M_{\mathcal{S}_\ell}}\log p_{\theta}(y|x)\nabla_{M_{\mathcal{S}_\ell}}\log p_{\theta}(y|x)^{\top}\Big], \\ A_{\ell-1} &:= \frac{1}{h_\ell w_\ell}E\Big[M_{\mathcal{A}_{\ell-1}}M_{\mathcal{A}_{\ell-1}}^{\top}\Big], \end{aligned} \tag{11}$$

and $G_\ell \in \mathbb{R}^{c_\ell \times c_\ell}, A_{\ell-1} \in \mathbb{R}^{c_{\ell-1}k_\ell^2 \times c_{\ell-1}k_\ell^2}$ [21].

#### 3.3.3 Inverting Kronecker-Factored FIM

By the property of the Kronecker product and the *factored Tikhonov damping technique* used in [19], the inverse of

[4] See Chainer documentation (https://docs.chainer.org/en/stable/reference/generated/chainer.functions.im2col.html) for detail.

$\hat{\boldsymbol{F}}_\ell + \lambda \boldsymbol{I}$ is approximated by the Kronecker product of the inverse of each Kronecker factor

$$\begin{aligned}\left(\hat{\boldsymbol{F}}_\ell + \lambda \boldsymbol{I}\right)^{-1} &= \left(\boldsymbol{G}_\ell \otimes \boldsymbol{A}_{\ell-1} + \lambda \boldsymbol{I}\right)^{-1} \\ &\approx \left(\boldsymbol{G}_\ell + \frac{1}{\pi_\ell}\sqrt{\lambda}\boldsymbol{I}\right)^{-1} \otimes \left(\boldsymbol{A}_{\ell-1} + \pi_\ell\sqrt{\lambda}\boldsymbol{I}\right)^{-1},\end{aligned} \tag{12}$$

where $\pi_\ell^2$ is the average eigenvalue of $\boldsymbol{A}_{\ell-1}$ divided by the average eigenvalue of $\boldsymbol{G}_\ell$ (which can be computed using the trace). $\pi > 0$ because both $\boldsymbol{G}_\ell$ and $\boldsymbol{A}_{\ell-1}$ are positive-semidefinite matrices as defined above.

## 4 Practical Natural Gradient

K-FAC for FC layer (9) and Conv layer (11) enables us to realize NGD in training deep ConvNets [21], [22]. For deep and wide neural architectures with a huge number of learnable parameters, however, due to the extra computation for the FIM, even NGD with K-FAC has considerable overhead compared to SGD. In this section, we introduce practical techniques to accelerate NGD for such huge neural architectures. Using our techniques, we are able to reduce the overhead of NGD to almost a negligible amount as shown in Section 7.

### 4.1 Fast Estimation With Empirical Fisher

Instead of using an estimation by a single Monte Carlo sampling defined in Eq. (5) ($\hat{\boldsymbol{F}}_{\ell,1mc}$), we adopt the *empirical Fisher* [19] to estimate the FIM $\boldsymbol{F}_\ell$

$$\boldsymbol{F}_\ell \approx \hat{\boldsymbol{F}}_{\ell,emp} := E\left[\nabla_{\boldsymbol{w}_\ell}\log p_{\boldsymbol{\theta}}(\boldsymbol{t}|\boldsymbol{x})\nabla_{\boldsymbol{w}_\ell}\log p_{\boldsymbol{\theta}}(\boldsymbol{t}|\boldsymbol{x})^\top\right]. \tag{13}$$

We implemented an efficient $\hat{\boldsymbol{F}}_{\ell,emp}$ computation in the Chainer framework [33] that allows us to compute $\hat{\boldsymbol{F}}_{\ell,emp}$ during the forward-pass and the backward-pass for the loss $\mathcal{L}(\boldsymbol{\theta})$[5] Therefore, we *do not need an extra backward-pass to compute* $\hat{\boldsymbol{F}}_{\ell,emp}$, while *an extra backward-pass is necessary for computing* $\hat{\boldsymbol{F}}_{\ell,1mc}$. This difference is critical especially for a deeper network which takes longer time for a backward-pass.

Although it is insisted that $\hat{\boldsymbol{F}}_{\ell,emp}$ is not a proper approximation of the FIM, and $\hat{\boldsymbol{F}}_{\ell,1mc}$ is better estimation in the literature [35], [36], we do not see any difference in the convergence behavior nor the final model accuracy between NGD with $\hat{\boldsymbol{F}}_{\ell,emp}$ and that with $\hat{\boldsymbol{F}}_{\ell,1mc}$ in training deep ConvNets for ImageNet classification as shown in Section 7. This could be because $\hat{\boldsymbol{F}}_{\ell,emp}$ is effectively being used as a pre-conditioning matrix to accelerate optimization, which only requires it to capture aspects of the FIM; in particular it also leads to parameterization invariant updates as discussed in [32].

### 4.2 Practical FIM Estimation for BatchNorm Layers

It is often the case that a deep ConvNet has Conv layers that are followed by BatchNorm layers [9]. The $\ell$th BatchNorm layer after the $(\ell-1)$th Conv layer has scale $\boldsymbol{\gamma}_\ell \in \mathbb{R}^{c_{\ell-1}}$ and bias $\boldsymbol{\beta}_\ell \in \mathbb{R}^{c_{\ell-1}}$ to be applied to the normalized features. When we see these parameters as learnable parameters, we can define

$$\boldsymbol{w}_\ell := \begin{pmatrix}\gamma_{\ell,1} & \beta_{\ell,1} & \ldots & \gamma_{\ell,c_{\ell-1}} & \beta_{\ell,c_{\ell-1}}\end{pmatrix}^\top \in \mathbb{R}^{2c_{\ell-1}}, \tag{14}$$

where $\gamma_{\ell,i}$ and $\beta_{\ell,i}$ are the $i$th element of $\boldsymbol{\gamma}_\ell$ and $\boldsymbol{\beta}_\ell$, respectively ($i = 1, \ldots, c_{\ell-1}$). For this BatchNorm layer, the FIM $\boldsymbol{F}_\ell$ (5) is a $2c_{\ell-1} \times 2c_{\ell-1}$ matrix, and the computation cost of inverting this matrix can not be ignored when $c_{\ell-1}$ (the number of output channels of the previous Conv layer) is large (e.g., $c_{out} = 1024$ for a Conv layer in ResNet-50 [37]).

#### 4.2.1 Unit-Wise Natural Gradient

We approximate this FIM by applying unit-wise natural gradient [38] to the learnable parameters of BatchNorm layers (Fig. 2). A "unit" in a neural network is a collection of input/output nodes connected to each other. The "unit-wise" natural gradient only takes into account the correlation of the parameters in the same node. Hence, for a BatchNorm layer, we only consider the correlation between $\gamma_{\ell_c}$ and $\beta_{\ell,c}$ of the same channel $c$

$$\begin{aligned}\boldsymbol{F}_\ell &\approx \hat{\boldsymbol{F}}_\ell \\ &= \boldsymbol{F}_{\ell,\text{unit}\,BN} \\ &:= \operatorname{diag}\left(\boldsymbol{F}_\ell^{(1)} \ldots \boldsymbol{F}_\ell^{(i)} \ldots \boldsymbol{F}_\ell^{(c_{\ell-1})}\right) \in \mathbb{R}^{2c_{\ell-1}\times 2c_{\ell-1}},\end{aligned} \tag{15}$$

where

$$\boldsymbol{F}_\ell^{(i)} = E\begin{bmatrix}\nabla_{\boldsymbol{\gamma}\ell}^{(i)^2} & \nabla_{\boldsymbol{\gamma}_\ell}^{(i)}\nabla_{\boldsymbol{\beta}_\ell}^{(i)} \\ \nabla_{\boldsymbol{\beta}_\ell}^{(i)}\nabla_{\boldsymbol{\gamma}_\ell}^{(i)} & \nabla_{\boldsymbol{\gamma}\ell}^{(i)^2}\end{bmatrix} \in \mathbb{R}^{2\times 2}. \tag{16}$$

$\nabla_{\boldsymbol{\gamma}_\ell}^{(i)}$, $\nabla_{\boldsymbol{\beta}_\ell}^{(i)}$ are the $i$th element of $\nabla_{\boldsymbol{\gamma}_\ell}\log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})$, $\nabla_{\boldsymbol{\beta}_\ell}\log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})$, respectively. The number of the elements to be computed and communicated is significantly reduced from $4c_{\ell-1}^2$ to $4c_{\ell-1}$, and we can get the inverse $\left(\boldsymbol{F}_{\ell,\text{unit}\,BN} + \lambda\boldsymbol{I}\right)^{-1}$ with little computation cost using the inverse matrix formula

$$\begin{bmatrix}a & b \\ c & d\end{bmatrix}^{-1} = \frac{1}{ad-bc}\begin{bmatrix}d & -b \\ -c & a\end{bmatrix}. \tag{17}$$

We observed that the unit-wise approximation on $\boldsymbol{F}_\ell$ of BatchNorm does not affect the accuracy of a deep ConvNet on ImageNet classification as shown in Section 7.

### 4.3 Natural Gradient With Stale Statistics

To achieve even faster training with NGD, it is critical to utilize stale statistics in order to avoid re-computing the matrices $\boldsymbol{A}_{\ell-1}, \boldsymbol{G}_\ell$ and $\boldsymbol{F}_{\ell,\text{unit}BN}$ (Fig. 2) at every step. Previous work on K-FAC [19] used a simple strategy where they refresh the Kronecker factors only once in 20 steps. However, as observed in [27], the statistics rapidly fluctuate at the beginning of training, and this simple strategy causes serious defects to the convergence. It was also observed that the degree of fluctuation of the statistics depends on the mini-batch size, the layer, and the type of the statistics (e.g., statistics with a larger mini-batch fluctuates less than that with a smaller mini-batch). Although the previous strategy [27] to reduce the frequency worked without any degradation of the accuracy, it requires the prior observation on the fluctuation of the statistics, and its effectiveness on training time has not been well studied.

[5] The same empirical Fisher computation can be implemented on PyTorch [34].

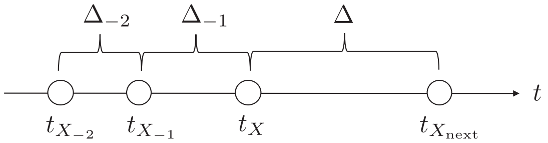

Fig. 3. Estimate the interval (steps) $\Delta$ until the next step $t_{X_{next}}$ to refresh the statistics $X$.

### 4.3.1 Adaptive Frequency to Refresh Statistics

We propose an improved strategy which adaptively determines the frequency to refresh each statistics based on its staleness during the training. Our strategy is shown in Algorithm 1. We calculate the timing (step) to refresh each statistics based on the *acceptable* interval (steps) estimated in Algorithm 2 (Fig. 3). In Algorithm 2, matrix $A$ is considered to be *similar to* matrix $B$ when $\|A - B\|_F / \|B\|_F < \alpha$, where $\| \cdot \|_F$ is Frobenius norm, and $\alpha > 0$ is the threshold.[6] We tuned $\alpha$ by running training for a few epochs to check if the threshold is not too large (preserves the same convergence). And it aims to find the *acceptable* interval $\Delta$ where the statistics $X$ calculated at step $t = t_X + \Delta$ is *similar to* that calculated at step $t = t_X$. With this strategy, we can estimate the *acceptable* interval for each statistics and skip the computation efficiently — it keeps almost the same training time as the original, while reducing the cost for constructing and inverting $\boldsymbol{A}_{\ell-1}, \boldsymbol{G}_\ell$ and $\boldsymbol{F}_{\ell,\mathrm{unit}BN}$ as much as possible. Note that $\mathcal{S}$ in Algorithm 1 is the set of the inverses of damped matrices $\boldsymbol{A}_{\ell-1}, \boldsymbol{G}_\ell$ and $\boldsymbol{F}_{\ell,\mathrm{unit}BN}$. That is, we observe the staleness of the inverse matrices. This significantly reduces the overhead of NGD. We observe the effectiveness of our approach in Section 7.[7]

**Algorithm 1.** Natural Gradient With the Stale Statistics

```
input : set of the statistics $\mathcal{S}$ (damped inverses)
input : initial parameters $\boldsymbol{\theta}$
output : parameters $\boldsymbol{\theta}$
$t \leftarrow 1$
foreach $X \in \mathcal{S}$ do
    $t_X \leftarrow 1$
end
while not converge do
    foreach $X \in \mathcal{S}$ do
        if $t == t_X$ then
            refresh statistics $X$
            $\Delta, \Delta_{-1} \leftarrow Algorithm\ 2(X, X_{-1}, X_{-2}, \Delta, \Delta_{-1})$
            $t_X \leftarrow t_X + \Delta$
            $X_{-1}, X_{-2} \leftarrow X, X_{-1}$
        end
    update $\boldsymbol{\theta}$ by natural gradient (6) using $\mathcal{S}$
    $t \leftarrow t + 1$
end
return $\boldsymbol{\theta}$
```

## 5 Scalable Natural Gradient

Based on the practical estimation of the FIM proposed in the previous section, we designed a distributed parallelization scheme among multiple GPUs so that the overhead of NGD compare to SGD decreases as the number of GPUs (processes) is increased.

**Algorithm 2.** Estimate the *acceptable* Interval Until Next Refresh Based on the Staleness of Statistics

```
input : statistics $X$, $X_{-1}$ (last), $X_{-2}$ (before the last)
input : last interval $\Delta_{-1}$, interval before the last $\Delta_{-2}$
output : next interval $\Delta$, last interval $\Delta_{-1}$
if $X$ is not similar to $X_{-1}$ then
    $\Delta \leftarrow \max\{1, \lfloor \Delta_{-1}/2 \rfloor\}$
else if $X$ is not similar to $X_{-2}$ then
    $\Delta \leftarrow \Delta_{-1}$
else
    $\Delta \leftarrow \Delta_{-1} + \Delta_{-2}$
end
return $\Delta, \Delta_{-1}$
```

### 5.1 Distributed Natural Gradient

Fig. 4 shows the overview of our design, which shows a single step of training with our distributed NGD. We use the term `Stage` to refer to each phase of computation and communication, which is indicated at the top of the figure. Algorithm 3 shows the pseudo code of our distributed NGD design.

**Algorithm 3.** Distributed Natural Gradient

```
while not converge do
    // Stage 1
    foreach $\ell = 1, \ldots, L$ do
        forward in $\ell$th layer
        if $\ell$th layer is Conv or FC then
            compute $\boldsymbol{A}_{\ell-1}$
    end
    // Stage 2
    ReduceScatterV$(\boldsymbol{A}_{0:L-1})$
    foreach $\ell = L, \ldots, 1$ do
        backward in $\ell$th layer
        if $\ell$th layer is Conv or FC then
            compute $\boldsymbol{G}_\ell$
        else if $\ell$th layer is BatchNorm then
            compute $\boldsymbol{F}_{\ell,\mathrm{unit}BN}$
    end
    // Stage 3
    ReduceScatterV$\big(\boldsymbol{G}_{1:L}/\boldsymbol{F}_{1:L,\mathrm{unit}BN}\,\mathrm{and}\nabla_{\boldsymbol{w}_{1:L}}\mathcal{L}\big)$
    // Stage 4
    for $\ell = 1, \ldots, L$ do in parallel
        update $\boldsymbol{w}_\ell$ by natural gradient (6)
    end
    // Stage 5
    AllGatherV$(\boldsymbol{w}_{1:L})$
end
return $\boldsymbol{\theta} = \big(\boldsymbol{w}_1^\top, \ldots, \boldsymbol{w}_L^\top\big)^\top$
```

In `Stage 1`, each process (GPU) receives a different part of the mini-batch and performs a forward-pass in which the Kronecker factor $\boldsymbol{A}_{\ell-1}$ is computed for the received samples, if $\ell$th layer is a Conv layer or a FC layer.

In `Stage 2`, two procedures are performed in parallel — communication among all the processes and a backward-

[6] $\alpha = 0.1$ for all the experiments in this work.

[7] For each mini-batch size, we do not use the decayed average of the FIM (Kronecker factors) as opposed to the previous K-FAC papers [19], [21], [22]. When training with extremely large batches, we observed that the decayed average has little effect.

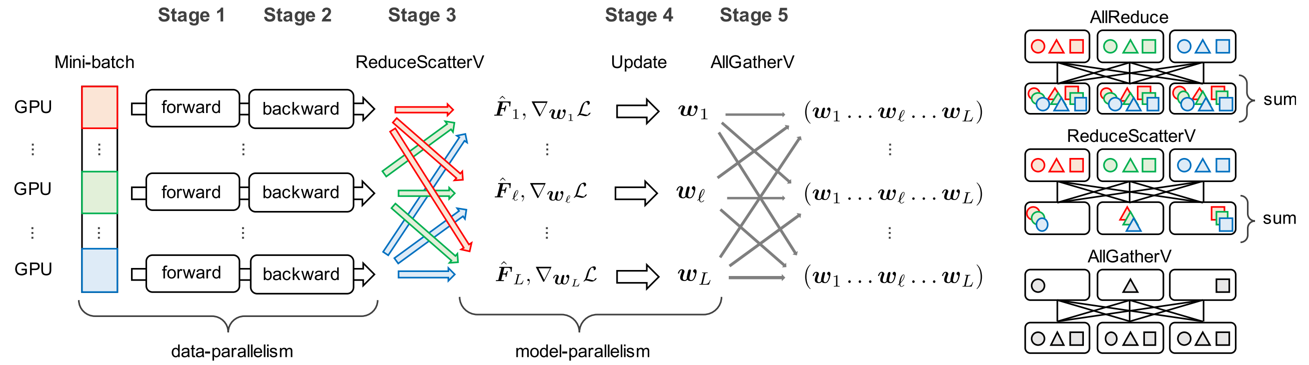

Fig. 4. (Left) Overview of our distributed natural gradient descent (a single step of training). (Right) Illustrations of `AllReduce`, `ReduceScatterV`, and `AllGatherV` collective. Different colors correspond to data (and its communication) from different data sources.

pass in each process. Since `Stage 1` is done in a *data-parallel* fashion, each process computes the statistics only for the different parts of the mini-batch. In order to compute these statistics for the entire mini-batch, we need to average these statistics over all the processes. This is performed using a `ReduceScatterV` collective communication, which transitions our approach from data-parallelism to model-parallelism by reducing (taking the sum of) $A_{\ell-1}$ for different $\ell$ to different processes. This collective is much more efficient than an `AllReduce` , where $A_{\ell-1}$ for all $\ell$ are reduced to all the processes (Fig. 4). While $A_{\ell-1}$ is communicated, each process also performs a backward-pass to get the gradient $\nabla_{w_\ell}\mathcal{L}$, the Kronecker factor $G_\ell$ for Conv, FC layers, and $F_{\ell,\mathrm{unit}BN}$ for BatchNorm layer, for each $\ell$.

In `Stage 3`, $G_\ell, F_{\ell,\mathrm{unit}BN}$ and $\nabla_{w_\ell}\mathcal{L}$ are communicated in the same way as $A_\ell$ by `ReduceScatterV` collective. At this point, only a single process has the FIM estimation $\hat{F}_\ell$ and the gradient $\nabla_{w_\ell}\mathcal{L}$ with the statistics for the entire mini-batch for the $\ell$th layer. In `Stage 4`, only the process that has the FIM computes the matrix inverse and applies the NGD update (6) to the weights $w_\ell$ of the $\ell$th layer. Hence, these computations are performed in a *model-parallel* fashion. When the number of layers is larger than the number of processes, multiple layers are handled by a process.

Once the weights $w_\ell$ of each $\ell$ are updated, we synchronize the updated weights among all the processes by calling an `AllGatherV` (Fig. 4) collective, and we switch back to data-parallelism. Combining the practical estimation of the FIM proposed in the previous section, we are able to reduce a significant amount of communication required for the Kronecker factors $A_{\ell-1}, G_\ell$ and $F_{\ell,\mathrm{unit}BN}$. Therefore, the amount of communication for our distirbuted NGD is similar to distributed SGD, where the `AllReduce` for the gradient $\nabla_{w_\ell}\mathcal{L}$ is implemented as a `ReduceScatter+AllGather`.

### 5.2 Further Acceleration

Our data-parallel and model-parallel hybrid approach allows us to minimize the overhead of NGD in a distributed setting. However, NGD still has a large overhead compared to SGD. There are two hotspots in our distributed NGD design. The first is the construction of the statistics $A_{\ell-1}, G_\ell$, and $F_{\ell,\mathrm{unit}BN}$, that cannot be done in a model-parallel fashion. The second is the communication (`ReduceScatterV`) for distributing these statistics. In this section, we discuss how we accelerate these two hotspots to achieve even faster training time.

*Mixed-Precision Computation.* We use the Tensor Cores in the NVIDIA Volta Architecture.[8] This more than doubles the speed of the calculation for this part. One might think that this low-precision computation affects the overall accuracy of the training, but in our experiments we do not find any differences between training with half-precision floating point computation and that with full-precision floating point computation.

*Symmetry-Aware Communication.* The statistics matrices $A_{\ell-1}, G_\ell$, and $F_{\ell,\mathrm{unit}BN}$ are symmetric matrices. We exploit this property to reduce the amount of communication without loss of information. To communicate a symmetric matrix of size $N \times N$, we only need to send the upper triangular matrix with $N(N+1)/2$ elements.

In addition to these two optimizations, we also adopted the performance optimizations done by [28]:

- Explicitly use NHWC (mini-batch, height, width, and channels) format for the input/output data (tensor) of Conv layers instead of NCHW format. This makes cuDNN API to fully benefit from the Tensor Cores.
- Data I/O pipeline using the NVIDIA Data Loading Library (DALI).[9]
- Hierarchical `AllReduce` collective proposed by Ueno *et al.* [39], which alleviates the latency of the `ring-AllReduce` communication among a large number of GPUs.
- Half-precision communication for `AllGatherV` collective.

## 6 Training for ImageNet Classification

The behavior of NGD on large models and datasets has not been studied in depth. Also, there are very few studies that use NGD (K-FAC) for large mini-batches (over 4 K) using distributed parallelism at scale [22]. Contrary to SGD, where the hyperparameters have been optimized by many practitioners even for large mini-batches, there is very little insight on how to tune hyperparameters for NGD. In this section, we have explored some methods, which we call training schemes, to achieve higher accuracy in our experiments. In this section, we show those training schemes in our large mini-batch training with NGD for ImageNet classification.

[8] https://www.nvidia.com/en-us/data-center/tensorcore/
[9] https://developer.nvidia.com/DALI

### 6.1 Data Augmentation

To achieve good generalization performance while keeping the benefit of the fast convergence that comes from NGD, we adopt the data augmentation techniques commonly used for training networks with large mini-batch sizes. We resize all the images in ImageNet to $256 \times 256$ ignoring the aspect ratio of original images and compute the mean value of the upper left portion ($224 \times 224$) of the resized images. When reading an image, we randomly crop a $224 \times 224$ image from it, randomly flip it horizontally, subtract the mean value, and scale every pixel to $[0, 1]$.

*Running Mixup.* We extend *mixup* [18] to increase its regularization effect. We synthesize virtual training samples from raw samples and virtual samples from the previous step (while the original *mixup* method synthesizes new samples only from the raw samples)

$$\tilde{\boldsymbol{x}}^{(t)} = \lambda \cdot \boldsymbol{x}^{(t)} + (1-\lambda) \cdot \tilde{\boldsymbol{x}}^{(t-1)} \,, \tag{18}$$

$$\tilde{\boldsymbol{t}}^{(t)} = \lambda \cdot \boldsymbol{t}^{(t)} + (1-\lambda) \cdot \tilde{\boldsymbol{t}}^{(t-1)} \,, \tag{19}$$

$\boldsymbol{x}^{(t)}, \boldsymbol{t}^{(t)}$ is a raw input and label (one-hot vector), and $\tilde{\boldsymbol{x}}^{(t)}, \tilde{\boldsymbol{t}}^{(t)}$ is a virtual input and label for $t$ th step. $\lambda$ is sampled from the Beta distribution with the beta function

$$B(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}\,\mathrm{d}t \,, \tag{20}$$

where we set $\alpha = \beta = \alpha_{\mathrm{mixup}}$.

*Random Erasing With Zero Value.* We also implemented *Random Erasing* [40]. We set elements within the erasing region of each input to zero instead of a random value as used in the original method. We set the erasing probability $p = 0.5$, the erasing area ratio $S_e \in [0.02, 0.25]$, and the erasing aspect ratio $r_e \in [0.3, 1]$. We randomly switch the size of the erasing area from $(H_e, W_e)$ to $(W_e, H_e)$.

### 6.2 Learning Rate and Momentum

The learning rate used for all of our experiments is scheduled by *polynomial decay*. The learning rate $\eta^{(e)}$ for $e$ th epoch is determined as follows:

$$\eta^{(e)} = \eta^{(0)} \cdot \left(1 - \frac{e - e_{start}}{e_{\mathrm{end}} - e_{start}}\right)^{p_{\mathrm{decay}}} \,, \tag{21}$$

$\eta^{(0)}$ is the initial learning rate and $e_{start}, e_{\mathrm{end}}$ is the epoch when the decay starts and ends. The decay rate $p_{\mathrm{decay}}$ guides the speed of the learning rate decay.

We use the momentum variant [3] for NGD updates. Because the learning rate decays rapidly in the final stage of the training with the polynomial decay, the current update can become smaller than the previous update. We adjust the momentum rate $m^{(e)}$ for $e$ th epoch so that the ratio between $m^{(e)}$ and $\eta^{(e)}$ is fixed throughout the training

$$m^{(e)} = \frac{m^{(0)}}{\eta^{(0)}} \cdot \eta^{(e)} \,, \tag{22}$$

where $m^{(0)}$ is the initial momentum rate. The weights are updated as follows:

$$\boldsymbol{w}_\ell^{(t+1)} \leftarrow \boldsymbol{w}_\ell^{(t)} - \eta^{(e)} \left(\hat{\boldsymbol{F}}_\ell^{(t)} + \lambda \boldsymbol{I}\right)^{-1} \nabla_{\boldsymbol{w}_\ell} \mathcal{L}^{(t)} + m^{(e)} \boldsymbol{v}^{(t)} \,, \tag{23}$$

where $\boldsymbol{v}^{(t)} = \boldsymbol{w}_\ell^{(t)} - \boldsymbol{w}_\ell^{(t-1)}$.

### 6.3 Weights Rescaling

We found that the choice of learning rate is very sensitive to the ratio of the (approximate) natural gradient norm to the weight norm. Although it is argued that weight decay regularization helps to control the effective damping value for NGD [41], tuning the learning rate with weight decay is difficult because it *indirectly* controls the weight norm. To alleviate this difficulty, we adopt the *Normalizing Weights* [42] technique, which *directly* controls the weight norm, to the $\boldsymbol{w}_\ell$ of FC and Conv layers. We rescale the $\boldsymbol{w}_\ell$ to have a norm $\sqrt{2 \cdot d_{out}}$ after (23)

$$\boldsymbol{w}_\ell^{(t+1)} \leftarrow \sqrt{2 \cdot d_{\mathrm{out}}} \cdot \frac{\boldsymbol{w}_\ell^{(t+1)}}{\|\boldsymbol{w}_\ell^{(t+1)}\| + \epsilon} \,, \tag{24}$$

where we use $\epsilon = 1 \cdot 10^{-9}$ to stabilize the computation. $d_{out}$ is the output dimension or channels of the layer.

## 7 Experiments

We train ResNet-50 [37] for ImageNet [4] in all of our experiments. We use the same hyperparameters for the same mini-batch size. The hyperparameters for our results are shown in Table 2. We implement all of our methods using Chainer [33]. Our Chainer extenstion is available at https://github.com/tyohei/chainerkfac. We initialize the weights by the `HeNormal` initializer of Chainer [10] with the default parameters.

### 7.1 Experiment Environment

We conduct all experiments on the AI Bridging Cloud Infrastructure (ABCI)[11] operated by the National Institute of Advanced Industrial Science and Technology (AIST) in Japan. ABCI has 1,088 nodes with four NVIDIA Tesla V100 GPUs per node. Due to the additional memory required by NGD, all of our experiments use a mini-batch size of 32 per GPU. We were only given a 24 hour window to use the full machine so we had to tune the hyperparameters on a smaller number of nodes while mimicking the global mini-batch size of the full node run. For large mini-batch size experiments which cannot be executed directly (BS=65 K, 131 K requires 2048, 4096 GPUs, respectively) ,we used an accumulation method to mimic the behavior by accumulating the statistics $\boldsymbol{A}_{\ell-1}, \boldsymbol{G}_\ell, \boldsymbol{F}_{\ell,\mathrm{unit}BN}$, and $\nabla_{\boldsymbol{w}_\ell}\mathcal{L}$ over multiple steps.

### 7.2 Extremely Large Mini-Batch Training

We trained ResNet-50 for ImageNet classification task with extremely large mini-batch size BS={4,096, 8,192, 16,384, 32,768, 65,536, 131,072} and achieved a competitive top-1 validation accuracy ($\geq 74.9\%$) compared to highly-tuned SGD training. The summary of the training is shown in Table 1. For BS={4K, 8K, 16K, 32K, 65K}, the training converges in much less than 90 epochs, which is the usual number of epochs required by SGD-based training of ImageNet [3], [10], [11], [12], [13]. For BS={4K,8K,16K}, the required epochs to reach higher than 75 percent top-1 validation

[10] https://docs.chainer.org/en/stable/reference/generated/chainer.initializers.HeNormal.html

[11] https://abci.ai/

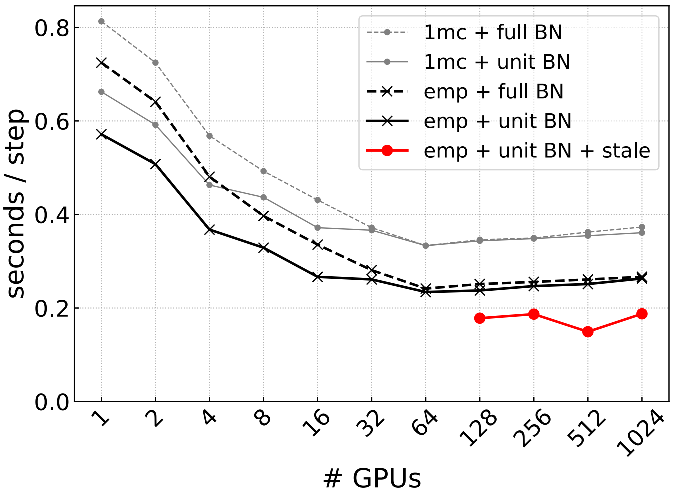

Fig. 5. Time per step for trainig ResNet-50 (107 layers in total) on ImageNet with our scalable and practical NGD. Each GPU processes 32 images. `1mc` and `emp` correspond to NGD with $\hat{F}_{\ell,1mc}$ and that with $\hat{F}_{\ell,emp}$, respectively. `fullBN` and `unitBN` correspond to NGD and unit-wise NGD on BatchNorm parameters, respectively. `stale` corresponds to NGD with the stale statistics.

accuracy is 35 epochs. Even for a relatively large mini-batch size of BS=32K, NGD still converges in 45 epochs, half the number of epochs compared to SGD. Note that the calculation time is still decreasing while the number of epochs is less than double when we double the mini-batch size, assuming that doubling the mini-batch corresponds to doubling the number of GPUs (and halving the execution time). When we increase the mini-batch size to BS={32K,65K,131K}, we see a significantly small number of steps it takes to converge. At BS=32 K and 65 K, it takes 1,760 steps (45 epochs) and 1,173 steps (60 epochs), respectively. At BS=131 K, there are less than 10 iterations per epoch since the dataset size of ImageNet is 1,281,167, and it only takes 873 steps to converge (90 epochs). None of the SGD-based training of ImageNet have sustained the top-1 validation accuracy at this mini-batch size. Furthermore, this is the first work that uses NGD for the training with extremely large mini-batch size BS={16K,32K,65K,131K} and achieves a competitive top-1 validation accuracy.

### 7.3 Scalability

We measure the scalability of our distributed NGD implementation for training ResNet-50 on ImageNet. Fig. 5 shows the time for one step with different number of GPUs and different techniques proposed in Section 7.4. Note that since we fix the number of images to be processed per GPU (=32), doubling the number of GPUs means doubling the total number of images (mini-batch size) to be processed in a step (e.g., 32 K images are processed with 1,024 GPUs in a step). In a distributed training with multiple GPUs, it is considered ideal if this plot shows a flat line parallel to the $x$-axis, that is, the time per step is independent of the number of GPUs, and the number of images processed in a certain time increases linearly with the number of GPUs. From 1 GPU to 64 GPUs, however, we observed a *superlinear* scaling. For example, the time per step with 64 GPUs is 300 percent faster than that with 1 GPU for `emp+fullBN`. This is the consequence of our model-parallel design since ResNet-50 has 107 layers in total when all the Conv, FC, and BatchNorm layers are accounted for. With more than 128 GPUs, we observe slight performance degradation due to the communication overhead comes from `ReduceScatterV` and `AllGatherV` collective. Yet for `emp+unitBN+stale`, we see almost the ideal scaling from 128 GPUs to 1,024 GPUs. Moreover, with 512 GPUs, which corresponds to BS=16 K, we see a *superlinear* scaling, again. We discuss this in the next sub-section.

### 7.4 Effectiveness of Practical Natural Gradient

We examine the effectiveness of our practical NGD approaches proposed in Section 7.4 for training ResNet-50 on ImageNet with extreamely large mini-batch. We show that our practical techniques makes the overhead of NGD close to a negligible amount and improves training time significantly. The summary of the training time is shown in Table 1 and Fig. 1.

*Natural Gradient by Empirical Fisher.* We compare the time and model accuracy in training by NGD with empirical Fisher and that with a Fisher estimation by a single Monte Carlo sampling ($\hat{F}_{\ell,emp}$ versus $\hat{F}_{\ell,1mc}$) . In Fig. 5, the time per a step by each training is labeled as `emp` and `1mc`, respectively. Due to the extra backward-pass required for constructing $\hat{F}_{\ell,1mc}$, `1mc` is slower than `emp` at any number of GPUs. We do not see any difference in the convergence behavior (accuracy versus steps) and the final accuracy for training ResNet-50 on ImageNet with BS={4K,8K,16K,32K,65K,131K}. Note that we used the same hyperparameters tuned for `emp` for each BS (shown in Table 2) for the limitation of computational resource to tune for `1mc`.

*Unit-Wise Natural Gradient.* We also compare training with natural gradient and that with unit-wise natural gradient on

TABLE 2
The Hyperparameters of the Training With Large Mini-Batch Size (BS) Used for Our Schemes in Section 7.2 and Top-1 Single-Crop Validation Accuracy of ResNet-50 for ImageNet

| BS | $\alpha_{mixup}$ | $p_{decay}$ | $e_{start}$ | $e_{end}$ | $\eta^{(0)}$ | $m^{(0)}$ | $\lambda$ | # steps | top-1 accuracy | reduction ↓ | speedup ↑ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4K | 0.4 | 11.0 | 1 | 53 | $8.18 \cdot 10^{-3}$ | 0.997 | $2.5 \cdot 10^{-4}$ | 10,948 | 75.2 % → 74.8 % | 23.6 % | × 1.33 |
| 8K | 0.4 | 8.0 | 1 | 53.5 | $1.25 \cdot 10^{-2}$ | 0.993 | $2.5 \cdot 10^{-4}$ | 5,434 | 75.5 % → 75.3 % | 15.1 % | × 1.32 |
| 16K | 0.4 | 8.0 | 1 | 53.5 | $2.5 \cdot 10^{-2}$ | 0.985 | $2.5 \cdot 10^{-4}$ | 2,737 | 75.3 % → 75.2 % | **5.4** % | × **1.68** |
| 32K | 0.6 | 3.5 | 1.5 | 49.5 | $3.0 \cdot 10^{-2}$ | 0.97 | $2.0 \cdot 10^{-4}$ | 1,760 | 75.6 % → 75.4 % | 7.8 % | × 1.40 |
| 65K | 0.6 | 2.9 | 2 | 64.5 | $4.0 \cdot 10^{-2}$ | 0.95 | $1.5 \cdot 10^{-4}$ | 1,173 | 75.6 % | n/a | n/a |
| 131K | 1.0 | 2.9 | 3 | 100 | $7.0 \cdot 10^{-2}$ | 0.93 | $1.0 \cdot 10^{-4}$ | 873 | 74.9 % | n/a | n/a |

*`reduction` and `speedup` correspond to the reduction rate of the communication amount and the speedup comes from that, respectively, for `emp+unitBN+stale` compare to `emp+unitBN` in Fig. 5.*

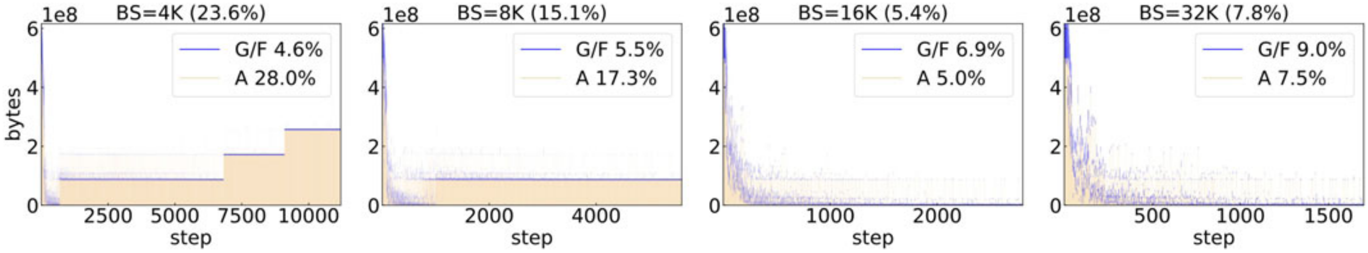

Fig. 6. The communication amount (bytes) for the statistics $(A_{\ell-1}, G_\ell, F_{\ell,\mathrm{unit}BN})$ in each step in training ResNet-50 on ImageNet with BS= {4K,8K,16K,32K} (stacked graph — the amount for `G/F` is stacked on the amount for `A`). `A` and `G/F` correspond to the communication amount for $A_{\ell-1}$ and $G_\ell/F_{\ell,\mathrm{unit}BN}$, respectively. The reduction rate (smaller is better) of the communication amount for all the statistics throughout the training is shown with the percentage (%).

BatchNorm parameters ($F_\ell$ versus $F_{\ell,\mathrm{unit}BN}$) . In Fig. 5, the time per step for each method is labeled as `fullBN` and `unitBN`, respectively. From 1 GPU to 16 GPUs, we can see that `unitBN` effectively accelerates the time per step compare to `fullBN`. For more than 32 GPUs, we can see only a slight improvements since inverting statistics ($A_{\ell-1}, G_\ell$ and $F_\ell$) for all the layers are already distributed among enough number of processes, and inverting $F_\ell$ is no longer a bottleneck of processing a step. We do not see any difference in the convergence behavior (accuracy versus steps) and the final accuracy for training ResNet-50 on ImageNet with BS= {4K,8K,16K,32K,65K,131K}.

*Natural Gradient With Stale Statistics.* We apply our adaptive frequency strategy described in Section 7.4 to training ResNet-50 on ImageNet with BS={4K,8K,16K,32K}. In Fig. 5, the time per a step with all the practical techniques is labeled as `emp+unitBN+stale`. The model accuracy before and after applying this technique, reduction rate (smaller is better) of the communication volume for the statistics, and speedup (`emp+unitBN` versus `emp+unitBN+stale`) is shown in Table 2. The communication amount (bytes) in the `ReduceScatterV` collective in each step during a training and the reduction rate are plotted in Fig. 6. With BS=16K,32K, we can reduce the communication amount for the statistics ($A_{\ell-1}, G_\ell$ and $F_\ell$) to 5.4,7.8 percent, respectively. We might be able to attribute this significant reduction rate to the fact that the statistics with larger BS (16K,32K) is more stable than that with smaller BS (4K,8K). Note that though we show the reduction rate of the amount of communication, this rate is also applicable to estimate the reduction rate of the amount of computation for the statistics, and the cost for inverting them is also removed. With these improvements on NGD, we see almost an ideal scaling from 128 GPUs to 1,024 GPUs, which corresponds to BS=4 K to 32 K.

### 7.5 Training ResNet-50 on ImageNet With NGD in 5.5 Minutes

Finally, we combine all the practical techniques — empirical Fisher, unit-wise NGD and NGD with stale statistics. Using 1,024 NVIDIA Tesla V100, we achieve 75.4 percent top-1 accuracy with ResNet-50 for ImageNet in 5.5 minutes (1,760 steps = 45 epochs, including a validation after each epoch). We used the same hyperparameters shown in Table 2. The training time and the validation accuracy are competitive with the results reported by related work that use SGD for training (the comparison is shown in Table 1). We refer to our training method as *Scarable and Practical NGD* (SP-NGD).

## 8 Discussion and Future Work

In this work, we proposed a *Scalable and Practical Natural Gradient Descent* (SP-NGD), a framework which combines i) a large-scale distributed computational design with data and model hybrid parallelism for the Natural Gradient Descent (NGD) [17] and ii) practical Fisher information estimation techniques including Kronecker-Factored Approximate Curvature (K-FAC) [19], that alleviates the computational overhead of NGD over SGD. Using our SP-NGD framework, we showed the advantages of the NGD over first-order stochastic gradient descent (SGD) for training ResNet-50 on ImageNet classification with extremely large mini-batches. We introduced several schemes for the training using the NGD with mini-batch sizes up to 131,072 and achieved over 75 percent top-1 accuracy in much fewer number of steps compared to the existing work using the SGD with large mini-batch. By using strong data augmentation (i.e., mixup [24]), we were able to show that solutions found by a second-order method generalize as well as that of SGD, even for extremely large mini-batches. Note that this data augmentation helps second-order methods more than they do SGD, so it is not a matter of simply increasing the baseline accuracy through data augmentation. Our SP-NGD framework allowed us to train on 1,024 GPUs and achieved 75.4 percent in 5.5 minutes. This is the first work which observes the relationship between the FIM of ResNet-50 and its training on large mini-batches ranging from 4 to 131 K.

*More Accurate and Efficient FIM Estimation.* We showed that NGD using the empirical Fisher matrix [19] (`emp`) is much faster than that with an estimation using a single Monte Carlo (`1mc`), which is widely used by related work on the approximate natural gradient. Although it is stated that `emp` is not a good approximation of the NGD in the literature [35], [36], we observed the same convergence behavior as `1mc` for training ResNet-50 on ImageNet. We might be able to attribute this result to the fact that `emp` is a good enough approximation to keep the behavior of the *true* NGD or that even `1mc` is not a good approximation. To know whether these hypotheses are correct or not and to examine the actual value of the true NGD, we need a more accurate and effcient estimation of the NGD with less computational cost.

*Towards Bayesian Deep Learning.* NGD has been applied to Bayesian deep learning for estimating the posterior distribution of the network parameters. For example, K-FAC [19] has been applied to Bayesian deep learning to realize *Noisy Natural Gradient* [24], and our distributed NGD has been applied to that at ImageNet scale [29]. We similarly expect that our SP-NGD framework will accelerate Bayesian deep learning research using natural gradient methods.

## Acknowledgments

Computational resource of AI Bridging Cloud Infrastructure (ABCI) was awarded by "ABCI Grand Challenge" Program, National Institute of Advanced Industrial Science and Technology (AIST). This work was supported by JSPS KAKENHI Grant Number JP18H03248 and JP19J13477. This work was also supported by JST CREST Grant Number JPMJCR19F5, Japan. Part of this work was conducted as research activities of AIST - Tokyo Tech Real World Big-Data Computation Open Innovation Laboratory (RWBC-OIL). This work was supported by "Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures" in Japan (Project ID: jh180012-NAHI). This research used computational resources of the HPCI system provided by Tokyo Tech through the HPCI System Research Project (Project ID: hp190122). The authors would like to thank Yaroslav Bulatov (South Park Commons) for helpful comments on the manuscript.

## References

[1] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," in *Proc. Int. Conf. Learn. Representations*, 2017, Art. no. 16.

[2] E. Hoffer, R. Banner, I. Golan, and D. Soudry, "Norm matters: Efficient and accurate normalization schemes in deep networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 2160–2170.

[3] P. Goyal *et al.*, "Accurate, large minibatch SGD: Training ImageNet in 1 hour," 2017, *arXiv: 1706.02677*.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. Int. Conf. Comput. Vis.*, 2015, pp. 1026–1034.

[6] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the effects of data parallelism on neural network training," *J. Mach. Learn. Res.*, vol. 20, no. 112, pp. 1–49, 2019.

[7] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.

[8] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Netw. Mach. Learn.*, vol. 4, pp. 26–31, 2012.

[9] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[10] T. Akiba, S. Suzuki, and K. Fukuda, "Extremely large minibatch SGD: Training ResNet-50 on ImageNet in 15 minutes," 2017, *arXiv: 1711.04325*.

[11] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," 2017, *arXiv: 1708.03888*.

[12] X. Jia *et al.*, "Highly scalable deep learning training system with mixed-precision: Training ImageNet in four minutes," 2018, *arXiv: 1807.11205*.

[13] H. Mikami, H. Suganuma, P. U-chupala, Y. Tanaka, and Y. Kageyama, "Massively distributed SGD: ImageNet/ResNet-50 training in a flash," 2018, *arXiv: 1811.05233*.

[14] M. Yamazaki *et al.*, "Yet another accelerated SGD: ResNet-50 training on ImageNet in 74.7 seconds," 2019, *arXiv: 1903.12650*.

[15] T. Lin, S. U. Stich, and M. Jaggi, "Don't use large mini-batches, Use Local SGD," in *Proc. Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=B1eyO1BFPr

[16] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng, "Image classification at supercomputer scale," 2018, *arXiv: 1811.06992*.

[17] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural Comput.*, vol. 10, pp. 251–276, 1998.

[18] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," in *Proc. Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=r1Ddp1-Rb

[19] J. Martens and R. Grosse, "Optimizing neural networks with Kronecker-factored approximate curvature," in *Proc. 32nd Int. Conf. Int. Conf. Mach. Learn.*, 2015, pp. 2408–2417.

[20] J. Martens, "Deep learning via Hessian-free optimization," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010, pp. 735–742.

[21] R. Grosse and J. Martens, "A Kronecker-factored approximate fisher matrix for convolution layers," in *Proc. 33rd Int. Conf. Int. Conf. Mach. Learn.*, 2016, pp. 573–582.

[22] J. Ba, R. Grosse, and J. Martens, "Distributed second-order optimization using Kronecker-factored approximations," in *Proc. Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=SkkTMpjex

[23] J. Martens and M. Johnson, "Kronecker-factored curvature approximations for recurrent neural networks," in *Proc. Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=HyMTkQZAb

[24] G. Zhang, S. Sun, D. Duvenaud, and R. Grosse, "Noisy natural gradient as variational inference," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 5852–5861.

[25] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5279–5288.

[26] G. Zhang *et al.*, "Which algorithmic choices matter at which batch sizes? Insights from a noisy quadratic model," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8196–8207.

[27] K. Osawa, Y. Tsuji, Y. Ueno, A. Naruse, R. Yokota, and S. Matsuoka, "Large-scale distributed second-order optimization using Kronecker-factored approximate curvature for deep convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 12 359–12 367.

[28] Y. Tsuji, K. Osawa, Y. Ueno, A. Naruse, R. Yokota, and S. Matsuoka, "Performance optimizations and analysis of distributed deep learning with approximated second-order optimization method," in *Proc. 48th Int. Conf. Parallel Process. Workshops*, 2019, pp. 21:1–21:8.

[29] K. Osawa *et al.*, "Practical deep learning with Bayesian principles," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 4287–4299.

[30] A. Botev, H. Ritter, and D. Barber, "Practical Gauss-Newton optimisation for deep learning," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 557–565.

[31] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 265–283.

[32] J. Martens, "New insights and perspectives on the natural gradient method," 2014, *arXiv:1412.1193*.

[33] S. Tokui *et al.*, "Chainer: A deep learning framework for accelerating the research cycle," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 2002–2011.

[34] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.

[35] V. Thomas, F. Pedregosa, B. van Merrienboer, P.-A. Mangazol, Y. Bengio, and N. L. Roux, "Information matrices and generalization," 2019, *arXiv: 1906.07774*.

[36] F. Kunstner, L. Balles, and P. Hennig, "Limitations of the empirical fisher approximation," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 4156–4167.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[38] S.-I. Amari, R. Karakida, and M. Oizumi, "Fisher information and natural gradient learning in random deep networks," in *Proc. 22nd Int. Conf. Artif. Intell. Statist.*, 2019, pp. 694–702.

[39] Y. Ueno and R. Yokota, "Exhaustive study of hierarchical All-Reduce patterns for large messages between GPUs," in *Proc. 19th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2019, pp. 430–439.

[40] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *Proc. AAAI Conf. Art. Intl.*, 2020, pp. 13001–13008.

[41] G. Zhang, C. Wang, B. Xu, and R. Grosse, "Three mechanisms of weight decay regularization," in *Proc. Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=B1lz-3Rct7

[42] T. van Laarhoven, "L2 regularization versus batch and weight normalization," 2017, *arXiv: 1706.05350*.

**Kazuki Osawa** (Student Member, IEEE) received the BS and MS degrees from the Tokyo Institute of Technology, Tokyo, Japan, in 2016 and 2018, respectively. He is currently working toward the PhD degree at the Tokyo Institute of Technology, Tokyo, Japan and a research fellow of Japan Society for the Promotion of Science (JSPS). His research interests include optimization, approximate Bayesian inference, and distributed computing for deep learning.

**Yohei Tsuji** received the BS and MS degrees from the Tokyo Institute of Technology, Tokyo, Japan, in 2017 and 2019, respectively. He is currently working toward the PhD degree at the Tokyo Institute of Technology, Tokyo, Japan. His research interests include high performance computing for machine learning, probabilistic programming.

**Yuichiro Ueno** received the BS degree from the Tokyo Institute of Technology, Tokyo, Japan, in 2019. He is currently working toward the master's degree at the Tokyo Institute of Technology, Tokyo, Japan. His research interests include a range of high-performance computing, such as GPU computing and networking, and its application to deep learning.

**Akira Naruse** received the MS degree in computer science from Nagoya University, Nagoya, Japan. He is  currently a senior developer technology engineer with NVIDIA. Prior to joining NVIDIA, he was a research engineer with Fujitsu Laboratory and was involved in various high performance computing projects. His main interests include performance analysis and optimization of scientific computing and deep learning applications on very large systems.

**Chuan-Sheng Foo** received the BS, MS, and PhD degrees from Stanford University, Stanford, California. He is currently a scientist with the Institute for Infocomm Research, A*STAR. His research interests include developing deep learning algorithms that can learn from less labeled data, inspired by applications in healthcare and medicine.

**Rio Yokota** received the BS, MS, and PhD degrees from Keio University, Tokyo, Japan, in 2003, 2005, and 2009, respectively. He is currently an associate professor with GSIC, Tokyo Institute of Technology. His research interests include high performance computing, hierarchical low-rank approximation methods, and scalable deep learning. He was part of the team that won the Gordon Bell Prize for Price/Performance, in 2009.