SKFAC: Training Neural Networks with Faster Kronecker-Factored Approximate Curvature

Zedong Tang[1,2]*, Fenlong Jiang[1]*, Maoguo Gong[1]†, Hao Li[1], Yue Wu[3]
Fan Yu[4], Zidong Wang[4], Min Wang[4]
[1] School of Electronic and Engineering, Xidian University
[2] Academy of Advanced Interdisciplinary Research, Xidian University
[3] School of Computer Science and Technology, Xidian University
tangzedong@hotmail.com, jiangfenlong@outlook.com, gong@ieee.org
ywu@xidian.edu.cn, omegalihao@gmail.com
[4] Central Software Institute, 2012 Lab, Huawei Technologies Co. Ltd
{fan.yu, wang1, wangmin10}@huawei.com

Abstract

The bottleneck of computation burden limits the widespread use of the 2nd order optimization algorithms for training deep neural networks. In this paper, we present a computationally efficient approximation for natural gradient descent, named Swift Kronecker-Factored Approximate Curvature (SKFAC), which combines Kronecker factorization and a fast low-rank matrix inversion technique. Our research aims at both fully connected and convolutional layers. For the fully connected layers, by utilizing the low-rank property of Kronecker factors of Fisher information matrix, our method only requires inverting a small matrix to approximate the curvature with desirable accuracy. For convolutional layers, we propose a way with two strategies to save computational efforts without affecting the empirical performance by reducing across the spatial dimension or receptive fields of feature maps. Specifically, we propose two effective dimension reduction methods for this purpose: Spatial Subsampling and Reduce Sum. Experimental results of training several deep neural networks on Cifar-10 and ImageNet-1k datasets demonstrate that SKFAC can capture the main curvature and yield comparative performance to K-FAC. The proposed method bridges the wall-clock time gap between the 1st and 2nd order algorithms.

1. Introduction

Deep learning, whose success is inseparable from the support of enormous computing power, has achieved fruitful results in many fields, such as computer vision [6][21] and natural language processing [1][11]. However, for large-scale tasks, effective training of neural networks is usually time-consuming [8], resulting in a new request for fast and efficient training methods.

In general, the training of neural networks is a process to optimize the network parameters θ by minimizing the regularized empirical loss function $L(\theta)$, and the parameters are updated as: $\theta \leftarrow \theta - \eta G^{-1} \nabla_\theta L(\theta)$, where G is a positive definite preconditioner of the gradient and η represents a positive learning rate. In particular, if G is an identity matrix, it will degrade to the classical 1^{st} order optimization algorithm, namely Stochastic Gradient Descent (SGD). It and its variants with momentum term are current mainstream in neural network training. SGD adopts consistent update step size for all parameters. As for 2^{nd} order optimization, in this case, G contains more curvature or correlation information, such as the Hessian matrix in Newton's method or Fisher Information Matrix (FIM) in natural gradient method (NG). For a network with massive parameters, G has a gigantic size of $n_\theta \times n_\theta$, where n_θ is the dimensions of θ, which renders it impractical to compute and invert it directly. Only if we can get huge time savings at the expense of a little information integrity, 2^{nd} order algorithms will show more potential than the 1^{st} order ones to fasten the training.

This has motivated researches into finding the approximation of G and its inverse G^{-1} with low computational cost. For instance, the well-known and widely used Ada-

*Equal Contribution
†Corresponding Author

This work was supported in part by the National Natural Science Foundation of China under Grant 62036006 and Grant 61906146, in part by CAAI Huawei MindSpore Open Fund, and in part by the Key Research and Development Program of Shaanxi Province under Grant 2018ZDXM-GY-045.

grad [2], Adadelta [26], RMSProp [25], and Adam [9] involve a simple diagonal approximation to the covariance matrix of the gradients. Moreover, more sophisticated algorithms are not limited to diagonalization, but try to preserve more important correlations between parameters instead. Le Roux *et al.* proposed a block-diagonal approximation of empirical FIM named TONGA [23], where each block corresponds to the weights of each neuron. It computes the inverse matrix of each block by maintaining a low rank plus diagonal term. In 2013, Ollivier *et al.* also proposed a similar block-diagonal approach [17], except that it used the standard FIM instead of the empirical one[1]. In addition, some methods leverage the hierarchical structure of neural network to perform block-diagonal approximation, such as K-FAC [14], and methods in [7][19]. Interestingly, in addition to similar approximations, they [7][19][14] all Kronecker-factorized the block matrices, which can further reduce the computational cost. However, for models that have thousands or even tens of thousands of neurons per layer, it is still too time-consuming to efficiently compute the inverse matrices of those factors, which also results in the consequence—compared with carefully tuned SGD and adaptive algorithms such as Adam—that these algorithms still have no significant advantage in overall time cost.

We present a much faster approximation algorithm for natural gradient descent, namely Swift Kronecker-Factored Approximate Curvature (SKFAC). Consider this case where a large batch of samples are distributed to multiple nodes, each node only processes a mini batch whose size might be smaller than the number of layers' inputs and outputs. The covariance matrices of activations and pre-activations are induced by the low-rank factors. Inspired by this, we propose a low-rank formulation of K-FAC by reformulating the FIM, such that the computations of preconditioners have moderate time which is sublinear in the number of layers' inputs and outputs. For convolutional layers, additional spatial structures of feature maps will bring out a sharp increase in computational burden in the proposed method. Therefore, we further propose two corresponding dimensionality reduction methods for the convolutional layers. We train canonical neural networks to verify the advancement of SKFAC on the Cifar/ImageNet datasets. The results show that SKFAC can not only retain improved convergence properties, but also tremendously improve the efficiency of computing natural gradients in large-scale networks. The contributions of our work can be summarized as follows:

- We attempt to reformulate the Kronecker-factored approximation of natural gradient for accelerating the inversion.
- Two effective dimensionality reduction methods are proposed to address the issue of too large dimension in convolutional layers.
- We propose to only transmit the activations and pre-activation derivations in distributed training to reduce communication latency.
- The experiments of training neural networks demonstrate that our method can retain the good convergence property, like that of K-FAC.

[1]Please refer to [13] and [10] for detailed discussions of the difference between the standard Fisher and the empirical one.

2. Background and Notations

Our work is an improvement method based on the previous Kronecker-factored approximation methods. The relevant background and works are described below.

2.1. Natural Gradient Descent

Given a train dataset D_{train} containing pairs of inputs and target samples $(\mathbf{x}, \mathbf{y})$, training a neural network with L layers can be seen as minimizing a cost function $J(\theta)$ where $\theta = (\text{vec}(\mathbf{W}_1), \cdots, \text{vec}(\mathbf{W}_L))$ is a vector consisted of all layers' parameters in the model. $J(\theta)$ is the average of the loss L between the model's predictions and targets over D_{train}:

$$J(\theta) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\in D_{train}} \mathcal{L}\left(f(\mathbf{x};\theta), \mathbf{y}\right). \tag{1}$$

The parameter optimization process of the natural gradient descent algorithm (NG) can be expressed as

$$\theta \leftarrow \theta - \eta \mathbf{F}^{-1} \nabla_\theta J(\theta), \tag{2}$$

where η represents a learning rate and $\mathbf{F}$ is the Fisher Information Matrix (FIM), as given by

$$\mathbf{F} = \mathbb{E}_{\mathbf{x}\in D_{train}, \mathbf{y}\sim p(\hat{\mathbf{y}}|\mathbf{x};\theta)} \left[\nabla_\theta J(\theta) \nabla_\theta J(\theta)^\top\right]. \tag{3}$$

It is worth noting that the expectation is taken over the data distribution and model's predictive distribution. The FIM represents the degree of correlation between the gradients. In contrast with the standard gradient descent, the natural gradient descent can be interpreted as the steepest descent measured by KL divergence instead of the Euclidean norm.

2.2. K-FAC

In natural gradient, the size of $\mathbf{F}$ is $|\theta| \times |\theta|$. For large networks with billions of parameters, it is impractical to compute $\mathbf{F}^{-1}$ directly. In [14], Martens *et al.* proposed a Kronecker-factored approximate curvature (K-FAC) for the natural gradient descent, which subtly leverages the hierarchical structure of neural networks to perform a block diagonal approximation of $\mathbf{F}$ and greatly reduce the computational cost.

Considering a neural network with L layers, we use $\mathbf{A}$ to represent the activations of each layer and $\mathcal{D}\mathbf{S}$ to represent

the pre-activation gradient of each layer obtained through backward propagation. The Kronecker factors of $\mathbf{F}$ are denoted as $\mathbf{\Omega}$ and $\mathbf{\Gamma}$, respectively. Taking the fashion of mini-batch stochastic training, we use $\mathcal{B}$ to represent a mini-batch of size M and $\mathbb{E}[\cdot]$ to represent the expectation over the model's predictive distribution.

2.2.1 K-FAC for Fully Connected Layers

We first discuss the situation of the fully connected layer. According to the idea of dividing FIM in blocks by layers [14], each block of FIM can be factorized as

$$\mathbf{F}_{i,j} = \mathbb{E}[\mathbf{A}_{i-1}^{\top}\mathbf{A}_{j-1} \otimes \mathcal{D}\mathbf{S}_i^{\top}\mathcal{D}\mathbf{S}_j], \tag{4}$$

where $i, j = 1, 2, \ldots, L$. $\mathbf{u} \otimes \mathbf{v}$ denotes the Kronecker product of arbitrary $\mathbf{u}$ and $\mathbf{v}$.

First, K-FAC approximates the expectation of a Kronecker product as the Kronecker product of expectations, as given by:

$$\mathbf{F}_{i,j} \approx \mathbf{\Omega}_{i-1,j-1} \otimes \mathbf{\Gamma}_{i,j}, \tag{5}$$

where $\mathbf{\Omega}_{i-1,j-1} = \mathbb{E}[\mathbf{A}_{i-1}^{\top}\mathbf{A}_{j-1}]$ and $\mathbf{\Gamma}_{i,j} = \mathbb{E}[\mathcal{D}\mathbf{S}_i^{\top}\mathcal{D}\mathbf{S}_j]$.

Then, K-FAC designs two approximate forms, block diagonal and block tridiagonal. Here, we use the relatively simple block diagonal approximation for computational efficiency. Only the FIM blocks located on the diagonal are reserved. In this case, $\mathbf{F}$ and $\mathbf{F}^{-1}$ can be represented as:

$$\mathbf{F} \approx diag\left(\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_{\ell}, \cdots, \mathbf{F}_L\right), \tag{6}$$

$$\mathbf{F}^{-1} \approx diag\left(\mathbf{F}_1^{-1}, \mathbf{F}_2^{-1}, \ldots, \mathbf{F}_{\ell}^{-1}, \cdots, \mathbf{F}_L^{-1}\right), \tag{7}$$

where $\mathbf{F}_{\ell} = \mathbf{\Omega}_{\ell-1} \otimes \mathbf{\Gamma}_{\ell}$ and $F_{\ell}^{-1} = \mathbf{\Omega}_{\ell-1}^{-1} \otimes \mathbf{\Gamma}_{\ell}^{-1}$. Computing the inverse factors $\mathbf{\Omega}_{\ell-1}^{-1}$ and $\mathbf{\Gamma}_{\ell}^{-1}$ is much faster than directly computing $\mathbf{F}^{-1}$.

Here, we detail the size of involved factors:

$$\begin{array}{ll} \mathbf{A}_{\ell-1} \in \mathbb{R}^{M \times n_{\ell}}, & \mathcal{D}\mathbf{S}_{\ell} \in \mathbb{R}^{M \times n_{\ell}}, \\ \mathbf{\Omega}_{\ell-1} \in \mathbb{R}^{n_{\ell} \times n_{\ell}}, & \mathbf{\Gamma}_{\ell} \in \mathbb{R}^{n_{\ell} \times n_{\ell}}, \end{array} \tag{8}$$

where $\ell = 1, 2, \ldots, L$ and n_{ℓ} represents the number of neurons in the ℓth layer.

2.2.2 K-FAC for Convolutional Layers

In [4], Grosse *et al.* proposed the K-FAC for convolutional layers. In general, by combining some additional assumptions, this algorithm has a similar form to that for fully connected layers.

For simplicity, the algorithm utilizes the expansion operation $[\![\cdot]\!]$ defined in [4]. This operation extracts the patches surrounding the spatial locations, stretches them into vectors and stacks these vectors into matrix. Besides, the convolution kernels are reshaped in size of $c_{\ell} \times k_{\ell}c_{\ell-1}$ where c_{ℓ} and $c_{\ell-1}$ are the number of feature maps and the number of input channels respectively, and k_{ℓ} is the kernel size. $\mathbf{F}_{\ell}$ can be written as an expectation on the two-dimensional space:

$$\mathbf{F}_{\ell} = \mathbf{\Omega}_{\ell-1} \otimes \mathbf{\Gamma}_{\ell}, \tag{9}$$

where $\mathbf{\Omega}_{\ell-1} = \mathbb{E}[[\![\mathbf{A}_{\ell-1}]\!][\![\mathbf{A}_{\ell-1}]\!]]$ and $\mathbf{\Gamma}_{\ell} = \mathbb{E}[\mathcal{D}\mathbf{S}_{\ell}\mathcal{D}\mathbf{S}_{\ell}^{\top}]$. Since the true covariance statistics are unknown, K-FAC uses the empirical statistics for a given mini-batch to appromaxiate the covariance matrice $\mathbf{\Omega}_{\ell-1}$ and $\mathbf{\Gamma}_{\ell}$ by following equation,

$$\widehat{\mathbf{\Omega}}_{\ell-1} = \frac{1}{M}[\![\widetilde{\mathbf{A}}_{\ell-1}]\!]^{\top}[\![\widetilde{\mathbf{A}}_{\ell-1}]\!], \widehat{\mathbf{\Gamma}}_{\ell} = \frac{1}{M|\mathcal{T}|}\mathcal{D}\widetilde{\mathbf{S}}_{\ell}^{\top}\mathcal{D}\widetilde{\mathbf{S}}_{\ell}, \tag{10}$$

where M is the batch size and $\mathcal{T}$ is the collection of spatial positions. Then, by employing the same approximate method used in K-FAC for fully connected layers, $\mathbf{F}_{\ell}^{-1}$ can be efficiently computed as $\mathbf{F}_{\ell}^{-1} = \widehat{\mathbf{\Omega}}_{\ell-1}^{-1} \otimes \widehat{\mathbf{\Gamma}}_{\ell}^{-1}$.

Accordingly, the derived size of corresponding factors are:

$$\begin{array}{ll} \tilde{\mathbf{A}}_{\ell-1} \in \mathbb{R}^{Ms_{\ell-1} \times c_{\ell-1}k_{\ell-1}}, & \mathcal{D}\tilde{\mathbf{S}}_{\ell} \in \mathbb{R}^{Ms_{\ell} \times c_{\ell}}, \\ \widehat{\mathbf{\Omega}}_{\ell-1} \in \mathbb{R}^{c_{\ell-1}k_{\ell-1} \times c_{\ell-1}k_{\ell-1}}, & \widehat{\mathbf{\Gamma}}_{\ell} \in \mathbb{R}^{c_{\ell} \times c_{\ell}}, \end{array} \tag{11}$$

where $\ell = 1, 2, \ldots, L$, c_{ℓ} represents the number of channels in the ℓ^{th} layer, k_{ℓ} and s_{ℓ} are corresponding spatial size of the filters and feature maps, respectively. The approximate natural gradient $\widehat{\nabla}_{\theta}$ is then computed as,

$$\widehat{\nabla}_{\theta} = \mathbf{F}^{-1}\nabla_{\theta} = \left[\mathrm{vec}(\widehat{\mathbf{\Gamma}}_{\ell}^{-1}(\nabla_{\mathbf{W}_{\ell}})\widehat{\mathbf{\Omega}}_{\ell-1}^{-1}), \ell = 1, \cdots, L\right]. \tag{12}$$

3. Proposed Method

We assuming a block-diagonal approximation to FIM like that in K-FAC, such that every block corresponds to the parameters of one layer. Then, the Kronecker factorization applies to fully connected layers and convolutional layers, because only these two widely used layers are discussed in this paper. In this section, we demonstrate our main contribution, an low-rank formation of K-FAC for each block of FIM for fully connected layers and convolutional layers. For clarity, in this section, we focus on one block $\mathbf{F}_{\ell}$ corresponding to one layer ℓ. Although, by utilizing the Kronecker factorization, K-FAC makes the computational natural gradients of a large-scale neural network computationally feasible, but the inverting of Kronecker factors can still spend many computational efforts.

Starting from K-FAC approximation for fully connected layers, we first derived our main theorem for the low-rank K-FAC approximation.

3.1. SKFAC for Fully Connected Layers

Based on the block-diagonal approximation of Fisher matrix in K-FAC, we first derive the low-rank inverse of Fisher matrix. Recall that the independent assumption between activations and output derivatives yields the Kronecker approximation of the Fisher matrix, $\mathbf{F}\ell = \widehat{\mathbf{\Omega}}_\ell^{(\lambda)} \otimes \widehat{\mathbf{\Gamma}}_\ell^{(\lambda)}$, where $\widehat{\mathbf{\Omega}}_\ell^{(\lambda)} = \lambda\mathbf{I} + \widehat{\mathbf{\Omega}}_\ell$ and $\widehat{\mathbf{\Gamma}}_\ell^{(\lambda)} = \lambda\mathbf{I} + \widehat{\mathbf{\Gamma}}_\ell$, herein, λ is the damping introduced by [14]. The unknown data distribution causes the difficulty in computing the Fisher matrix information. To address this difficulty, we estimate the Kronecker factors by the empirical statistics over a mini-batch of training data $\mathcal{B} \subset \mathcal{D}_{training}, |\mathcal{B}| = M$ at iteration t. The empirical statistics of the Kronecker factors over a given mini-batch is defined as

$$\lambda\mathbf{I} + \widehat{\mathbf{\Omega}}_\ell = \lambda\mathbf{I} + \mathbb{E}_{\mathcal{B}}[\mathbf{A}_\ell^\top \mathbf{A}_\ell] = (\lambda\mathbf{I} + \frac{\mathbf{A}_\ell^\top \mathbf{A}_\ell}{M}), \quad (13)$$

$$\lambda\mathbf{I} + \widehat{\mathbf{\Gamma}}_\ell = \lambda\mathbf{I} + \mathbb{E}_{\mathcal{B}}[\mathcal{D}\mathbf{S}_\ell^\top \mathcal{D}\mathbf{S}_\ell] = (\lambda\mathbf{I} + \frac{\mathcal{D}\mathbf{S}_\ell^\top \mathcal{D}\mathbf{S}_\ell}{M}). \quad (14)$$

In Section II, we mentioned that $\mathbf{A}$ and $\mathcal{D}\mathbf{S}$ are $\mathbb{R}^{M\times n_\ell}$ and $\mathbb{R}^{M\times n_{\ell-1}}$, respectively. When $M << n_\ell, n_{\ell-1}$, they have low-rank structure. The approximation of above terms over a mini batch suggests that the factors of FIM is practically low-rank. Note that although we assume the factors of FIM having low-rank structure, the resulting approximation of FIM is not low-rank. Existing methods [14][4] attempt to reduce the computation cost of the fully exact inverse of the Kronecker factors $\widehat{\mathbf{\Gamma}}^{-1}$ and $\widehat{\mathbf{\Omega}}^{-1}$, where the large matrices have to decompose into the low-rank factors first. In contrast, our method utilizes the low-rank nature of activation likelihood covariance matrix $\widehat{\mathbf{\Omega}}_\ell = \mathbf{A}_\ell\mathbf{A}_\ell^\top$ and pre-activation derivative likelihood covariance matrix $\widehat{\mathbf{\Gamma}}_\ell = \mathcal{D}\mathbf{S}_\ell\mathcal{D}\mathbf{S}_\ell^\top$ matrices to yield an low-rank formulation of inverses of $(\widehat{\mathbf{\Gamma}}_\ell^\lambda)^{-1}$ and $(\widehat{\mathbf{\Omega}}_\ell^\lambda)^{-1}$.

Theorem 1 *The inverse of two Kronecker factors of Fisher information matrix is derived as*

$$[\widehat{\mathbf{\Omega}}_\ell^{(\lambda)}]^{-1} = (\lambda\mathbf{I} + \widehat{\mathbf{\Omega}}_\ell)^{-1} = \frac{1}{\lambda}\mathbf{I} - \frac{1}{\lambda}\mathbf{A}_\ell^\top \mathbf{\Omega}_\ell^* \mathbf{A}_\ell, \quad (15)$$

$$[\widehat{\mathbf{\Gamma}}_\ell^{(\lambda)}]^{-1} = (\lambda\mathbf{I} + \widehat{\mathbf{\Gamma}}_\ell)^{-1} = \frac{1}{\lambda}\mathbf{I} - \frac{1}{\lambda}\mathcal{D}\tilde{\mathbf{S}}_\ell^\top \mathbf{\Gamma}_\ell^* \mathcal{D}\tilde{\mathbf{S}}_\ell, \quad (16)$$

where $\mathbf{\Omega}_\ell^* = (M\lambda\mathbf{I} + \mathbf{A}_\ell\mathbf{A}_\ell^\top)^{-1}$, *and* $\mathbf{\Gamma}_\ell^* = (M\lambda\mathbf{I} + \mathcal{D}\mathbf{S}_\ell\mathcal{D}\mathbf{S}_\ell^\top)^{-1}$.

Proof Combined equations (13)(14) with Woodbury formula [5][20], the claim follows. Complete proof is given in Appendix. □

Using Theorem 1, we get following equation,

$$\begin{aligned}\mathbf{F}_\ell^{-1} =& \lambda^{-1}[\mathbf{I} - \mathbf{A}_{\ell-1}^\top(M\lambda\mathbf{I} + \mathbf{A}_{\ell-1}\mathbf{A}_{\ell-1}^\top)^{-1}\mathbf{A}_{\ell-1}] \\ & \otimes \lambda^{-1}[\mathbf{I} - \mathcal{D}\mathbf{S}_\ell^\top(M\lambda\mathbf{I} + \mathcal{D}\mathbf{S}_\ell\mathcal{D}\mathbf{S}_\ell^\top)^{-1}\mathcal{D}\mathbf{S}_\ell]. \quad (17)\end{aligned}$$

Taking $[\widehat{\mathbf{\Gamma}}_\ell^{(\lambda)}]^{-1}$ as an example, the computation cost of inverting the factors of FIM reduces from $O(n_\ell^3)$ to roughly $O(M^3)$, where n_ℓ is the number of outputs of the lth layer and M is the batch size. Therefore, as long as $M << n_\ell$ is satisfied, our derived inverse procedure is much more efficient than inverting the full Kronecker factors directly. Compared with K-FAC, the theoretical computational cost of computing the inverse of Kronecker factors of FIM is reduced from $O(\sum_\ell n_\ell^3 + M\sum_\ell n_\ell^2)$ to $O(\sum_\ell M^3 + M\sum_\ell n_\ell^2)$. By parallel computing, the matrix multiplication in equations (15)(16) has a sublinear complexity $O(M\sum_\ell (n_\ell^2/N_T)^2)$, where N_T is the number of computation threads. The steps is shown in Algorithm 1.

Algorithm 1 SKFAC for fully connected layers

Require: layer ℓ, batch size M, ℓ's input $\mathbf{A}_\ell \in \mathbb{R}^{M\times n_\ell}$, ℓ's pre-activation derivations $\mathcal{D}\mathbf{S}_\ell \in \mathbb{R}^{M\times n_\ell}$, damping coefficient λ

Ensure: the inverse of damped factor matrices: $[\widehat{\mathbf{\Omega}}_\ell^{(\lambda)}]^{-1}$ and $[\widehat{\mathbf{\Gamma}}_\ell^{(\lambda)}]^{-1}$

for $\mathbf{Y}, \mathbf{X}$ in $[\{\mathbf{A}_\ell, \widehat{\mathbf{\Omega}}_\ell^{(\lambda)}\}, \{\mathcal{D}\mathbf{S}_\ell, \widehat{\mathbf{\Gamma}}_\ell^{(\lambda)}\}]$ **do**
  $M, n = shape(\mathbf{Y})$.
  if $M \geq n$ **then**
    $\mathbf{X}^{-1} = (\lambda\mathbf{I} + M^{-1}\mathbf{Y}^\top\mathbf{Y})^{-1}$
  else
    $\mathbf{X}^{-1} = \lambda^{-1}[\mathbf{I} - \mathbf{Y}^\top(M\lambda\mathbf{I} + \mathbf{Y}\mathbf{Y}^\top)^{-1}\mathbf{Y}]$
  end if
end for

Due to the reception fields existing in convolutional layers, Theorem 1 might not be directly applied to these layers. Deriving an analogous low-rank K-FAC approximation for convolution layers needs some additional approximation. Next, we present the specific techniques for convolutional layers.

3.2. SKFAC for Convolutional Layers

We begin by considering the inverse of $\mathbf{\Omega}$. Let M denote the batch size, and let s be the size of spatial dimensions of input feature map. If we apply Theorem 1 to convolutional layers directly like that for fully connected layers, it is very likely that the size of the inverse matrix involved will increase dramatically to $Ms \times Ms$ due to the exist of spatial dimensions. This is much larger than the original size of Kronecker factors in most cases. However, it is observed that when calculating the corresponding factors of $\mathbf{A}$ or $\mathcal{D}\mathbf{S}$, the larger dimension (Ms) is aggregated by inner product.

This motivates us to obtain an analogous low-rank formulation of SKFAC for convolutional layers like that for fully connected layers by reducing the size of spatial dimensions. In the following, we propose two feasible methods for this purpose.

3.2.1 Spatial Sampling:

From Equation (9), $\mathbf{F}_\ell$ is an expectation on the spatial size of activations, and corresponding samples is s, which is usually very large. If we reduce it to s' and $s' \leq s$, $\mathbf{F}_\ell$ can be seen as the expectation on a subspace $\mathcal{T}' \subseteq \mathcal{T}$, which ensures that the estimation is still unbiased. According ID, patches are drawn from the identical distribution, a sub set of sampling patches can approximate the FIM well. Thereby, we can subsample the patches to reduce the computation complex. Then, we have:

$$\begin{aligned}&\left(\lambda\mathbf{I}+\widehat{\mathbf{\Omega}}_\ell\right)^{-1}\\&\approx\lambda^{-1}\left[\mathbf{I}-[\![\tilde{\mathbf{A}}]\!]_\ell^\top\left(M\lambda\mathbf{I}+[\![\tilde{\mathbf{A}}]\!]_\ell[\![\tilde{\mathbf{A}}]\!]_\ell^\top\right)^{-1}[\![\tilde{\mathbf{A}}]\!]_\ell\right],\end{aligned}\tag{18}$$

$$\begin{aligned}&\left(\lambda\mathbf{I}+\widehat{\mathbf{\Gamma}}_\ell\right)^{-1}\\&\approx\lambda^{-1}\left[\mathbf{I}-\mathcal{D}\tilde{\mathbf{S}}_\ell^\top\left(M|\mathcal{T}_\ell'|\lambda\mathbf{I}+\mathcal{D}\tilde{\mathbf{S}}_\ell\mathcal{D}\tilde{\mathbf{S}}_\ell^\top\right)^{-1}\mathcal{D}\tilde{\mathbf{S}}_\ell\right],\end{aligned}\tag{19}$$

where $[\![\tilde{\mathbf{A}}]\!]_\ell \in \mathbb{R}^{Ms'\times c_\ell k_\ell}$ and $\mathcal{D}\tilde{\mathbf{S}}_\ell \in \mathbb{R}^{Ms'\times c_\ell k_\ell}$. The final size of inversion becomes $Ms'\times Ms'$, which can accelerate the computation if $Ms' < c_\ell k_\ell$.

3.2.2 Reduce Sum:

One of the most straightforward ideas is to average the larger dimension. According to Equation (9), we first approximates the expectation on the space of the Kronecker product as the Kronecker product of expectations just like what is done on batches as shown in Equation (5).

We found that we could make the activations and pre-activation derivations become low-rank without affecting empirical performance by accelerating over the spatial dimensions. Namely, let $\tilde{\mathbf{\Omega}}$ and $\tilde{\mathbf{\Gamma}}$ denote the approximations of $\mathbf{\Omega}$ and $\mathbf{\Gamma}$ by dimension reduction. And $[\![\bar{\mathbf{A}}_\ell]\!]$ and $\mathcal{D}\bar{\mathbf{S}}_\ell$ denote the acceleration of $[\![\mathbf{A}_\ell]\!]$ and $\mathcal{D}\mathbf{S}_\ell$ over spatial dimensions. The approximate Kronecker factors can be efficiently computed by following equations,

$$\begin{aligned}\widehat{\mathbf{\Omega}}_\ell&=\mathbb{E}(\sum_{\mathcal{T}}[\![\mathbf{A}_\ell]\!][\![\mathbf{A}_\ell]\!]^\top)\approx\mathbb{E}(\frac{1}{|\mathcal{T}|}\sum_{\mathcal{T}}[\![\mathbf{A}_\ell]\!]\sum_{\mathcal{T}}[\![\mathbf{A}_\ell]\!])^\top\\&=\mathbb{E}(\frac{1}{|\mathcal{T}|}[\![\bar{\mathbf{A}}_\ell]\!][\![\bar{\mathbf{A}}_\ell]\!]^\top),\end{aligned}\tag{20}$$

$$\begin{aligned}\widehat{\mathbf{\Gamma}}_\ell&=\mathbb{E}(\sum_{\mathcal{T}}\mathcal{D}\mathbf{S}_\ell\mathcal{D}\mathbf{S}_\ell^\top)\approx\mathbb{E}(\frac{1}{|\mathcal{T}|}\sum_{\mathcal{T}}\mathcal{D}\mathbf{S}_\ell\sum_{\mathcal{T}}\mathcal{D}\mathbf{S}_\ell)^\top\\&=\mathbb{E}(\frac{1}{|\mathcal{T}|}\mathcal{D}\bar{\mathbf{S}}_\ell\mathcal{D}\bar{\mathbf{S}}_\ell^\top).\end{aligned}\tag{21}$$

Then, using one of the above dimension reducing methods, we can further make the approximation for $[\![\mathbf{A}]\!]$ and $\mathcal{D}\mathbf{S}$:

$$\mathbf{F}_\ell\approx\mathbb{E}_{\mathcal{B}}[[\![\bar{\mathbf{A}}_{\ell-1}]\!]^\top[\![\bar{\mathbf{A}}_{\ell-1}]\!]]\otimes\mathbb{E}_B[\mathcal{D}\bar{\mathbf{S}}_\ell^\top\mathcal{D}\bar{\mathbf{S}}_\ell],\tag{22}$$

where $[\![\bar{\mathbf{A}}_{\ell-1}]\!]=\sum_{\mathcal{T}}[\![\mathbf{A}_{\ell-1}]\!]$ and $\mathcal{D}\bar{\mathbf{S}}_\ell=\sum_{\mathcal{T}}\mathcal{D}\mathbf{S}_\ell$ are just the sum of $\mathbf{A}$ and $\mathcal{D}\mathbf{S}$ over spatial dimensions. Then, by Kronecker-factorizing above equation and reformulating the inverse factors via Theorem 1, we give the efficient approximation of these two inverse factors as follows,

$$\begin{aligned}&(\widehat{\mathbf{\Omega}}_\ell^{(\lambda)})^{-1}\approx(\tilde{\mathbf{\Omega}}_\ell^{(\lambda)})^{-1}\\&=\lambda^{-1}[\mathbf{I}-[\![\bar{\mathbf{A}}_\ell]\!]^\top(M\lambda\mathbf{I}+[\![\bar{\mathbf{A}}_\ell]\!][\![\bar{\mathbf{A}}_\ell]\!]^\top)^{-1}[\![\bar{\mathbf{A}}_\ell]\!]],\end{aligned}\tag{23}$$

$$\begin{aligned}&(\widehat{\mathbf{\Gamma}}_\ell^{(\lambda)})^{-1}\approx(\tilde{\mathbf{\Gamma}}_\ell^{(\lambda)})^{-1}\\&=\lambda^{-1}[\mathbf{I}-\mathcal{D}\bar{\mathbf{S}}_\ell^\top(M\lambda\mathbf{I}+\mathcal{D}\bar{\mathbf{S}}_\ell\mathcal{D}\bar{\mathbf{S}}_\ell^\top)^{-1}\mathcal{D}\bar{\mathbf{S}}_\ell],\end{aligned}\tag{24}$$

$$\mathbf{F}_\ell^{-1}=(\tilde{\mathbf{\Omega}}_\ell^{(\lambda)})^{-1}\otimes(\tilde{\mathbf{\Gamma}}_\ell^{(\lambda)})^{-1}.\tag{25}$$

Let S be Ms' in SKFAC-SS or the batch size M in SKFAC-RS. Taking $(\widehat{\mathbf{\Omega}}_\ell^{(\lambda)})^{-1}$ as an example, the size of matrix having to invert is reduced to $S\times S$, and the computational cost is dropped down to $O(\sum_l S^3)+O(S\sum_l(k_l\times c_l)^2)$. Therein, the matrix multiplication spends $O(S\sum_l(k_l\times c_l)^2)$, which can be optimized by parallel computing again leading to a sublinear complexity $O(S\sum_l(\frac{k_l\times c_l}{N_T})^2)$, where N_T is the number of computation threads. Moreover, for the sake of ensuring the batch stochastic optimization, it should be noted that reduce sum or subsampling is only performed on the spatial dimension. The main steps is shown in Algorithm 2.

3.3. Low-Rank Natural Gradient Descent

According to the above discussion, we claim that we can efficiently compute the natural gradients without obtaining neither the full Fisher information matrix nor the Kronecker factors. Depicted as equations (18), (19), (21) and (22), the only necessary terms needed to record are $\mathbf{A}$, $\mathcal{D}\mathbf{S}$ and $\mathbf{\Omega}^*$, $\mathbf{\Gamma}^*$. Due to the inverse operation of matrix is not linearly incompatible, we instead summarized the history of $\mathbf{A}$ and $\mathcal{D}\mathbf{S}$ over the batches of training data by moving averages,

$$\mathbf{A}_\ell^{t+1}=(1-\alpha)\mathbf{A}_\ell^t+\alpha\mathbf{A}_\ell,\tag{26}$$

$$\mathcal{D}\mathbf{S}_\ell^{t+1}=(1-\alpha)\mathcal{D}\mathbf{S}_\ell^t+\alpha\mathcal{D}\mathbf{S}_\ell.\tag{27}$$

In distributed training, only $\mathbf{A}_\ell^t$, $\mathbf{S}_\ell^t$, $\mathbf{\Omega}^*$ and $\mathbf{\Gamma}^*$ needs to be transmitted, which results in less communication latency.

We substitute the moving averages $\mathbf{A}_\ell^t$ and $\mathcal{DS}_\ell^t$ into equations (15)(16)(22)(23) to obtain matrices $\mathbf{\Omega}_\ell^\lambda$ and $\mathbf{\Gamma}_\ell^\lambda$. Following the computation of natural gradients in K-FAC, it is necessary to compute the full Fisher matrix. Instead, the natural gradients can be obtained in the form of matrix. Here is the equation for computing rescaled updates:

$$\begin{aligned}\overline{\text{vec}}(F_\ell^{-1}\text{vec}(\nabla_{W\ell})) &= \widehat{\mathbf{\Omega}}_\ell^{(\lambda)-1}\nabla_{W\ell}\widehat{\mathbf{\Gamma}}_\ell^{(\lambda)-1} \\ &= (\lambda\mathbf{I}+\widehat{\mathbf{\Omega}}_\ell)^{-1}\nabla_{W\ell}(\lambda\mathbf{I}+\widehat{\mathbf{\Gamma}}_\ell)^{-1}. \end{aligned} \tag{28}$$

Algorithm 2 SKFAC for convolutional layers

Require: layer ℓ, batch size M, ℓ's input $[\![\mathbf{A}_\ell]\!] \in \mathbb{R}^{Ms_\ell \times c_\ell k_\ell}$, ℓ's pre-activation derivations $\mathcal{DS}_\ell \in \mathbb{R}^{Ms_\ell \times c_\ell}$, damping coefficient λ, subsampling amount s'

Ensure: the inverse of damped factor matrices: $[\widehat{\mathbf{\Omega}}_\ell^{(\lambda)}]^{-1}$ and $[\widehat{\mathbf{\Gamma}}_\ell^{(\lambda)}]^{-1}$

for $\mathbf{Y}, \mathbf{X}$ in $[\{[\![\mathbf{A}_\ell]\!], \widehat{\mathbf{\Omega}}_\ell^{(\lambda)}\}, \{\mathcal{DS}_\ell, \widehat{\mathbf{\Gamma}}_\ell^{(\lambda)}\}]$ **do**
  $M, s, n = shape(\mathbf{Y}), \mathbf{Y} \in \mathbb{R}^{Ms \times n}$
  if Perform SKFAC with Reduce Sum **then**
    if $M \geq n$ **then**
      $\mathbf{X}^{-1} = (\lambda\mathbf{I} + M^{-1}\mathbf{Y}^\top\mathbf{Y})^{-1}$
    else
      Get $\bar{\mathbf{Y}}$ by accumulating $\mathbf{Y}$ over spatial dimension.
      $\mathbf{X}^{-1} = \lambda^{-1}[\mathbf{I} - \bar{\mathbf{Y}}^\top(M\lambda\mathbf{I} + \bar{\mathbf{Y}}\bar{\mathbf{Y}}^\top)^{-1}\bar{\mathbf{Y}}]$
    end if
  else if Perform SKFAC with Spatial Subsampling **then**
    if $s' \geq s$ **then**
      $s' = s$
    end if
    if $Ms' \geq n$ **then**
      $\mathbf{X}^{-1} = (\lambda\mathbf{I} + M^{-1}\mathbf{Y}^\top\mathbf{Y})^{-1}$
    else
      Form $\tilde{\mathbf{Y}}$ by sampling s' elements from $\mathbf{Y}$ along with the spatial dimension.
      $\mathbf{X}^{-1} = \lambda^{-1}[\mathbf{I} - \tilde{\mathbf{Y}}^\top(M\lambda\mathbf{I} + \tilde{\mathbf{Y}}\tilde{\mathbf{Y}}^\top)^{-1}\tilde{\mathbf{Y}}]$
    end if
  end if
end for

3.4. Implementation Details

Moving Average. To maintain running estimates of the required $\mathbf{A}$ and $\mathcal{DS}$, K-FAC adopted a simple exponentially decaying averaging scheme [14]. However, SKFAC approximates the Kronecker factors of the FIM by considering partial information of $\mathbf{A}$ and $\mathcal{DS}$. Therefore, we maintain the moving average of $\mathbf{A}$ and $\mathcal{DS}$ to alleviate the influence of random factors and enhance the robustness of our method. In distributed computing, this can also further save communication overhead.

Matrix Decomposition. Matrix decomposition is often the underlying support technique for practical inversion computation and using the proper decomposition algorithm for matrices with special properties can often give faster computations. In K-FAC and SKFAC, all the matrices that need to be inverted are strictly positive definite. Therefore, we employ the eigenvalue decomposition in the implementation of SKFAC.

Stale Fisher Information. Some works shows that FIM changes very slowly in neural network training [18]. At the same time, due to the high cost of FIM inverse matrix computation, they generally adopted the periodic computation strategy in the practical application, making a trade off between the accuracy and time cost, and the higher computation frequency, the more K-FAC will accelerate. We can also incorporate this strategy into SKFAC.

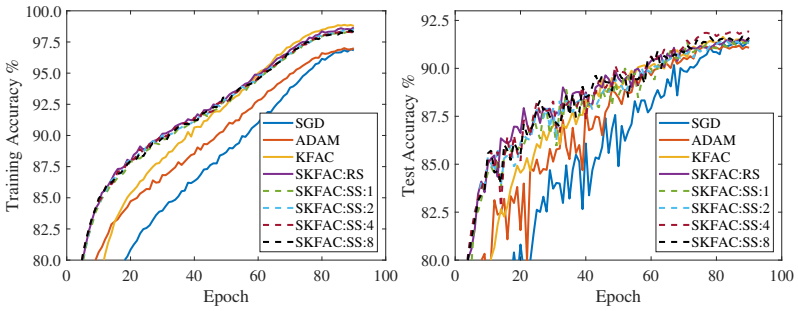
3.5. Related Work

Currently, there are also some works employing Sherman–Morrison–Woodbury (SMW) formula in the second-order optimization. O'Leary-Roseberry *et al.*[16] used the SMW formula to efficiently solve the Newton rescaled negative gradient. Ren *et al.*[22] combined the SMW formula with automatic differentiation to efficiently perform the Newton method and natural gradient descent. However, these algorithms do not consider the structures of activations and pre-activation derivations. SKFAC differs from them in that it can use these inherent structures to cleverly reduce the computation complexity.

4. Experiments

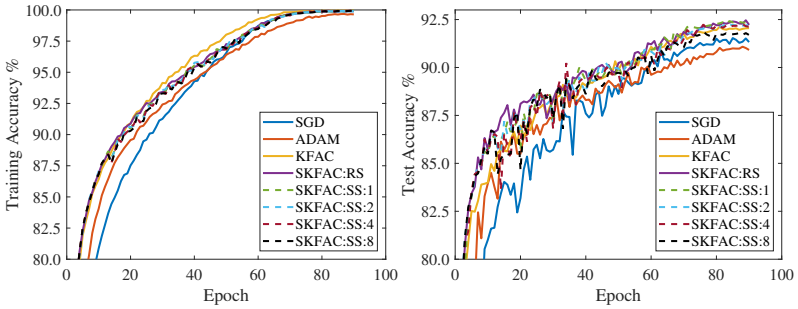
In this section, we exhibit the practical benefits of our SKFAC on several neural network training tasks using the published state-of-art architectures as baselines. We conduct experiments on the following three tasks: training **VGG-11** with **Cifar-10** dataset; training **ResNet-34** with **Cifar-10** dataset; training **ResNet-50** with the **ImageNet-1K** dataset. We conduct Cifar-10 experiments on a single GPU and ImageNet-1K on 4 RTX 2080Ti GPUs. Our method is implemented by PyTorch and Horovod.

Hyper-Parameter Settings. First, we evaluate the effectiveness of the proposed algorithm on the tasks of training VGG11 [24] and ResNet34 [6] with the relatively simple Cifar-10 dataset. We compare our SKFACs with three baselines (K-FAC, Adam and SGD with momentum). Moreover, we also investigate the sensitivity of the sampling number sn to the performance of SKFAC-SS. For this purpose, we run SKFAC-SS with different sn settings in $\{1, 2, 4, 8\}$ one by one. For the sake of fairness, we employ the warm-up learning rate settings for all methods given in [3]



(a) Training Accuracy for ResNet-34

(b) Test Accuracy for ResNet-34



(c) Training Accuracy for VGG-11

(d) Test Accuracy for VGG-11

Figure 1: The convergence curves of SGD, Adam, K-FAC, SKFAC-RS and SKFAC-SS with sampling amount $\{1, 2, 4, 8\}$ in terms of validation error and loss for training ResNet34 and VGG11 on **Cifar-10**.

with a cosine-annealing and initial value of $1e-3$ as depicted in [12], and batch size is set to 32. Since SKFACs have similar properties as K-FAC, we can directly follow the settings of hyper-parameters in K-FAC. Thus, the covariance matrix updating period and the corresponding inverse matrix updating period are set to 20 and 200 iterations, respectively. The damping is set to 0.001.

To verify the efficiency of our proposed algorithms in training the deep neural network on large-scale datasets, we use SKFAC-SS and SKFAC-RS to train the ResNet50 [6] with the ImageNet-1K dataset. We compare SKFACs with three baselines (K-FAC, K-FAC with SUA and SGD with momentum). One might be interested in K-FAC with SUA. SUA is a further approximation of the activations for convolutional layers [4], which can accelerate the computation of FIM. For SGD, we followed the prescription given by [3] for adapting the learning rate, which is the state-of-the-art baseline of SGD. For K-FAC and SKFACs, polynomial decay with an initial value of 0.025 and a half-life of 5 epochs is employed to tune the learning rate. Batch size per GPU is set to 32, and for 4 GPUs the effective batch size is $4 \times 32 = 128$. The sampling amount sn of SKFAC-SS is set to 1. Other parameter settings of K-FAC and SKFACs are the same as those in the Cifar-10 experiment.

Results on Cifar-10. When training **ResNet34** and **VGG11** with **Cifar-10**, following findings confirm the enhanced performance of SKFAC-RS and SKFAC-SS. It can be seen from Fig. 1 that, compared with SGD and ADAM, K-FAC and SKFACs can achieve significant improvement in convergence rate which further translates into improvement in validate accuracy. Comparing SKFACs with K-FAC, SKFACs obtain a similar convergence property like that of K-FAC. Although our optimization method may have some fluctuations in loss function values during the first dozens of epochs, SKFACs tends to more stable in the subsequent training process. Moreover, when training VGG11 which is a relatively shallow network, SGD also has a fast convergence rate like that of K-FAC and SKFACs. While training ResNet34, which is a deeper network, the convergence rate of K-FAC and SKFACs is faster. This is mainly because with the network getting deeper, SGD is more likely to be affected by gradient vanishing, while the second-order optimization algorithm, which leverage a preconditioner to adjust the update step, can effectively suppress the effect of gradient vanishing.

The result of SKFAC-SS with different sn of $\{1, 2, 4, 8\}$ is also shown in Fig. 1. From the curves, it can be seen that the overall trend of SKFAC-SSs with different sn settings have a similar convergence property. And, the convergence trends of training validation accuracy are similar to K-FAC and SKFAC-RS. Then, when we get a close observation of the convergence curves, we will find that in the

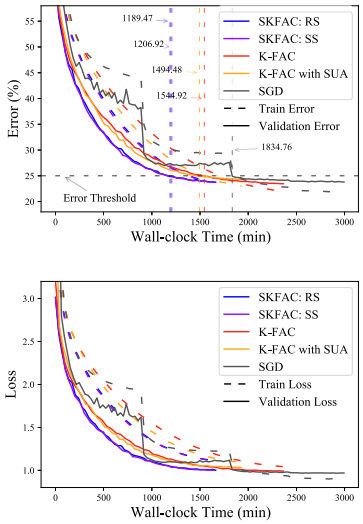


Figure 2: The convergence trends of SGD, K-FAC, SKFAC-RS and SKFAC-SS during training ResNet50 on **ImageNet-K** dataset. **Upper:** Classification error. **Lower:** Loss function value.

first tens epochs, subsampling strategy could cause the fluctuation. By observing the K-FAC convergence curve, K-FAC can converge smoothly and stably, but it needs more computational efforts to compute relatively precious natural gradients. However, due to the moving average strategy, the SKFAC-SS can obtain a stable convergence trend at the late stage of training as the histories of activations and pre-activation derivations are accumulated. In terms of the performance of the finally trained model, SKFAC-SS with different sampling numbers can work as well as K-FAC and SKFAC-RS.

Results on ImageNet-1K. When training **ResNet-50** with **ImageNet-1K** dataset, following findings are obtained from Fig. 2. The carefully tuned SGD baseline converges very fast during the first 200 min. However, SKFACs can reserve a better convergence property than SGD after 400min. Since the purpose of developing this algorithm is to speed up K-FAC by using further curvature approximation, we can see from Fig. 2 that our algorithm can achieve a good convergence property like that of K-FAC in terms of both the loss function values and the validate errors. The statistical results of training are listed in Table 1. SGD needs 61 epochs to reach a validate accuracy of 75%. K-FAC needs 33 epochs to achieve it, while SKFAC-RS and SKFAC-SS need 36/37 epochs. Therefore, it is verified that two proposed algorithms can work as well as K-FAC. Moreover, from the perspective of running time, our algorithms spend 1206.9 minutes (SKFAC-SS), 1189.5 minutes (SKFAC-RS) respectively. K-FAC needs 1544.9 minutes and SGD needs 1834.76 minutes. SUA trick results in about a 3% decrease in overall wall-clock time. Compared to K-FAC, our methods saves at most 23% training time. In short, our algorithm has shown its superiority in terms of iteration rounds and wall-clock time. Fig. 3 shows the faster inversion and less communication latency leading to the time-saving in computing natural gradient in each iteration.

Table 1: Statistical results of SGD, K-FAC, SKFAC-RS and SKFAC-SS when reaching the accuracy checkpoint of 75%.

Algorithm	SGD	K-FAC	SUA	SKFAC-RS	SKFAC-SS
Epoch	61	**33**	39	36	37
Time (min)	1834.8	1544.9	1494.5	**1189.5**	1206.9
Time Gap	+18.76%	0%	-3.3%	**-23.00%**	-21.88%

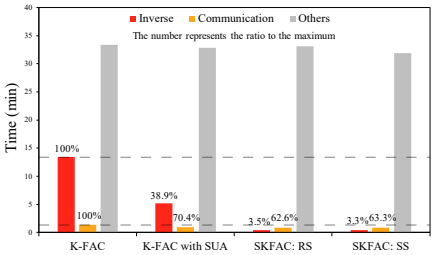


Figure 3: The proportions of time spent by computing natural gradients when training ResNet50 on **ImageNet-1K**.

5. Conclusion

We presented SKFAC, a computationally efficient 2^{nd} order optimization approach for training neural networks, which is built on the low-rank approximation of Kronecker factors. We show that our approach can obtain significantly reduction of total wall-clock time and achieve competitive accuracy for training large-scale neural networks on both Cifar-10 and ImageNet-1K datasets. In a series of validations[2], we demonstrate that utilizing the low-rank natural of Kronecker factors in K-FAC, the proposed variant of K-FAC achieves more than 20% running time saving than canonical K-FAC. Summarily, these improvements made by SKFAC would open up new probabilities of the 2^{nd} order optimization approaches for fast and robustly training of large-scale neural networks. In future work, we will focus on enhancing the robustness of SKFAC and further extending it to train the recurrent networks for text classification. Moreover, the inverse of small matrices can be accelerated by batch inverse powered by MindSpore[15] and exclusive chips. Therefore, in the future work, we will implement the proposed method on MindSpore for better computational efficiency, which will be available soon.

[2]The code is available in https://github.com/fL0n9/SKFAC-MindSpore.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 1

[2] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011. 2

[3] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 6, 7

[4] Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582, 2016. 3, 4, 7

[5] William W. Hager. Updating the inverse of a matrix. *SIAM Review*, 31(2), Jun. 1989. 4

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 6, 7

[7] Tom Heskes. On "natural" learning and pruning in multilayered perceptrons. *Neural Computation*, 12(4):881–901, 2000. 2

[8] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018. 1

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2

[10] Frederik Kunstner, Lukas Balles, and Philipp Hennig. Limitations of the empirical fisher approximation. *arXiv preprint arXiv:1905.12558*, 2019. 2

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. 1

[12] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, 2016. 7

[13] James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014. 2

[14] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015. 2, 3, 4, 6

[15] MindSpore. https://www.mindspore.cn/, 2021. 8

[16] Thomas O'Leary-Roseberry, Nick Alger, and Omar Ghattas. Inexact newton methods for stochastic non-convex optimization with applications to neural network training. *arXiv preprint arXiv:1905.06738*, 2019. 6

[17] Yann Ollivier. Riemannian metrics for neural networks. *Information and Inference: a Journal of the IMA*, 2013. 2

[18] Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. Large-scale distributed second-order optimization using kronecker-factored approximate curvature for deep convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12359–12367, 2019. 6

[19] Daniel Povey, Xiaohui Zhang, and Sanjeev Khudanpur. Parallel training of dnns with natural gradient and parameter averaging. *arXiv preprint arXiv:1410.7455*, 2014. 2

[20] WH Press, SA Teukolsky, WT Vetterling, and BP Flannery. *Woodbury Formula*, chapter 2.7.3. Cambridge University Press, New York, 2007. 4

[21] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 1

[22] Yi Ren and Donald Goldfarb. Efficient subsampled gauss-newton and natural gradient methods for training neural networks. *arXiv preprint arXiv:1906.02353*, 2019. 6

[23] Nicolas L Roux, Pierre-Antoine Manzagol, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. In *Advances in neural information processing systems*, pages 849–856, 2008. 2

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 6

[25] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012. 2

[26] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. 2