Structured Inverse-Free Natural Gradient Descent: Memory-Efficient & Numerically-Stable KFAC

Wu Lin [*1] **Felix Dangel** [*1] **Runa Eschenhagen** [2] **Kirill Neklyudov** [1] **Agustinus Kristiadi** [1]
Richard E. Turner [2] **Alireza Makhzani** [1 3]

Abstract

Second-order methods such as KFAC can be useful for neural net training. However, they are often memory-inefficient since their preconditioning Kronecker factors are dense, and numerically unstable in low precision as they require matrix inversion or decomposition. These limitations render such methods unpopular for modern mixed-precision training. We address them by (i) formulating an *inverse-free* KFAC update and (ii) imposing *structures* in the Kronecker factors, resulting in *structured inverse-free natural gradient descent (SINGD)*. On modern neural networks, we show that SINGD is memory-efficient and numerically robust, in contrast to KFAC, and often outperforms AdamW even in half precision. Our work closes a gap between first- and second-order methods in modern low-precision training.

1. Introduction

The continuing success of deep learning (DL) is—to a large extent—powered by scaling up computational power (Thompson et al., 2020) to increase the number of trainable neural network (NN) parameters. Contemporary natural language processing (Radford et al., 2019; Brown et al., 2020; Touvron et al., 2023) and computer vision (Dehghani et al., 2023) models often consist of billions of parameters, and will likely grow further in the future. To compensate for increasing computational demands, many training pipelines use lower precision data types (Micikevicius et al., 2018) and memory-efficient first-order optimizers like SGD (Robbins & Monro, 1951) or Adam(W) (Kingma & Ba, 2015; Loshchilov & Hutter, 2019).

Second-order methods, like natural gradient descent (NGD, Amari, 1998), leverage curvature information which has many applications in DL: It is useful for improving training dynamics (Martens & Grosse, 2015; Osawa et al., 2023), pruning (Wang et al., 2019), understanding the influence of training examples (Bae et al., 2022), and uncertainty estimation (Zhang et al., 2018; Immer et al., 2021; Daxberger et al., 2021). One major obstacle why those methods are rarely used is their higher memory consumption and iteration cost.

The perhaps most common concept to scale second-order methods for DL is Kronecker-factored approximate curvature (KFAC, Heskes, 2000; Martens & Grosse, 2015) which approximates the Fisher's block diagonals via Kronecker products. The KFAC optimizer built on top of this curvature approximation, and its variants such as George et al. (2018) show promising results for medium-sized NNs (e.g. Osawa et al., 2023), its usefulness is often limited by (i) memory consumption, and (ii) the use of low-precision floating-point (FP) training that renders matrix decompositions/inversions required to pre-condition the gradient numerically unstable.

Recently, Lin et al. (2023) proposed an inverse-free Kronecker-factored natural gradient descent (INGD) algorithm that replaces matrix inversion with subtraction in a matrix logarithm space. Their update is purely based on matrix multiplications and therefore numerically stable in single-precision (FP-32); however, it is unclear whether this extends to half-precision (BFP-16). Furthermore, INGD has not been derived from the popular natural gradient approaches for DL. It is unclear if and how the method is connected to the predominant KFAC optimizer. Also, INGD does not improve over KFAC's memory complexity since its Kronecker factors are dense matrices of the same size. And lastly, INGD has only been tested on convolution-based models and it is unclear whether it is useful for training modern transformer-based architectures (Vaswani et al., 2017).

Here, we extend INGD to lower its computational cost and theoretically resolve its connection to other approximate NGD methods for DL (overview in Figure 2): First, we show that a special case of INGD recovers the KFAC method. This allows us to effectively perform KFAC updates in an *inverse-free* fashion. We call this modification of INGD *inverse-free KFAC (IKFAC)*. Second, we exploit an alge-

[*]Equal contribution [1]Vector Institute [2]University of Cambridge [3]University of Toronto. Correspondence to: Wu Lin <yorker.lin@gmail.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

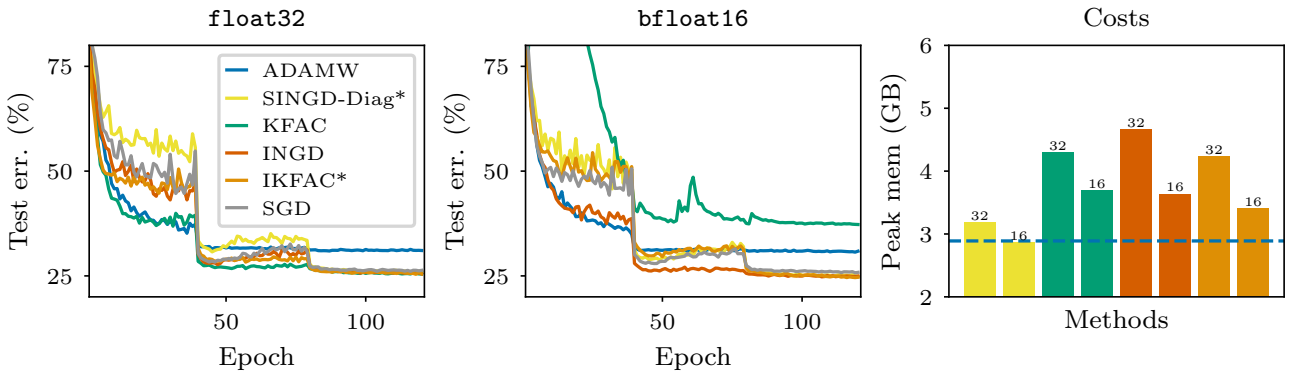

Figure 1: CIFAR-100 experiments on VGG net. *Left/Center:* Our methods (IKFAC and SINGD) outperform AdamW and perform stably in FP-32 *and* BFP-16—unlike KFAC—as they do not require matrix inversions. IKFAC effectively performs KFAC updates and achieves similar performance in FP-32. For this task, replacing the dense Kronecker factors (INGD = SINGD-Dense) with diagonal ones (SINGD-Diag) does not harm performance while reducing cost. *Right:* Memory consumption. Removing Riemannian momentum (IKFAC) or using structured Kronecker factors (SINGD-Diag) reduces INGD's memory in FP-32 and BFP-16. In BFP-16, SINGD-Diag achieves AdamW's memory consumption (dashed line).

braic structure in the matrix logarithm space and propose structure-preserving updates to maintain sparse structures on Kronecker factors. This significantly reduces memory and leads to a novel, scalable second-order optimization algorithm we call *structured inverse-free natural gradient descent (SINGD)* which contains INGD and IKFAC as special cases. We evaluate SINGD on convolution- and transformer-based models and show that it can (i) outperform SGD and AdamW while using as little memory as the latter thanks to structured Kronecker factors and (ii) yield better performance than KFAC while being stable in half-precision:

(a) We bridge the gap between INGD (Lin et al., 2023) and the original KFAC (Martens & Grosse, 2015), whose matrix inversions are unstable in low precision. Thereby, we effectively make KFAC inverse-free and amenable to low-precision training (Figure 1, *left/center*).

(b) We impose various structures (block-diagonal, low-rank, Toeplitz, hierarchical) on INGD's Kronecker factors, allowing them to be sparse to lower the memory consumption and run time (Figure 1, *right* and Table 1). Unlike many existing second-order methods tailored to a form of structure, our proposed update rule (Figure 4) is unified, efficient, and inverse-free for a range of structures. We analyze the impact of structures on downstream performance and find that structures with considerably lower memory consumption (even lower than AdamW) can yield competitive performance.

(c) Unlike other second-order methods, we show that SINGD can stably train a range of modern architectures (transformers, CNNs, GNNs) in BFP-16. In contrast to first-order methods which are often useful in narrower scopes (SGD is best for CNNs, AdamW is best for transformers), SINGD works well and outperforms SGD and AdamW in many cases (see Section 4).

Table 1: Training times and memory consumption for the optimizers shown in Figure 1 (parenthesized values are normalized relative to SGD; our methods are marked with an asterisk). INGD has 80 % time and 30 % memory overhead compared to SGD. In contrast, our SINGD-Diag only has 30 % time and 2 % memory overhead. This means that by using structures we can reduce INGD's time overhead by more than half, and basically eliminate its memory overhead compared to first-order competitors.

Method	Peak memory [GiB]	Training time [min]
SGD (BFP-16)	2.63 (1.00 x)	18.5 (1.00 x)
AdamW (BFP-16)	2.69 (1.02 x)	19.7 (1.07 x)
SINGD-Diag* (BFP-16)	2.67 (1.02 x)	23.8 (1.29 x)
IKFAC* (BFP-16)	3.18 (1.21 x)	34.0 (1.84 x)
INGD (BFP-16)	3.39 (1.29 x)	34.1 (1.84 x)
KFAC (FP-32)	4.00 (1.52 x)	83.2 (4.49 x)

Our work closes a gap between first- and second-order methods in modern low precision neural network training[1].

2. Preliminaries

We first introduce the necessary ingredients to establish a connection between INGD and KFAC, which are derived from different perspectives. We start by describing Newton's method since both methods can be seen as approxi-

[1]PyTorch implementation: `github.com/f-dangel/singd`

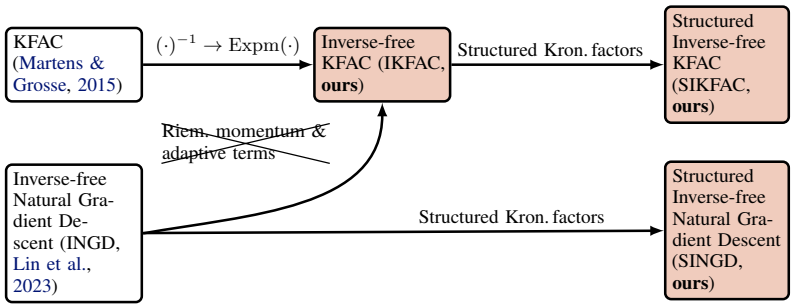

Figure 2: Existing methods and their relation to our proposed methods. IKFAC behaves like KFAC (Theorem 1), but is numerically stable in low precision. In contrast to IKFAC, INGD has Riemannian momenta and adaptive damping and curvature, which can yield better performance in practice (Section 4). INGD is equivalent to SINGD with unstructured Kronecker factors (SINGD-Dense). Structured Kronecker factors reduce memory and computational cost.

mate Newton methods using NGD. NN training often corresponds to an unconstrained minimization problem. Consider training a NN for image classification. Given a set of N examples $\{y_i, \mathbf{x}_i\}_{i=1}^N$ with labels y_i and images $\mathbf{x}_i$, the optimization problem is

$$\min_{\boldsymbol{\mu}} \ell(\boldsymbol{\mu}; \mathbf{y}, \mathbf{X}) \coloneqq \min_{\boldsymbol{\mu}} \textstyle\sum_{i=1}^N c(y_i, f(\boldsymbol{\mu}; \mathbf{x}_i)) \,, \tag{1}$$

where $\mathbf{y} \coloneqq (y_1, \dots, y_N)$, $\mathbf{X} \coloneqq (\mathbf{x}_1, \dots, \mathbf{x}_N)$, and $\hat{y}_i \coloneqq f(\boldsymbol{\mu}; \mathbf{x}_i)$ is a NN that outputs a predicted label $\hat{y}_i$ for an image $\mathbf{x}_i$. Parameters $\boldsymbol{\mu}$ denote learnable weights of the NN and $c(y_i, \hat{y}_i)$ is a differentiable loss function to measure the difference between a true label y_i and a predicted label $\hat{y}_i$. To solve Equation (1), Newton's method follows the update

$$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \mathbf{S}^{-1} \left(\nabla_\mu \ell(\boldsymbol{\mu}; \mathbf{y}, \mathbf{X})\right) \,, \tag{2}$$

where $\mathbf{S} \coloneqq \nabla_\mu^2 \ell(\boldsymbol{\mu}; \mathbf{y}, \mathbf{X})$ is the Hessian of the loss.

2.1. KFAC: Approximate NGD for MLE

Computing the Hessian, as required by Newton's method, is usually intractable for NNs. NGD uses a Fisher information matrix (FIM) instead of the Hessian by reformulating problem (1) as maximum likelihood estimation (MLE) of $p(\mathbf{y} \mid \boldsymbol{\mu}, \mathbf{X}) = \prod_i p(y_i \mid \boldsymbol{\mu}, \mathbf{x}_i)$, where $p(y_i \mid \boldsymbol{\mu}, \mathbf{x}_i) \coloneqq \exp(-c(y_i, f(\boldsymbol{\mu}, \mathbf{x}_i)))$. The maximization problem $\max_\mu p(\mathbf{y} \mid \boldsymbol{\mu}, \mathbf{X})$ is equivalent to the MLE problem

$$\min_{\boldsymbol{\mu}} -\log p(\mathbf{y} \mid \boldsymbol{\mu}, \mathbf{X}) = \min_{\boldsymbol{\mu}} \ell(\boldsymbol{\mu}; \mathbf{y}, \mathbf{X}) \,. \tag{3}$$

This formulation allows to exploit additional statistical structures such as the FIM which is defined as shown below (Kunstner et al., 2019), where we assume a label y is sampled from the likelihood $p(y \mid \boldsymbol{\mu}, \mathbf{x}_i)$ given an image $\mathbf{x}_i$. With $\mathbf{s}_i(y) \coloneqq \log p(y \mid \mu, \mathbf{x}_i)$, we have

$$\begin{aligned} F(\boldsymbol{\mu}) &\coloneqq \sum_{i=1}^N \mathbb{E}_{y \sim p(y \mid \mu, x_i)} \left[\nabla_\mu \mathbf{s}_i(y) (\nabla_\mu \mathbf{s}_i(y))^\top\right] \\ &= \sum_{i=1}^N \mathbb{E}_{y \sim p(y \mid \mu, x_i)} \left[-\nabla_\mu^2 \mathbf{s}_i(y)\right] \,. \end{aligned} \tag{4}$$

For ubiquitous loss functions like the mean-squared error and cross-entropy, and more generally, many members of the exponential family with natural parameterization, the FIM coincides with the generalized Gauss-Newton (GGN) matrix (Wang, 2010; Martens, 2014), a common approximation of the Hessian in deep learning (Schraudolph, 2002; Botev et al., 2017). This relationship connects NGD to Newton's method. A common approximation of the FIM/GGN and Hessian is the so-called *empirical* Fisher $\hat{F}(\boldsymbol{\mu})$, which replaces the samples y from the model's predictive distribution in Equation (4) with the empirical data labels y_i:

$$\begin{aligned} \hat{F}(\boldsymbol{\mu}) &\coloneqq \sum_{i=1}^N \nabla_\mu \mathbf{s}_i(y_i) (\nabla_\mu \mathbf{s}_i(y_i))^\top \\ &\approx -\sum_{i=1}^N \nabla_\mu^2 \mathbf{s}_i(y_i) = \mathbf{S} \,. \end{aligned}$$

While there is no clear theoretical justification for this Hessian approximation (Kunstner et al., 2019), it simplifies the implementation, reduces cost, and has been shown to work well in practice (Graves, 2011; Osawa et al., 2019). This approximation is also known as Fisher's scoring with observed FIM for nonlinear models (Osborne, 1992; Smyth, 1996; 2015). With this, we can formulate an NGD update with the *empirical* FIM $\hat{F}(\boldsymbol{\mu})$ to approximate Newton's method as

$$\begin{aligned} \boldsymbol{\mu} &\leftarrow \boldsymbol{\mu} - \beta \left(\hat{F}(\boldsymbol{\mu})\right)^{-1} \nabla_\mu \ell(\boldsymbol{\mu}; \mathbf{y}, \mathbf{X}) \\ &\approx \boldsymbol{\mu} - \beta \mathbf{S}^{-1} \nabla_\mu \ell(\boldsymbol{\mu}; \mathbf{y}, \mathbf{X}). \end{aligned}$$

We call this update NGD for MLE.

KFAC (Heskes, 2000; Martens & Grosse, 2015) is the probably most common second-order optimizer in DL. The KFAC algorithm is based on a Kronecker-factored approximation of the Fisher, which is also sometimes referred to as KFAC. Here, we refer to the algorithm as *KFAC* or *KFAC method* and to the approximation as *Kronecker approximation*; we will consider the *empirical* Fisher's Kronecker approximation. It approximates the per-layer FIM with a Kronecker-factored block $\tilde{F}_l$ for each layer l of the net. This approximation has first been derived for linear layers, later for convolutional (Grosse & Martens, 2016) and recurrent layers (Martens et al., 2018), and recently been generalized to all linear layers that use weight sharing (Eschenhagen et al., 2023), e.g. graph neural networks

and transformers. A block is given by $\tilde{F}_l(\boldsymbol{\mu}) \coloneqq \mathbf{U}_l \otimes \mathbf{G}_l$, with $\mathbf{U}_l \coloneqq \mathbf{u}_l \mathbf{u}_l^\top \in \mathbb{R}^{d_i \times d_i}$ and $\mathbf{G}_l \coloneqq \mathbf{g}_l \mathbf{g}_l^\top \in \mathbb{R}^{d_o \times d_o}$, where $\mathbf{u}_l \in \mathbb{R}^{d_i}$ is the lth layer's input and $\mathbf{g}_l \in \mathbb{R}^{d_o}$ is the gradient of the loss w.r.t. the layer's output. We suppress the dependence on the parameters $\boldsymbol{\mu}$ and the input $\mathbf{x}_i$ and, for simplicity, assume no weight sharing. KFAC also uses exponential moving averages (β_1) over $\mathbf{U}$ and $\mathbf{G}$ (yielding $\mathbf{S}_K, \mathbf{S}_C$) and damping λ, see Figure 3.

While the Kronecker approximation enables more efficient gradient preconditioning, KFAC needs to store the dense Kronecker factors $\mathbf{S}_K$ and $\mathbf{S}_C$ and invert them at every preconditioner update. The run time overhead is usually amortized by updating the preconditioner less frequently, but this can cause instabilities, especially in low-precision settings. Second, the Kronecker factors introduce significant memory overhead, which poses issues in large models. Since low-precision training is becoming the standard norm in fields like natural language processing, these issues will become more apparent in modern DL. There are multiple numerical concerns when using KFAC or variants thereof in low precision. In PyTorch (Paszke et al., 2019) and JAX (Bradbury et al., 2018) implementations, all tensors must be casted into FP-32 as (B)FP-16 matrix inverses/decompositions are not supported. Moreover, $\mathbf{g}_l$ has to be rescaled to avoid over- or under-flows when calculating $\mathbf{G}_l$. Memory consumption has previously been addressed through diagonal or block-diagonal versions of $\mathbf{U}_l, \mathbf{G}_l$ (Zhang et al., 2018; Grosse et al., 2023). However, it is unclear if these simple structures maintain downstream performance.

2.2. INGD: Approximate NGD for Bayesian estimation

Derived from Bayesian principles, INGD (Lin et al., 2023) directly approximates the Hessian inverse. We first introduce two ingredients INGD builds on: the Bayesian learning rule (BLR, Khan & Lin, 2017; Zhang et al., 2018; Khan et al., 2018; Osawa et al., 2019; Lin et al., 2020; Khan & Rue, 2021; Tan, 2022) and an inverse-free second-order method from Lin et al. (2021). By the BLR, Newton's method to solve the MLE (3) can be seen as another natural-gradient update to solve a variational inference (VI) problem with a delta approximation (Khan & Rue, 2021). This interpretation allows to view a precision matrix in the variational problem as Hessian estimation in the MLE problem. Thus, Lin et al. (2021) suggest reparameterizing the Hessian as the precision of the Gaussian posterior in a matrix logarithm space and exploiting the parameterization invariance of natural gradients to obtain an inverse-free update.

BLR Consider a Bayesian problem formulation, where NN weights are random variables. We denote these weights by new parameters $\mathbf{w}$ since random variables are no longer learnable and use a variational Gaussian distribution to approximate the posterior over the random variables. Its mean and precision will be treated as the learnable weights $\boldsymbol{\mu}$ and the Hessian estimation $\mathbf{S}$ in Newton's step (2).

The VI problem considered in the learning rule is defined as $\min_\tau -\mathcal{L}(\boldsymbol{\tau})$ with the evidence lower bound (ELBO)

$$\mathcal{L}(\boldsymbol{\tau}) \coloneqq \mathbb{E}_{w \sim q(w|\tau)} \left[\log p(\mathbf{w}) + \log p(\mathbf{y} \mid \mathbf{w}, \mathbf{X})\right] + H_q(\boldsymbol{\tau}) \,. \tag{5}$$

$\boldsymbol{\tau} = \{\boldsymbol{\mu}, \mathbf{S}\}$ are the learnable parameters of the variational Gaussian distribution $q(\mathbf{w} \mid \boldsymbol{\tau}) = \mathcal{N}(\mathbf{w} \mid \boldsymbol{\mu}, \mathbf{S})$ with mean $\boldsymbol{\mu}$ and precision $\mathbf{S}$. The likelihood $p(\mathbf{y} \mid \mathbf{w}, \mathbf{X}) = \exp(-\ell(\mathbf{w}; \mathbf{y}, \mathbf{X}))$ takes the same form as in the MLE setting while the prior $p(\mathbf{w}) \propto \exp(-R(\mathbf{w}))$ is defined by a regularizer $R(\mathbf{w}) \geq 0$. To recover the MLE problem, we consider an uninformative prior $p(\mathbf{w})$ (i.e., $R(\mathbf{w}) = 0$). $H_q(\boldsymbol{\tau}) \coloneqq \mathbb{E}_{w \sim q}\left[-\log q\right]$ is the entropy of $q(\mathbf{w} \mid \boldsymbol{\tau})$.

Similar to the MLE case, the Bayesian formulation allows to exploit additional statistical structures in form of another FIM, which is that of the variational Gaussian defined as

$$\begin{aligned} F(\boldsymbol{\tau}) &\coloneqq \mathbb{E}_{w \sim q(w|\tau)} \left[\nabla_\tau \log q(\mathbf{w} \mid \boldsymbol{\tau}) \nabla_\tau^\top \log q(\mathbf{w} \mid \boldsymbol{\tau})\right] \\ &= -\mathbb{E}_{w \sim q}\left[\nabla_\tau^2 \log q(\mathbf{w} \mid \boldsymbol{\tau})\right] \,, \end{aligned}$$

and has a closed-form expression. This FIM should *not* be confused with the FIM used for MLE (4).

Under the BLR, we perform NGD updates not only on $\boldsymbol{\mu}$ but also on $\mathbf{S}$. Khan & Rue (2021) formulate a step with the *exact* FIM $F(\boldsymbol{\tau})$ and stepsize $\beta > 0$ to update $\boldsymbol{\tau} = \{\boldsymbol{\mu}, \mathbf{S}\}$,

$$\boldsymbol{\tau} \leftarrow \boldsymbol{\tau} - \beta \Big(F(\boldsymbol{\tau})\Big)^{-1} \nabla_\tau \left(-\mathcal{L}(\boldsymbol{\tau})\right) \,.$$

This is the NGD update for BLR, vis-à-vis for MLE. Following Khan & Nielsen (2018), the update simplifies to

$$\begin{aligned} \mathbf{S} &\leftarrow (1-\beta)\mathbf{S} + \beta \mathbb{E}_{w \sim q(w|\mu,S)} \left[\nabla_w^2 \ell(\mathbf{w}; \mathbf{y}, \mathbf{X})\right] \,, \\ \boldsymbol{\mu} &\leftarrow \boldsymbol{\mu} - \beta \mathbf{S}^{-1} \mathbb{E}_{w \sim q(w|\mu,S)} \left[\nabla_w \ell(\mathbf{w}; \mathbf{y}, \mathbf{X})\right] \,. \end{aligned}$$

Further simplifying expectations with a delta approximation (highlighted in red) at mean $\boldsymbol{\mu}$, we obtain

$$\begin{aligned} \mathbf{S} &\leftarrow (1-\beta)\mathbf{S} + \beta \nabla_\mu^2 \ell(\boldsymbol{\mu}; \mathbf{y}, \mathbf{X}) \,, \\ \boldsymbol{\mu} &\leftarrow \boldsymbol{\mu} - \beta \mathbf{S}^{-1} \nabla_\mu \ell(\boldsymbol{\mu}; \mathbf{y}, \mathbf{X}) \,. \end{aligned}$$

which recovers Newton's method in (2) for $\beta = 1$.

Removing inversion Lin et al. (2021) reparameterize the precision matrix $\mathbf{S}$ in a matrix logarithm space and perform natural gradient updates in this space, which transforms inversion into subtraction. One can go back directly to the original space, without explicitly inverting a matrix, via a truncated matrix exponential. The method is inverse-free and, since NGs are parameterization invariant, Newton-like.

KFAC (Martens & Grosse, 2015)

Each T iters, update $\mathbf{S}_K, \mathbf{S}_C$
Obtain $\mathbf{U} \otimes \mathbf{G}$ to approximate $\nabla^2_\mu \ell(\boldsymbol{\mu})$
$\mathbf{S}_K \leftarrow (1-\beta_1)\mathbf{S}_K + \beta_1 \mathbf{U}$
$\mathbf{S}_C \leftarrow (1-\beta_1)\mathbf{S}_C + \beta_1 \mathbf{G}$
$\mathbf{S}_K^{-1} \leftarrow (\mathbf{S}_K + \lambda \mathbf{I}_{d_i})^{-1}$
$\mathbf{S}_C^{-1} \leftarrow (\mathbf{S}_C + \lambda \mathbf{I}_{d_o})^{-1}$
$\mathbf{m}_\mu \leftarrow \alpha_2 \mathbf{m}_\mu + \mathbf{S}_C^{-1} \text{vec}^{-1}(\mathbf{g}) \mathbf{S}_K^{-1} + \gamma \text{vec}^{-1}(\boldsymbol{\mu})$
$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \beta_2 \text{vec}(\mathbf{m}_\mu)$

IKFAC (ours)

Each T iters, update $\mathbf{m}_K, \mathbf{m}_C, \mathbf{K}, \mathbf{C}$
Obtain $\mathbf{U} \otimes \mathbf{G}$ to approximate $\nabla^2_\mu \ell(\boldsymbol{\mu})$
$\mathbf{m}_K \leftarrow 0\mathbf{m}_K + \frac{1}{2d_o}(d_o \mathbf{H}_K + \lambda d_o \mathbf{K}^\top \mathbf{K} - d_o \mathbf{I}_{d_i})$
$\mathbf{m}_C \leftarrow 0\mathbf{m}_C + \frac{1}{2d_i}(d_i \mathbf{H}_C + \lambda d_i \mathbf{C}^\top \mathbf{C} - d_i \mathbf{I}_{d_o})$
$\mathbf{K} \leftarrow \mathbf{K}(\mathbf{I}_{d_i} - \beta_1 \mathbf{m}_K)$
$\mathbf{C} \leftarrow \mathbf{C}(\mathbf{I}_{d_o} - \beta_1 \mathbf{m}_C)$
$\mathbf{m}_\mu \leftarrow \alpha_2 \mathbf{m}_\mu + \mathbf{C}\mathbf{C}^\top \text{vec}^{-1}(\mathbf{g}) \mathbf{K}\mathbf{K}^\top + \gamma \text{vec}^{-1}(\boldsymbol{\mu})$
$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \beta_2 \text{vec}(\mathbf{m}_\mu)$

Figure 3: Comparison between KFAC and IKFAC update for one weight matrix $\text{vec}^{-1}(\boldsymbol{\mu}) \in \mathbb{R}^{d_o \times d_i}$. The flattened gradient is $\mathbf{g} \coloneqq \nabla_\mu \ell(\boldsymbol{\mu}) \in \mathbb{R}^{d_o d_i}$ and $\text{vec}^{-1}(\mathbf{g}) \in \mathbb{R}^{d_o \times d_i}$ is its matrix reshape. IKFAC uses $\mathbf{H}_K \coloneqq \mathbf{K}^\top \mathbf{U} \mathbf{K}$ and $\mathbf{H}_C \coloneqq \mathbf{C}^\top \mathbf{G} \mathbf{C}$ to incorporate the Kronecker curvature $\mathbf{U}$ and $\mathbf{G}$. Both methods use momentum buffers $\mathbf{m}_\mu$ for the weight-decayed update direction with momentum α_2 and weight decay γ, and a learning rate β_2 for the parameter update. (*Left*) KFAC uses an exponentially moving average with decay $1 - \beta_1$ to accumulate the Kronecker factors and applies a damping term $\lambda \mathbf{I}$ before inversion to handle potential singularities in $\mathbf{S}_K, \mathbf{S}_C$. (*Right*) In contrast to KFAC, IKFAC directly approximates $(\mathbf{S}_K + \lambda \mathbf{I})^{-1}$ and $(\mathbf{S}_C + \lambda \mathbf{I})^{-1}$ by $\mathbf{K}\mathbf{K}^\top$ and $\mathbf{C}\mathbf{C}^\top$. The pre-conditioner update is a modification of INGD (Lin et al., 2023) and the changes—zero Riemannian momentum, and non-adaptive damping and curvature—are highlighted in red.

INGD (Lin et al., 2023)

Each T iterations, update $\mathbf{m}_K, \mathbf{m}_C, \mathbf{K}, \mathbf{C}$
Obtain $\mathbf{U} \otimes \mathbf{G}$ to approximate $\nabla^2_\mu \ell(\boldsymbol{\mu})$
$\mathbf{m}_K \leftarrow \alpha_1 \mathbf{m}_K + \frac{1}{2d_o}(\text{Tr}(\mathbf{H}_C)\mathbf{H}_K + c^2 \mathbf{K}^\top \mathbf{K} - d_o \mathbf{I}_{d_i})$
$\mathbf{m}_C \leftarrow \alpha_1 \mathbf{m}_C + \frac{1}{2d_i}(\text{Tr}(\mathbf{H}_K)\mathbf{H}_C + \kappa^2 \mathbf{C}^\top \mathbf{C} - d_i \mathbf{I}_{d_o})$
$\mathbf{K} \leftarrow \mathbf{K}(\mathbf{I}_{d_i} - \beta_1 \mathbf{m}_K)$
$\mathbf{C} \leftarrow \mathbf{C}(\mathbf{I}_{d_o} - \beta_1 \mathbf{m}_C)$
$\mathbf{m}_\mu \leftarrow \alpha_2 \mathbf{m}_\mu + \mathbf{C}\mathbf{C}^\top \text{vec}^{-1}(\mathbf{g}) \mathbf{K}\mathbf{K}^\top + \gamma \text{vec}^{-1}(\boldsymbol{\mu})$
$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \beta_2 \text{vec}(\mathbf{m}_\mu)$

SINGD (ours)

Each T iterations, update $\hat{\mathcal{L}}_{m_K}, \hat{\mathcal{L}}_{m_C}, \hat{\mathcal{L}}_K, \hat{\mathcal{L}}_C$
Obtain $\mathbf{U} \otimes \mathbf{G}$ to approximate $\nabla^2_\mu \ell(\boldsymbol{\mu})$
$\hat{\mathcal{L}}_{m_K} \leftarrow \alpha_1 \hat{\mathcal{L}}_{m_K} + \frac{1}{2d_o} \hat{\Pi}_K(\text{Tr}(\mathbf{H}_{\hat{\mathcal{L}}_C})\mathbf{H}_{\hat{\mathcal{L}}_K} + c^2 (\hat{\mathcal{L}}_K)^\top \hat{\mathcal{L}}_K - d_o \mathbf{I}_{d_i})$
$\hat{\mathcal{L}}_{m_C} \leftarrow \alpha_1 \hat{\mathcal{L}}_{\mathbf{m}_C} + \frac{1}{2d_i} \hat{\Pi}_C(\text{Tr}(\mathbf{H}_{\hat{\mathcal{L}}_K})\mathbf{H}_{\hat{\mathcal{L}}_C} + \kappa^2 (\hat{\mathcal{L}}_C)^\top \hat{\mathcal{L}}_C - d_i \mathbf{I}_{d_o})$
$\hat{\mathcal{L}}_K \leftarrow \hat{\mathcal{L}}_K(\mathbf{I}_{d_i} - \beta_1 \hat{\mathcal{L}}_{m_K})$
$\hat{\mathcal{L}}_C \leftarrow \hat{\mathcal{L}}_C(\mathbf{I}_{d_o} - \beta_1 \hat{\mathcal{L}}_{m_C})$
$\mathbf{m}_\mu \leftarrow \alpha_2 \mathbf{m}_\mu + \hat{\mathcal{L}}_C(\hat{\mathcal{L}}_C)^\top \text{vec}^{-1}(\mathbf{g}) \hat{\mathcal{L}}_K(\hat{\mathcal{L}}_K)^\top + \gamma \text{vec}^{-1}(\boldsymbol{\mu})$
$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \beta_2 \text{vec}(\mathbf{m}_\mu)$

Figure 4: Comparison of a single weight matrix's update between INGD and our extension—SINGD—via structured Kronecker factors. (*Left*) INGD features Riemannian momentum (α_1), adaptive curvature ($\text{Tr}(\mathbf{H}_C)$, $\text{Tr}(\mathbf{H}_K)$), adaptive damping ($c^2 \coloneqq \lambda \text{Tr}(\mathbf{C}^\top \mathbf{C})$, $\kappa^2 \coloneqq \lambda \text{Tr}(\mathbf{K}^\top \mathbf{K})$), and correlated updates of $\mathbf{K}$ and $\mathbf{C}$ ($\mathbf{m}_K$, $\mathbf{m}_C$). The pre-conditioner matrices are updated with a learning rate β_1, and the optimizer keeps a momentum buffer on the weight-decayed update with momentum α_2 and weight decay γ. The learning rate for the parameters is β_2. (*Right*) SINGD's update is similar but each Kronecker factor and its momentum ($\bullet$) is replaced by its structured version ($\hat{\mathcal{L}}_\bullet$, e.g. (block-)diagonal); likewise in the computation of c^2, κ^2, $\mathbf{H}_K$, and $\mathbf{H}_C$. When updating the momenta, their structure is preserved through a subspace projection map $\hat{\Pi}_\bullet(\,\cdot\,)$ that restores $\hat{\mathcal{L}}_\bullet$'s structure from a dense symmetric matrix $\cdot$ (e.g. taking the (block) diagonal). Importantly, we can efficiently compute the extraction map without expanding its argument in dense form, which reduces memory and run time. The extension of IKFAC to SIKFAC is analogous. One of the notable elements of INGD and SINGD is that they are scale invariant to the choice of the Kronecker approximation (see Appendix E) as the approximation is not unique.

The first step is to express the precision matrix $\mathbf{S}$ using a non-singular square matrix $\mathbf{A}$ as $\mathbf{S} = \mathbf{A}^{-\top}\mathbf{A}^{-1}$ and perform a natural gradient step using the exact FIM in a tangent space (denoted by $\mathbf{M}$) of $\mathbf{A}_t$ at iteration t. We then construct a new map as $\mathbf{A} \coloneqq \boldsymbol{\phi}(\mathbf{A}_t, \mathbf{M}) \coloneqq \mathbf{A}_t \text{Expm}(1/2 \mathbf{M})$ using both the current point $\mathbf{A}_t$ and $\mathbf{M}$ as input, where $\text{Expm}(\mathbf{N}) = \mathbf{I} + \sum_{j=1}^\infty \mathbf{N}^j / j!$ is the matrix exponential. Observe that $\mathbf{M}$ stays in a matrix logarithm space. At each iteration t, we use a new matrix logarithm space associated to $\mathbf{A}_t$ and generate a new origin $\mathbf{M}_0 = \mathbf{0}$ in this space to represent $\mathbf{A}_t$ since $\mathbf{A}_t \equiv \boldsymbol{\phi}(\mathbf{A}_t, \mathbf{0}) = \mathbf{A}_t \text{Expm}(1/2 \mathbf{M}_0)$. The map $\boldsymbol{\phi}$ is a *local reparameterization* map that takes not only $\mathbf{M}$ but also $\mathbf{A}_t$ as input. Thanks to this map, the Fisher block is *locally orthonormalized* (Lin et al., 2023) at origin $\mathbf{M}_0$. Since we used the origin to represent $\mathbf{A}_t$ in the local coordinate $\mathbf{M}$, a natural gradient step becomes a (Euclidean) gradient step in the space of $\mathbf{M}$, which makes it easy to add Riemannian momentum (Lin et al., 2023) into the structured positive-definite matrix $\mathbf{S}$. This allows to perform updates in the logarithmic space of $\mathbf{M}$ and avoid matrix inversions:

$$\begin{aligned} \mathbf{M} &\leftarrow \mathbf{M}_0 - \beta \mathbf{N}\,, \\ \boldsymbol{\mu} &\leftarrow \boldsymbol{\mu} - \beta \mathbf{A}_{t+1}\mathbf{A}_{t+1}^\top \nabla_\mu \ell(\boldsymbol{\mu}; \mathbf{y}, \mathbf{X})\,, \end{aligned} \tag{6}$$

where $\mathbf{A}_{t+1} \coloneqq \boldsymbol{\phi}(\mathbf{A}_t, \mathbf{M}) = \mathbf{A}_t \text{Expm}\,(1/2 \mathbf{M})$ and $\mathbf{N} \coloneqq \mathbf{A}_t^\top \nabla^2_\mu \ell(\boldsymbol{\mu}; \mathbf{y}, \mathbf{X}) \mathbf{A}_t - \mathbf{I}$. Equation (6) is a Newton-like update without matrix inverse. To see that, we can reexpress the update of $\mathbf{A}$ in terms of $\mathbf{S}$ and use properties of the

matrix exponential function,

$$\begin{aligned}\mathbf{S}_{t+1} &= \mathbf{A}_{t+1}^{-T}\mathbf{A}_{t+1}^{-1} = \mathbf{A}_t^{-T}\text{Expm}\left(\beta\mathbf{N}\right)\mathbf{A}_t^{-1} \\ &= (1-\beta)\mathbf{S}_t + \beta\nabla_\mu^2\ell(\boldsymbol{\mu};\mathbf{y},\mathbf{X}) + O(\beta^2).\end{aligned}$$

Next, we can construct a structured precision matrix $\mathbf{S}$ as a structured Hessian estimation using a sparse non-singular matrix $\mathbf{A}$. As we will discuss in Section 3.2, it is essential to update $\mathbf{M}$ to preserve sparsity in $\mathbf{A}$. The space of $\mathbf{M}$ as a tangent/logarithm space of $\mathbf{A}$ allows us to efficiently impose sparse structures on $\mathbf{A}$ without requiring the Hessian $\nabla_\mu^2\ell(\boldsymbol{\mu};\mathbf{y},\mathbf{X})$ or a Hessian approximation to be sparse or structured. This is different from another inverse-free method (Tan, 2022) that considers directly performing NGD updates of $\mathbf{A}$ instead of $\mathbf{M}$, where $\mathbf{A}$ must be restricted to a (triangular) Cholesky factor. This does not preserve sparsity in $\mathbf{A}$ unless the Hessian or its approximation admit a special structure, which is usually not the case in DL problems.

INGD Our work is built on INGD (Figure 4) where $\mathbf{A} = \mathbf{K}\otimes\mathbf{C}$ is factorized into two Kronecker factors. The exact FIM under this parameterization is singular due to a correlation between $\mathbf{K}$ and $\mathbf{C}$: the Kronecker factorization is not unique. Lin et al. (2023) propose a (non-singular) block-diagonal approximated FIM by ignoring the correlation in the original FIM and perform NGD with this block-diagonal FIM on tangent spaces of the factors. Riemannian momentum is further introduced in the update of $\mathbf{K}$ and $\mathbf{C}$. They use the Kronecker approximation discussed in Section 2.1 to approximate the Hessian $\nabla_\mu^2\ell(\boldsymbol{\mu};\mathbf{y},\mathbf{X})$ and truncate the matrix exponential to obtain a purely matrix-multiplication based update scheme. It is unclear how INGD is related to KFAC which uses another Kronecker factorization $\mathbf{S} = \mathbf{S}_K\otimes\mathbf{S}_C$. INGD also remains memory-inefficient due to the use of dense Kronecker factors. The authors only consider and evaluate it on convolution-based models in single precision. It remains unclear whether INGD is useful to train transformer-based models, and in half-precision.

3. Structured inverse-free NGD

Inspired by INGD, we propose an inverse-free KFAC update as a specific setting of INGD to address KFAC's numerical instability in low precision. We show that this scheme effectively recovers KFAC. We then address the memory inefficiency of KFAC and INGD for training transformer-based models by extending INGD with structures.

3.1. Inverse-free KFAC Updates for Numerical Stability

We first propose a new inverse-free update to mimic the behavior of the KFAC update; we call this update IKFAC. We then show that IKFAC corresponds to a specific setting of INGD. This bridges the gap between INGD and KFAC and sheds light on the difference between both methods.

Inspired by INGD, we replace matrix inversion with matrix subtraction in a matrix logarithm space, then go back to the original space without explicitly inverting any matrix using a truncated matrix exponential map. The IKFAC update is related to the KFAC update as we will use $\mathbf{K}\mathbf{K}^\top$ and $\mathbf{C}\mathbf{C}^\top$ to approximate the inverse Kronecker factors $\left(\mathbf{S}_K+\lambda\mathbf{I}\right)^{-1}$ and $\left(\mathbf{S}_C+\lambda\mathbf{I}\right)^{-1}$ in KFAC, respectively. We propose the following IKFAC update with learning rate β_1 for $\mathbf{K}$ and $\mathbf{C}$ using a truncated matrix exponential

$$\begin{aligned}\mathbf{K}^{\text{new}} &\leftarrow \mathbf{K}\left(\mathbf{I} - {}^{\beta_1}\!/_2\mathbf{m}_K\right), \\ \mathbf{C}^{\text{new}} &\leftarrow \mathbf{C}\left(\mathbf{I} - {}^{\beta_1}\!/_2\mathbf{m}_C\right),\end{aligned} \tag{7}$$

where $\mathbf{H}_K \coloneqq \mathbf{K}^\top\mathbf{U}\mathbf{K}$, $\mathbf{H}_C \coloneqq \mathbf{C}^\top\mathbf{G}\mathbf{C}$, $\mathbf{m}_K \coloneqq \mathbf{H}_K + \lambda\mathbf{K}^\top\mathbf{K} - \mathbf{I}$, $\mathbf{m}_C \coloneqq \mathbf{H}_C + \lambda\mathbf{C}^\top\mathbf{C} - \mathbf{I}$. This update is inverse- and matrix-decomposition-free. Since we truncate the matrix exponential $\text{Expm}(-{}^{\beta_1}\!/_2\mathbf{m}_K) \approx (\mathbf{I} - {}^{\beta_1}\!/_2\,\mathbf{m}_K)$, $\mathbf{m}_K$ indeed stays in a matrix logarithm space (see Appendix C). The logarithm space allows to impose structural constraints on $\mathbf{K}$ we discuss in Section 3.2.

The following theorem—proof in Appendix D—formally shows that $\mathbf{K}\mathbf{K}^\top$ used in IKFAC is an approximation of $(\mathbf{S}_K+\lambda\mathbf{I})^{-1}$ in KFAC at every step even with a truncated matrix exponential. Similarly, $\mathbf{C}\mathbf{C}^\top$ is an approximation of $(\mathbf{S}_C+\lambda\mathbf{I})^{-1}$. Thus, IKFAC effectively recovers KFAC up to a first-order accuracy.

Theorem 1. If $\mathbf{K}$ is updated according to the IKFAC scheme (Figure 3) with the truncation of the matrix exponential and these two updates use the same initialization and the same sequence of curvature matrices $\mathbf{U}$, then the product $\mathbf{K}\mathbf{K}^\top$ has a first-order accuracy of the KFAC update of $\left(\mathbf{S}_K+\lambda\mathbf{I}\right)^{-1}$ at each iteration, i.e., $\mathbf{K}\mathbf{K}^\top = \left(\mathbf{S}_K+\lambda\mathbf{I}\right)^{-1} + O(\beta_1^2)$.

Theorem 1 trivially extends to diagonal and block-diagonal structures. I.e., KFAC with diagonal or block-diagonal Kronecker factors is equivalent to IKFAC with diagonal or block-diagonal structure up to first order in β_1.

Now, we show that IKFAC is a specific case of INGD, whose update of $\mathbf{K}$ without Riemannian momentum ($\alpha_1 = 0$) is

$$\mathbf{K}^{\text{new}} \leftarrow \mathbf{K}\left[\mathbf{I}_{d_i} - \frac{\beta_1}{2d_o}\left(\text{Tr}(\mathbf{H}_C)\mathbf{H}_K + \lambda\text{Tr}(\mathbf{C}^\top\mathbf{C})\mathbf{K}^\top\mathbf{K} - d_o\mathbf{I}_{d_i}\right)\right] \tag{8}$$

Since $\text{Tr}(\mathbf{I}_{d_o}) = d_o$, $\mathbf{H}_C \in \mathbb{R}^{d_o\times d_o}$, $\mathbf{C} \in \mathbb{R}^{d_o\times d_o}$, and $\mathbf{K} \in \mathbb{R}^{d_i\times d_i}$, we can obtain IKFAC from INGD by simply replacing $\text{Tr}(\mathbf{H}_C)$ and $\text{Tr}(\mathbf{C}^\top\mathbf{C})$ with $\text{Tr}(\mathbf{I}_{d_o})$:

$$\mathbf{K}^{\text{new}} \leftarrow \mathbf{K}\left[\mathbf{I}_{d_i} - \frac{\beta_1}{2d_o}\left(\text{Tr}(\mathbf{I}_{d_o})\mathbf{H}_K + \lambda\text{Tr}(\mathbf{I}_{d_o})\mathbf{K}^\top\mathbf{K} - d_o\mathbf{I}_{d_i}\right)\right]. \tag{9}$$

This sheds light on the difference between both methods. In IKFAC (see Appendix C for details), $\mathbf{H}_K$ and $\lambda\mathbf{K}^\top\mathbf{K}$ are used for incorporating KFAC's curvature $\mathbf{U}$ and damping $\lambda\mathbf{I}$, respectively. In contrast, the curvature and damping are *adaptively* incorporated in INGD using $(\text{Tr}(\mathbf{H}_C)/d_o)\mathbf{H}_K$ and $(\lambda\text{Tr}(\mathbf{C}^\top\mathbf{C})/d_o)\mathbf{K}^\top\mathbf{K}$. The updates of $\mathbf{K}$ and $\mathbf{C}$ are

Table 2: Subspaces of the logarithm space and their projection maps $\hat{\Pi}(\mathbf{M})$, where $\mathbf{M}$ is a symmetric matrix. The hierarchical structure is constructed by replacing the diagonal matrix $\mathbf{D}_{22}$ in the rank-k upper-triangular structure with another rank-k triangular matrix $\begin{bmatrix}\mathbf{A}_{22} & \mathbf{0}\\ \mathbf{A}_{23} & \mathbf{A}_{33}\end{bmatrix}$ for a better approximation.

Subspace of the log (Lie-algebraic) space	Matrix Lie sub-group structure in $\mathbf{K}$	Subspace projection map $\hat{\Pi}(\mathbf{M})$
$\begin{bmatrix} a_{1,1} & 0 & \dots & 0 \\ a_{2,1} & a_{2,2} & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{d_i,1} & a_{d_i,2} & \dots & a_{d_i,d_i} \end{bmatrix}$	Lower-triangular (Tril.)	$\begin{bmatrix} m_{1,1} & 0 & \dots & 0 \\ 2m_{2,1} & m_{2,2} & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 2m_{d_i,1} & 2m_{d_i,2} & \dots & m_{d_i,d_i} \end{bmatrix}$
$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{A}_{qq} \end{bmatrix}$	(Block) Diagonal (block size k)	$\begin{bmatrix} \mathbf{M}_{11} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_{22} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{M}_{qq} \end{bmatrix}$
$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{bmatrix}$, $\mathbf{A}_{22}$ is diag., $\mathbf{A}_{11} \in \mathbb{R}^{d_2 \times d_2}$, $\mathbf{A}_{33} \in \mathbb{R}^{d_3 \times d_3}$	Hierarchical ($k := d_2 + d_3$)	$\begin{bmatrix} \mathbf{M}_{11} & 2\mathbf{M}_{12} & 2\mathbf{M}_{13} \\ \mathbf{0} & \text{Diag}(\mathbf{M}_{22}) & \mathbf{0} \\ \mathbf{0} & 2\mathbf{M}_{32} & \mathbf{M}_{33} \end{bmatrix}$
$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{0} & \mathbf{D}_{22} \end{bmatrix}$, $\mathbf{D}_{22}$ is diag., $\mathbf{A}_{11} \in \mathbb{R}^{k \times k}$	Rank-k upper-triangular	$\begin{bmatrix} \mathbf{M}_{11} & 2\mathbf{M}_{12} \\ \mathbf{0} & \text{Diag}(\mathbf{M}_{22}) \end{bmatrix}$
$\begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{(d_i-1)} \\ 0 & a_0 & a_1 & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & a_2 \\ \vdots & \ddots & \ddots & \ddots & a_1 \\ 0 & \cdots & \ddots & 0 & a_0 \end{bmatrix}$	Upper-triangular Toeplitz (Triu-Toepl.)	$\begin{bmatrix} b_0 & 2b_1 & 2b_2 & \cdots & 2b_{(d_i-1)} \\ 0 & b_0 & 2b_1 & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & 2b_2 \\ \vdots & \ddots & \ddots & \ddots & 2b_1 \\ 0 & \cdots & \cdots & 0 & b_0 \end{bmatrix}$, $b_j := \frac{1}{d_i - j} \sum_{k=1}^{d_i - j} m_{k,k+j}$

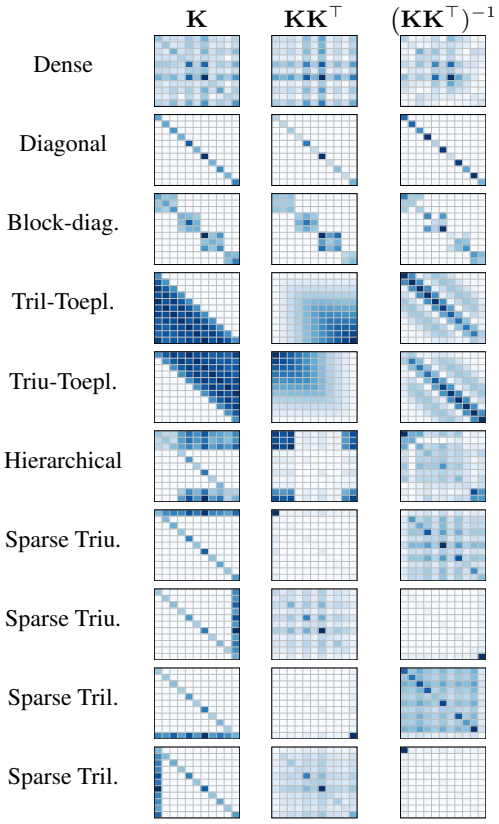

Figure 5: Illustration of structured matrices (Kronecker factors) supported by SINGD, their self-outer product (approximate inverse Hessian factor), and its inverse (approximate Hessian factor). With rank-one triangular matrices $\mathbf{K}$, we can easily impose a low-rank structure on $\mathbf{K}\mathbf{K}^\top$ or $(\mathbf{K}\mathbf{K}^\top)^{-1}$; the latter is difficult to achieve with other approaches.

correlated in INGD due to the trace terms, while $\mathbf{K}$ and $\mathbf{C}$ are updated independently in IKFAC—just like $\mathbf{S}_K$ and $\mathbf{S}_C$ in KFAC. These trace terms are needed to satisfy the orthonormalization condition of the Fisher matrix (Lin et al., 2023). They make INGD and SINGD scale-invariant to the Kronecker approximation (see Appendix E) as the approximation is not unique. In contrast, KFAC and IKFAC are not scale-invariant. The trace terms together with Riemannian momentum ($\alpha_1 > 0$) are missing in KFAC and IKFAC. Our experiments show that they can contribute to stability.

3.2. Sparse Kronecker Factors for Reducing Memory

Now, we extend INGD to reduce its memory and iteration cost. Existing sparse KFAC methods use (block-)diagonal structures for $\mathbf{S}_K$ and $\mathbf{S}_C$ (Zhang et al., 2019; Grosse et al., 2023). In contrast, we propose using sparse Kronecker factors $\mathbf{K}$ and $\mathbf{C}$ in INGD and exploiting Lie-algebraic properties in the logarithm space and algebraic sparsity of the Kronecker factors. This enables more flexible structures (Figure 5) that potentially achieve better downstream performance than (block-)diagonal structures in $\mathbf{S}_K$, $\mathbf{S}_C$.

Other related works are Lie group preconditioners (Li, 2018; 2022) derived from directly approximating the Hessian. However, these methods can be computationally expensive and numerically unstable due to sampling random weights, using Hessian-vector products (Pearlmutter, 1994), and solving linear systems that are unstable in low precision. Our approach is *sampling-free* and *inverse-free*.

We want to construct sparse factors $\mathbf{K}$ and $\mathbf{C}$ without requiring the Kronecker/Hessian approximation ($\mathbf{U} \otimes \mathbf{G}$) to be further sparse or structured. Imposing sparsity often leads to a complicated FIM which makes it difficult to perform NGD due to the FIM inversion. It is essential to update $\mathbf{m}_K$ as the logarithm space of $\mathbf{K}$ to impose sparsity on $\mathbf{K}$ as the FIM in this (moving) coordinate $\mathbf{m}_K$ is simplified and becomes an identity matrix due to the orthonormalization condition.

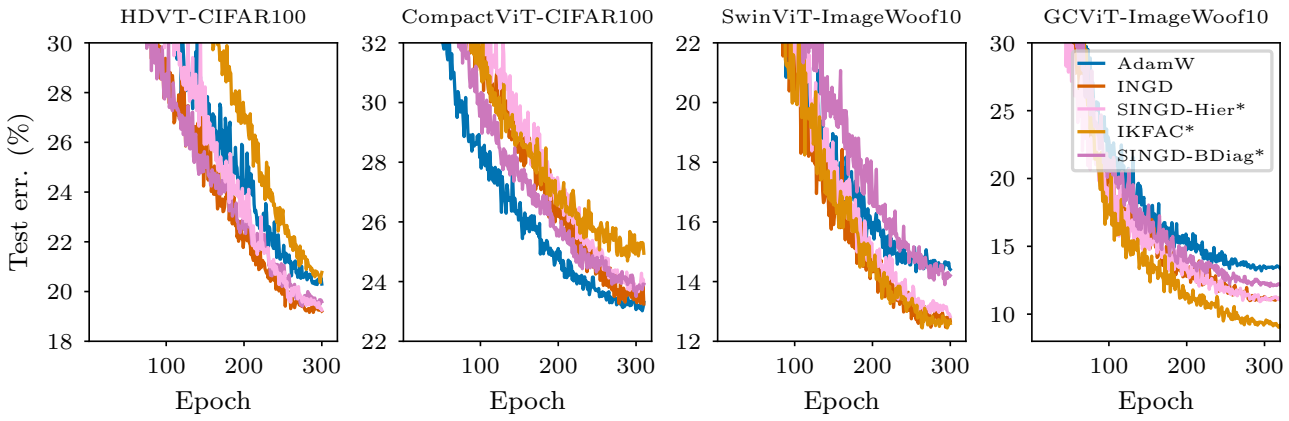

Figure 6: Test error curves for mixed-precision training in the transformer-based models with BFP-16 on datasets 'CIFAR-100' and 'ImageWoof-10'. SINGD performs as well as INGD while being memory efficient and, including IKFAC and INGD as special cases, outperforms AdamW in most of the cases. We omit KFAC since it performs unstably in BFP-16. The hierarchical structure often performs as well as the dense structure and outperforms the block-diagonal structure.

This condition (Lin et al., 2023) makes it easy for us to impose a range of sparse structures on $\mathbf{K}$ through *a unified and inverse-free update rule* (Figure 4) since we can avoid inverting the Fisher block regarding the sparse structures. We also exploit the algebraic sparsity in these structures to make our rule more efficient than INGD (Table 3).

We exploit Lie-algebraic properties in the log space of $\mathbf{m}_K$ to construct sparse structures of $\mathbf{K}$. As a general design principle, we consider structures of $\mathbf{K}$ preserved under (i) elementwise matrix operations (subtraction and scalar multiplication) and (ii) matrix multiplication, which are needed for our updates. Concretely, we construct a new local reparameterization for $\mathbf{K}$ at iteration t via

$$\mathbf{K} \coloneqq \psi(\mathbf{K}_t, \mathbf{m}_K) \coloneqq \mathbf{K}_t \text{Expm}\left(\frac{1}{\sqrt{2d_i}} \hat{\Pi}_K(\mathbf{m}_K)\right) ,$$

where $\hat{\Pi}_K(\mathbf{m}_K)$ projects the dense $\mathbf{m}_K$ to a subspace (identically for $\mathbf{C}$, but potentially using a different structure $\hat{\Pi}_C$.

Many popular structures such as tri-diagonal matrices do not satisfy our requirements as they are not closed under matrix multiplication. Moreover, it can be difficult to construct the projection map to satisfy the orthonormalization condition. One subspace structure satisfying the requirements are upper/lower triangular matrices. The subspace projection $\hat{\Pi}_K$ is a weighted extraction map since projecting the logarithm space onto a subspace is like projecting a dense square matrix onto a triangular matrix. Technically, we use

$$\mathbf{A} \coloneqq \mathbf{K}_t \text{Expm}\left(\frac{\hat{\Pi}_K(\mathbf{m}_K)}{\sqrt{2d_i}}\right) \otimes \mathbf{C}_t$$

to update $\mathbf{K}$ at iteration t, treating $\mathbf{C}_t$ and $\mathbf{K}_t$ as constants. Given a subspace $\Omega_K \subset \mathbb{R}^{d_i \times d_i}$ in the matrix logarithm space, the subspace projection map $\hat{\Pi}_K : \text{Sym}^{d_i \times d_i} \mapsto \Omega_K$ is specified by satisfying the local orthonormalization condition of the Fisher block regarding $\mathbf{m}_K$:

$$F|_{m_K=\mathbf{0}} \coloneqq -\mathbb{E}_{w\sim q}\left[\nabla^2_{m_K} \log q(\mathbf{w} \mid \boldsymbol{\mu}, \mathbf{S})\right]\big|_{m_K=\mathbf{0}} = \mathbf{I} ,$$

with the variational Gaussian $q(\mathbf{w} \mid \boldsymbol{\mu}, \mathbf{S})$ with mean $\boldsymbol{\mu}$, precision $\mathbf{S} \coloneqq \mathbf{A}^{-\top}\mathbf{A}^{-1}$ and $\text{Sym}^{d_i \times d_i}$ the set of symmetric square real matrices. Similarly, we can obtain $\hat{\Pi}_C$ for $\mathbf{C}$.

We consider several sparsities and block extensions of triangular matrices illustrated in Figure 5. E.g., the subspace projection map for a diagonal structure simply extracts diagonal entries of its input. As a non-trivial example, the subspace projection map for a lower-triangular structure extracts lower-triangular entries of its input and multiplies the entries below the main diagonal by 2. Table 2 summarizes structures and their projection maps mathematically.

Using such a subspace and its projection map, we obtain a structured INGD update (Figure 4), and similar for IKFAC. Our approach allows to use more expressive structures than the block-diagonal structure shown in Figure 5, e.g. low-rank, flexible hierarchical, and Toeplitz structures. While existing methods mainly support low-rank structures. For an efficient implementation, we only compute and store non-zero entries of $\hat{\Pi}_K(\mathbf{m}_K)$ and $\mathbf{K}$ without explicitly forming dense matrices. These structures lower not only memory consumption (Table 4), but also the iteration cost (Table 3).

4. Experiments

We evaluate SINGD on convolutional, transformer, and graph NNs, using mixed-precision training in BFP-16 with KFAC-reduce (Eschenhagen et al., 2023) and numerical tricks (Dangel, 2023) to further reduce memory consump-

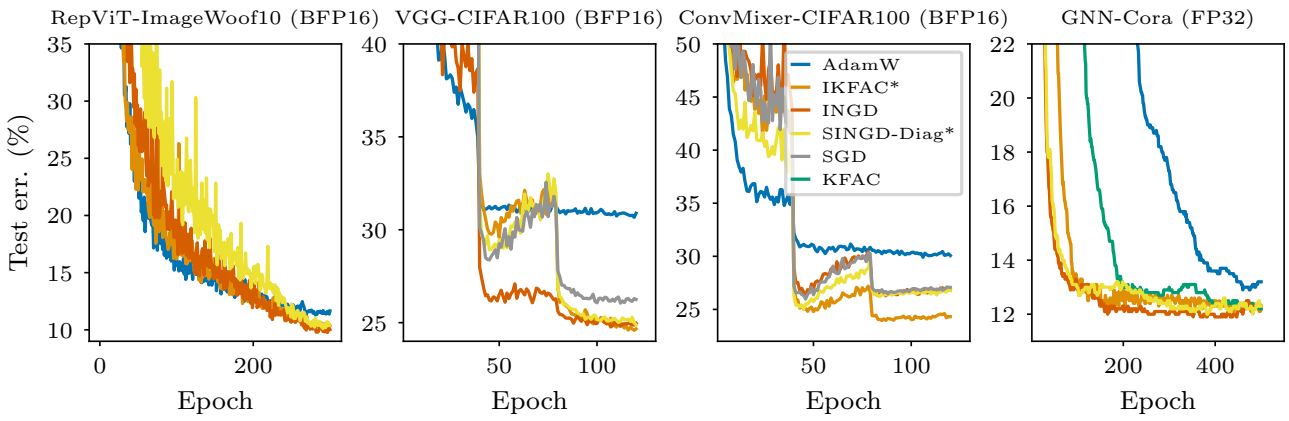

Figure 7: Test error curves for mixed-precision training in CNN and GNN models on datasets 'ImageWoof-10', 'CIFAR-100' and 'Cora'. 'Rep-ViT' is a CNN model inspired by transformers. SINGD performs as well as INGD while being memory efficient. SINGD including IKFAC and INGD as special cases, outperforms AdamW on all the models. The diagonal structure can perform as well as the dense structure on these models. KFAC only appears in the rightmost plot since it performs unstably in the other plots due to numerical issues in half-precision settings.

tion and iteration cost for convolutions. The performance metric is test error. To be memory-efficient, we consider SINGD with sparse structures such as 'diagonal', 'block-diagonal', and 'hierarchical'. We also consider IKFAC, INGD (recall SINGD with dense structure becomes INGD), and AdamW as baselines. All methods except KFAC directly support training in BFP-16. For KFAC, we have to transform a matrix into FP-32 and then transform its inverse into BFP-16. We find that KFAC performs unstably in BFP-16. For 'VGG' and 'ConvMixer', we also consider SGD as a strong baseline, We fix momentum to 0.9 and tune other hyper-parameters of each optimizer using random search. For 'VGG' and 'ConvMixer', we decrease the learning rate β_2 every 40 epochs. For 'GNN', we use a constant learning rate; all other models use a cosine learning rate schedule. We consider KFAC as a strong baseline for the GNN as suggested by Izadi et al. (2020). We train the GNN in FP-32 so that KFAC performs stably. The search space for the random search can be found in Table 5 in Appendix B.

From Figure 6 and 7, we can observe that SINGD, including IKFAC and INGD as special cases, outperforms AdamW in many cases. SINGD works well for mixed-precision training. We do not show KFAC in the plots as it performs unstably due to numerical issues. We also observe that the hierarchical structure often performs as well as the dense structure (INGD) on all the models. In several cases, the hierarchical structure outperforms the block-diagonal and diagonal structures. However, on the models shown in Figure 7, even the diagonal structure can perform as well as the dense one. Thus, we can reduce INGD's memory consumption and make SINGD as competitive as AdamW. We also train a ViT model on "ImageNet-100" to demonstrate the superior performance of SINGD over AdamW in large-scale settings (see Figure 9 in Appendix B).

5. Conclusion

We propose an inverse-free, memory-efficient natural gradient descent method—SINGD—which addresses the numerical instability and memory inefficiency of second-order methods like KFAC (Martens & Grosse, 2015). The algorithm is an extension of the inverse-free natural gradient (INGD) method from Lin et al. (2023), whose update relies only on matrix multiplications. We theoretically establish the algorithm's relation to KFAC by showing that a modification of INGD effectively performs KFAC-like updates and further improve its memory efficiency through sparse Kronecker factors. We showed that SINGD supports low-precision training and often outperforms AdamW on transformer-based models. Our work expands the scope of second-order methods to training transformer-based NNs and in low precision, making them more widely applicable.

Acknowledgements

Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring Vector Institute. Runa Eschenhagen is supported by ARM and the Cambridge Trust. Richard E. Turner is supported by Google, Amazon, ARM, Improbable and EPSRC grant EP/T005386/1.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal

consequences of our work, none which we feel must be specifically highlighted here.

References

Amari, S.-I. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Bae, J., Ng, N., Lo, A., Ghassemi, M., and Grosse, R. B. If influence functions are the answer, then what is the question? In *NeurIPS*, 2022.

Botev, A., Ritter, H., and Barber, D. Practical Gauss-Newton optimisation for deep learning. In *ICML*, 2017.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. In *NeurIPS*, 2020.

Dangel, F. Convolutions through the lens of tensor networks. arXiv 2307.02275, 2023.

Daxberger, E., Kristiadi, A., Immer, A., Eschenhagen, R., Bauer, M., and Hennig, P. Laplace redux—effortless Bayesian deep learning. In *NeurIPS*, 2021.

Dehghani, M., Djolonga, J., Mustafa, B., Padlewski, P., Heek, J., Gilmer, J., Steiner, A. P., Caron, M., Geirhos, R., Alabdulmohsin, I., et al. Scaling vision transformers to 22 billion parameters. In *ICML*, 2023.

Eschenhagen, R., Immer, A., Turner, R. E., Schneider, F., and Hennig, P. Kronecker-Factored Approximate Curvature for modern neural network architectures. In *NeurIPS*, 2023.

George, T., Laurent, C., Bouthillier, X., Ballas, N., and Vincent, P. Fast approximate natural gradient descent in a kronecker factored eigenbasis. In *NeurIPS*, 2018.

Graves, A. Practical variational inference for neural networks. In *NeurIPS*, 2011.

Grosse, R. and Martens, J. A kronecker-factored approximate fisher matrix for convolution layers. In *ICML*, 2016.

Grosse, R., Bae, J., Anil, C., Elhage, N., Tamkin, A., Tajdini, A., Steiner, B., Li, D., Durmus, E., Perez, E., et al. Studying large language model generalization with influence functions. *arXiv preprint arXiv:2308.03296*, 2023.

Hassani, A., Walton, S., Shah, N., Abuduweili, A., Li, J., and Shi, H. Escaping the big data paradigm with compact transformers. *arXiv preprint arXiv:2104.05704*, 2021.

Hatamizadeh, A., Yin, H., Heinrich, G., Kautz, J., and Molchanov, P. Global context vision transformers. In *International Conference on Machine Learning*, pp. 12633–12646. PMLR, 2023.

Heskes, T. On "natural" learning and pruning in multilayered perceptrons. *Neural Computation*, 12(4), 2000.

Immer, A., Bauer, M., Fortuin, V., Rätsch, G., and Emtiyaz, K. M. Scalable marginal likelihood estimation for model selection in deep learning. In *ICML*, 2021.

Izadi, M. R., Fang, Y., Stevenson, R., and Lin, L. Optimization of graph neural networks with natural gradient descent. In *2020 IEEE international conference on big data (big data)*, pp. 171–179. IEEE, 2020.

Khan, M. and Lin, W. Conjugate-computation variational inference: Converting variational inference in non-conjugate models to inferences in conjugate models. In *Artificial Intelligence and Statistics*, pp. 878–887, 2017.

Khan, M. E. and Nielsen, D. Fast yet Simple Natural-Gradient Descent for Variational Inference in Complex Models. *arXiv preprint arXiv:1807.04489*, 2018.

Khan, M. E. and Rue, H. The bayesian learning rule. *arXiv preprint arXiv:2107.04562*, 2021.

Khan, M. E., Nielsen, D., Tangkaratt, V., Lin, W., Gal, Y., and Srivastava, A. Fast and scalable Bayesian deep learning by weight-perturbation in Adam. In *ICML*, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Kunstner, F., Balles, L., and Hennig, P. Limitations of the empirical Fisher approximation for natural gradient descent. In *NeurIPS*, 2019.

Li, X. Black box lie group preconditioners for sgd. *arXiv preprint arXiv:2211.04422*, 2022.

Li, X.-L. Preconditioner on matrix lie group for sgd. In *International Conference on Learning Representations*, 2018.

Lin, W., Schmidt, M., and Khan, M. E. Handling the positive-definite constraint in the bayesian learning rule. In *ICML*, 2020.

Lin, W., Nielsen, F., Emtiyaz, K. M., and Schmidt, M. Tractable structured natural-gradient descent using local parameterizations. In *ICML*, 2021.

Lin, W., Duruisseaux, V., Leok, M., Nielsen, F., Khan, M. E., and Schmidt, M. Simplifying momentum-based positive-definite submanifold optimization with applications to deep learning. In *ICML*, 2023.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 10012–10022, 2021.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *ICLR*, 2019.

Lu, Z., Xie, H., Liu, C., and Zhang, Y. Bridging the gap between vision transformers and convolutional neural networks on small datasets. *Advances in Neural Information Processing Systems*, 35:14663–14677, 2022.

Martens, J. New insights and perspectives on the natural gradient method. *JMLR*, 21(146), 2014.

Martens, J. and Grosse, R. Optimizing neural networks with Kronecker-factored approximate curvature. In *ICML*, 2015.

Martens, J., Ba, J., and Johnson, M. Kronecker-factored curvature approximations for recurrent neural networks. In *ICLR*, 2018.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. Mixed precision training. In *International Conference on Learning Representations (ICLR)*, 2018.

Osawa, K., Swaroop, S., Khan, M. E. E., Jain, A., Eschenhagen, R., Turner, R. E., and Yokota, R. Practical deep learning with Bayesian principles. In *NeurIPS*, 2019.

Osawa, K., Li, S., and Hoefler, T. PipeFisher: Efficient training of large language models using pipelining and Fisher information matrices. In *MLSys*, 2023.

Osborne, M. R. Fisher's method of scoring. *International Statistical Review/Revue Internationale de Statistique*, pp. 99–117, 1992.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

Pearlmutter, B. A. Fast exact multiplication by the Hessian. *Neural Computation*, 1994.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Robbins, H. and Monro, S. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 1951.

Schraudolph, N. N. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7), 2002.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Smyth, G. K. Partitioned algorithms for maximum likelihood and other non-linear estimation. *Statistics and Computing*, 6:201–216, 1996.

Smyth, G. K. Optimization and nonlinear equations. *Statistics reference online*, 1:1–9, 2015.

Tan, L. S. Analytic natural gradient updates for cholesky factor in gaussian variational approximation. *arXiv preprint arXiv:2109.00375*, 2022.

Thompson, N. C., Greenewald, K., Lee, K., and Manso, G. F. The computational limits of deep learning. 2020.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Trockman, A. and Kolter, J. Z. Patches are all you need? *Transactions on Machine Learning Research*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *NIPS*, 2017.

Wang, A., Chen, H., Lin, Z., Pu, H., and Ding, G. Repvit: Revisiting mobile cnn from vit perspective. *arXiv preprint arXiv:2307.09283*, 2023.

Wang, C., Grosse, R., Fidler, S., and Zhang, G. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In *ICML*, 2019.

Wang, Y. Fisher scoring: An interpolation family and its Monte Carlo implementations. *Comput. Stat. Data Anal.*, 54(7), 2010.

Zhang, G., Sun, S., Duvenaud, D., and Grosse, R. Noisy natural gradient as variational inference. In *ICML*, 2018.

Zhang, G., Li, L., Nado, Z., Martens, J., Sachdeva, S., Dahl, G. E., Shallue, C. J., and Grosse, R. B. Which algorithmic choices matter at which batch sizes? Insights from a noisy quadratic model. In *NeurIPS*, 2019.

A. space and time complexity

	Method	$\triangle\boldsymbol{\mu}$ (descent direction)	Update $\mathbf{S}_K$ or $\mathbf{K}$	Update $\mathbf{S}_C$ or $\mathbf{C}$	$\nabla_\mu \ell$ (BackProp)
Iteration Cost	KFAC	$O(d_i^2 d_o + d_o^2 d_i)$	$O(\frac{1}{T}(md_i^2 + d_i^3))$	$O(\frac{1}{T}(md_o^2 + d_o^3))$	$O(md_i d_o)$
	INGD/SINGD (Dense)	$O(d_i^2 d_o + d_o^2 d_i)$	$O(\frac{1}{T}(md_i^2 + d_i^3))$	$O(\frac{1}{T}(md_o^2 + d_o^3))$	$O(md_i d_o)$
	SINGD (Block-Diag. with block size k)	$O(kd_i d_o)$	$O(\frac{1}{T}(kmd_i))$	$O(\frac{1}{T}(kmd_o))$	$O(md_i d_o)$
	SINGD (Toeplitz)	$O(d_i d_o \log(d_o d_i))$	$O(\frac{1}{T}(md_i \log d_i))$	$O(\frac{1}{T}(md_o \log d_o))$	$O(md_i d_o)$
	SINGD (Rank-1 Triangular)	$O(d_i d_o)$	$O(\frac{1}{T}(md_i))$	$O(\frac{1}{T}(md_o))$	$O(md_i d_o)$
	SINGD (Hierarchical with parameter k)	$O(kd_i d_o)$	$O(\frac{1}{T}(kmd_i))$	$O(\frac{1}{T}(kmd_o))$	$O(md_i d_o)$
	AdamW	$O(d_i d_o)$	NA	NA	$O(md_i d_o)$

Table 3: Iteration cost for a non-weight-sharing layer, where m is the size of a mini-batch and $\boldsymbol{\mu} \in \mathbb{R}^{d_i \times d_o}$ is a learnable weight matrix. We assume factors $\mathbf{K}$ and $\mathbf{C}$ use the same structure.

	Method	$\nabla_\mu \ell \odot \nabla_\mu \ell$	$\mathbf{S}_K$ or $\mathbf{K}$	$\mathbf{S}_C$ or $\mathbf{C}$
Memory Usage	KFAC	NA	$O(d_i^2)$	$O(d_o^2)$
	INGD/SINGD (Dense)	NA	$O(d_i^2)$	$O(d_o^2)$
	SINGD (Block-Diag. with block size k)	NA	$O(kd_i)$	$O(kd_o)$
	SINGD (Toeplitz)	NA	$O(d_i)$	$O(d_o)$
	SINGD (Rank-1 Triangular)	NA	$O(d_i)$	$O(d_o)$
	SINGD (Hierarchical with parameter k)	NA	$O(kd_i)$	$O(kd_o)$
	AdamW	$O(d_i d_o)$	NA	NA

Table 4: Additional Storage

B. Details of the Experiments

To demonstrate the robustness and memory efficiency of our method, we consider image classification tasks with transformer-based models such as “Compact-ViT" (Hassani et al., 2021), “Swin-ViT" (Liu et al., 2021), “GC-ViT" (Hatamizadeh et al., 2023), and “HDVT” (Lu et al., 2022). We also consider convolution-based models such as “VGG” (Simonyan & Zisserman, 2014), “ConvMixer” (Trockman & Kolter, 2023), and “Rep-ViT" (Wang et al., 2023). We train these models on datasets “CIFAR-100" and “ImageWoof-10". Note that “Rep-ViT" is a CNN model inspired by transformers while “Compact-ViT" is a data-efficient transformer using convolutional tokenization. We also consider a graph convolution model (Kipf & Welling, 2016) denoted by “GNN” for node classification on dataset “Cora". We also train a ViT model on “ImageNet-100" (`https://www.kaggle.com/datasets/ambityga/imagenet100`) to demonstrate the performance of SINGD in large-scale settings (see Fig. 9).

B.1. Hyper-parameter Tuning

Hyperparameter	Meaning	KFAC/IKFAC/SINGD in Figure 4 and 8	AdamW in Figure 8
β_2	Standard stepsize	Tuned	Tuned
α_2	Standard momentum weight	0.9	0.9
γ	(L2) weight decay	Tuned	Tuned
λ	Damping	Tuned	Tuned
β_1	Stepsize for preconditioner	Tuned	Tuned
α_1	Riemannian Momentum	(SINGD only) Tuned	NA

Table 5: Hyperparameters used for a random search.

Table 6: Peak memory and run time of different optimizers for GCViT on ImageWoof10 (Figure 6, *right*). Parenthesized values are normalized relative to SGD. For this vision transformer task, we observe that the backpropagation dominates both run time and memory. In this setting, all our methods as well as INGD have basically no run time and memory overhead compared to the first-order methods. INGD and our proposed methods are even able to beat AdamW and SGD in terms of test error. INGD, KFAC and SINGD update their preconditioner every $T = 5$ iterations.

Method	**Peak memory [GiB]**	**Training time [min]**
SGD (BFP-16)	15.6 (1.00 x)	190 (1.00 x)
AdamW (BFP-16)	15.7 (1.00 x)	191 (1.01 x)
SINGD-Diag* (BFP-16)	15.8 (1.02 x)	200 (1.06 x)
IKFAC* (BFP-16)	16.0 (1.02 x)	197 (1.04 x)
INGD (BFP-16)	16.0 (1.02 x)	203 (1.07 x)
KFAC (FP-32)	16.0 (1.02 x)	359 (1.89 x)

INGD

Each T iter., update $\mathbf{m}_K, \mathbf{m}_C, \mathbf{K}, \mathbf{C}$
Obtain $\boldsymbol{\mu}_{AA} \otimes \boldsymbol{\mu}_{GG}$ to approximate $\nabla^2_\mu \ell(\boldsymbol{\mu})$
$\mathbf{m}_K \leftarrow \alpha_1 \mathbf{m}_K + \frac{\beta_1}{2d}(\mathrm{Tr}(\mathbf{H}_C)\mathbf{H}_K + c^2\mathbf{K}^T\mathbf{K} - d\mathbf{I}_p)$
$\mathbf{m}_C \leftarrow \alpha_1 \mathbf{m}_C + \frac{\beta_1}{2p}(\mathrm{Tr}(\mathbf{H}_K)\mathbf{H}_C + \kappa^2\mathbf{C}^T\mathbf{C} - p\mathbf{I}_d)$
$\mathbf{K} \leftarrow \mathbf{K}\mathrm{Expm}(-\mathbf{m}_K) \approx \mathbf{K}(\mathbf{I}_p - \mathbf{m}_K)$
$\mathbf{C} \leftarrow \mathbf{C}\mathrm{Expm}(-\mathbf{m}_C) \approx \mathbf{C}(\mathbf{I}_d - \mathbf{m}_C)$
$\mathbf{M}_\mu \leftarrow \alpha_2 \mathbf{M}_\mu + \mathbf{C}\mathbf{C}^T \mathrm{vec}^{-1}(\nabla_\mu \ell(\boldsymbol{\mu}))\mathbf{K}\mathbf{K}^T + \gamma \mathrm{vec}^{-1}(\boldsymbol{\mu})$
$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \beta_2 \mathrm{vec}(\mathbf{M}_\mu)$

AdamW Optimizer

At iter. t, update $\mathbf{m}_s, \mathbf{s}$
Use $(\nabla_\mu \ell(\boldsymbol{\mu}))^2$ to approximate $\mathrm{diag}\,(\nabla^2_\mu \ell(\boldsymbol{\mu}))$
$\mathbf{m}_s \leftarrow (1-\beta_1)\mathbf{m}_s + \beta_1 \left(\nabla_\mu \ell(\boldsymbol{\mu})\right)^2$
$\mathbf{s}^2 \leftarrow \mathbf{m}_s / (1-(1-\beta_1)^t)$
$\mathbf{s} \leftarrow \sqrt{\mathbf{s}^2} + \lambda$
$\mathbf{m}_\mu \leftarrow \alpha_2 \mathbf{m}_\mu + (1-\alpha_2)\nabla_\mu \ell(\boldsymbol{\mu})$
$\mathbf{M}_\mu \leftarrow \mathbf{s}^{-1}\mathbf{m}_\mu / (1-\alpha_2^t)$
$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \beta_2 \mathbf{M}_\mu + \gamma\boldsymbol{\mu}$

Figure 8: Baseline methods in the same notation for a hyperparameter search.

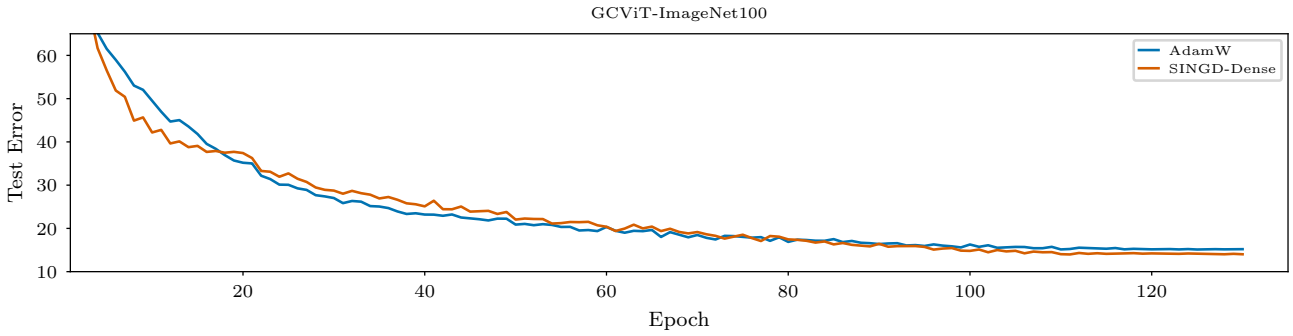

Figure 9: Test error curves for mixed-precision training on a GCViT model on dataset 'ImageNet-100'. SINGD has a similar iteration cost as AdamW while achieving better performance.

C. Connection between IKFAC and KFAC

To relate to the KFAC method, we now show that $\mathbf{K}^{\text{new}}\left(\mathbf{K}^{\text{new}}\right)^\top$ is an approximation of $\left(\mathbf{S}_K^{\text{new}} + \lambda\mathbf{I}\right)^{-1}$ at a new step of our scheme. For simplicity, we first assume $\mathbf{K}\mathbf{K}^\top$ exactly equals to $(\mathbf{S}_K^{\text{cur}} + \lambda\mathbf{I})^{-1}$ at the current step. Later, we will relax this assumption and prove that $\mathbf{K}\mathbf{K}^\top$ is an approximation of $(\mathbf{S}_K + \lambda\mathbf{I})^{-1}$ at every step as stated in Theorem 1. For notation simplicity, we denote $\bar{\mathbf{S}}_K \coloneqq \mathbf{S}_K + \lambda\mathbf{I}$. The update of $\mathbf{S}_K$ with damping $\lambda\mathbf{I}$ can be reexpressed as an update of $\bar{\mathbf{S}}_K$:

$$(\mathbf{S}_K^{\text{new}} + \lambda\mathbf{I}) = \bar{\mathbf{S}}_K^{\text{new}} \leftarrow (1-\beta_1)\bar{\mathbf{S}}_K^{\text{cur}} + \beta_1 (\mathbf{U} + \lambda\mathbf{I}) .$$

Since $\hat{\mathbf{S}}_K^{\text{cur}} = \mathbf{K}^{-T}\mathbf{K}^{-1}$ by our assumption, we can express update of $\mathbf{S}_K$ in terms of $\mathbf{K}$ as follows.

$$\bar{\mathbf{S}}_K^{\text{new}} \leftarrow (1-\beta_1)\bar{\mathbf{S}}_K^{\text{cur}} + \beta_1 (\mathbf{U} + \lambda\mathbf{I}) = \mathbf{K}^{-T}\left(\mathbf{I} + \beta_1 \left(\mathbf{K}^\top\mathbf{U}\mathbf{K} + \lambda\mathbf{K}^\top\mathbf{K} - \mathbf{I}\right)\right)\mathbf{K}^{-1} = \mathbf{K}^{-T}\left(\mathbf{I} + \beta_1\mathbf{m}_K\right)\mathbf{K}^{-1}$$

$\bar{\mathbf{S}}_K^{\text{new}}$ in the KFAC update can be approximated as below, where we consider $\mathbf{I} + \beta_1\mathbf{m}_K$ as an ap-

proximate of the matrix exponential $\text{Expm}(\beta_1 \mathbf{m}_K) \approx \mathbf{I} + \beta_1 \mathbf{m}_K$ and notice that $\mathbf{m}_K$ is symmetric.

$$\bar{\mathbf{S}}_K^{\text{new}} = \mathbf{K}^{-T}\left(\mathbf{I} + \beta_1 \mathbf{m}_K\right)\mathbf{K}^{-1} \approx \mathbf{K}^{-T}\text{Expm}\left(\beta_1 \mathbf{m}_K\right)\mathbf{K}^{-1} = \mathbf{K}^{-T}\text{Expm}\Big(\frac{\beta_1}{2}\mathbf{m}_K\Big)^\top \text{Expm}\Big(\frac{\beta_1}{2}\mathbf{m}_K\Big)\mathbf{K}^{-1}.$$

Informally, we can see that $\mathbf{K}^{\text{new}}\big(\mathbf{K}^{\text{new}}\big)^\top$ approximates $\big(\bar{\mathbf{S}}_K^{\text{new}}\big)^{-1}$ by using the matrix exponential. We can see that $\mathbf{m}_K$ stays in a matrix logarithm space.

$$\big(\bar{\mathbf{S}}_K^{\text{new}}\big)^{-1} \approx \mathbf{K}\text{Expm}\Big(-\frac{\beta_1}{2}\mathbf{m}_K\Big)\text{Expm}\Big(-\frac{\beta_1}{2}\mathbf{m}_K\Big)^\top \mathbf{K}^\top \approx \mathbf{K}\Big(\mathbf{I} - \frac{\beta_1}{2}\mathbf{m}_K\Big)\Big(\mathbf{I} - \frac{\beta_1}{2}\mathbf{m}_K\Big)^T \mathbf{K}^\top = \mathbf{K}^{\text{new}}\big(\mathbf{K}^{\text{new}}\big)^\top$$

Theorem 1 formally shows that $\mathbf{K}\mathbf{K}^\top$ used in our update is an approximation of $\Big(\mathbf{S}_K + \lambda\mathbf{I}\Big)^{-1}$ in the KFAC update for every step even when the truncation of the matrix exponential is employed.

D. Proof of Theorem 1

We first consider the following lemmas in order to prove Theorem 1.

Recall that we denote $\bar{\mathbf{S}}_K \coloneqq \mathbf{S}_K + \lambda\mathbf{I}$. For notation simplicity, we will drop the subscript K in this section and use $\bar{\mathbf{S}}_t$ to denote $\bar{\mathbf{S}}_K$ at iteration t. Notice that $\bar{\mathbf{S}}_t$ is non-singular at each iteration t so that we can inverse it in the original KFAC update (see Figure 3).

Lemma D.1. *Consider the following update in the original KFAC update at iteration t.*

$$\bar{\mathbf{S}}_t \coloneqq (1-\beta_1)\bar{\mathbf{S}}_{t-1} + \beta_1\big(\hat{\mathbf{U}}_{t-1} + \lambda\mathbf{I}\big)$$

where $\mathbf{S}_t$ is the factor $\mathbf{S}_K$ used in the original KFAC update, β_1 is known as the weight of the moving average, and $\hat{\mathbf{U}}_{t-1}$ is a curvature matrix.

The initial factor $\bar{\mathbf{S}}_0$ can be decomposed as $\bar{\mathbf{S}}_0 = \hat{\mathbf{K}}_0^{-T}\hat{\mathbf{K}}_0^{-1}$ since $\bar{\mathbf{S}}_0$ as a preconditioning factor is symmetric positive definite.

Define $\hat{\mathbf{N}}_i \coloneqq \hat{\mathbf{K}}_0^T\hat{\mathbf{U}}_i\hat{\mathbf{K}}_0 + \lambda\hat{\mathbf{K}}_0^T\hat{\mathbf{K}}_0 - \mathbf{I}$.

The Kronecker factor can be reexpressed as

$$\bar{\mathbf{S}}_t = \hat{\mathbf{K}}_0^{-T}\left(\mathbf{I} + \beta_1\sum_{i=0}^{t-1}\hat{\mathbf{N}}_i\right)\hat{\mathbf{K}}_0^{-1} + O(\beta_1^2)$$

Lemma D.2. *Consider the following update in our inverse-free KFAC at iteration t.*

$$\mathbf{K}_t \coloneqq \mathbf{K}_{t-1}\left(\mathbf{I} - \frac{\beta_1}{2}\left(\mathbf{K}_{t-1}^\top\mathbf{U}_{t-1}\mathbf{K}_{t-1} + \lambda\mathbf{K}_{t-1}^\top\mathbf{K}_{t-1} - \mathbf{I}\right)\right)$$

where $\mathbf{K}_{t-1}^\top\mathbf{U}_{t-1}\mathbf{K}_{t-1}$ is used in our update and $\mathbf{U}_{t-1}$ is a curvature matrix.

Define $\mathbf{N}_i \coloneqq \mathbf{K}_i^\top\mathbf{U}_i\mathbf{K}_i + \lambda\mathbf{K}_i^\top\mathbf{K}_i - \mathbf{I}$.

Our update of $\mathbf{K}$ can be reexpressed as

$$\mathbf{K}_t = \mathbf{K}_0\left(\mathbf{I} - \frac{\beta_1}{2}\sum_{i=0}^{t-1}\mathbf{N}_i\right) + O(\beta_1^2)$$

Moreover, the product $\mathbf{K}\mathbf{K}^\top$ can be reexpressed as

$$\mathbf{K}_t\mathbf{K}_t^\top = \mathbf{K}_0\left(\mathbf{I} - \beta_1\sum_{i=0}^{t-1}\mathbf{N}_i\right)\mathbf{K}_0^\top + O(\beta_1^2)$$

Lemma D.3 is useful to establish a relationship between the KFAC update and our inverse-free update.

Lemma D.3. *If we use the same sequence of curvature matrices in both the original KFAC update and our update such as* $\hat{\mathbf{U}}_i = \mathbf{U}_i$ *for each iteration* i *and* $\hat{\mathbf{K}}_0 = \mathbf{K}_0$ *are used on the initialization, we have the following expression.*

$$\mathbf{N}_i = \hat{\mathbf{N}}_i + O(\beta_1)$$

Similarly, we have the following result for $\mathbf{C}$.

Theorem 2. The product $\mathbf{C}\mathbf{C}^\top$ has a first-order accuracy of the KFAC update of $\left(\mathbf{S}_C + \lambda\mathbf{I}\right)^{-1}$ at each iteration if the update of $\mathbf{C}$ is updated according to Figure 3 with the truncation of the matrix exponential and these two updates use the same initialization and the same sequence of curvature matrices $\mathbf{G}$.

$$\mathbf{C}\mathbf{C}^\top = \left(\mathbf{S}_C + \lambda\mathbf{I}\right)^{-1} + O(\beta_1^2)$$

D.1. Proof of Lemma D.1

We prove the lemma by induction We first show the base case when $t = 1$. By definition, we have

$$\begin{aligned}
\bar{\mathbf{S}}_1 &= (1-\beta_1)\bar{\mathbf{S}}_0 + \beta_1\left(\hat{\mathbf{U}}_0 + \lambda\mathbf{I}\right) && (10)\\
&= (1-\beta_1)\hat{\mathbf{K}}_0^{-T}\hat{\mathbf{K}}_0^{-1} + \beta_1\left(\hat{\mathbf{U}}_0 + \lambda\mathbf{I}\right) && (11)\\
&= \hat{\mathbf{K}}_0^{-T}\Big[\mathbf{I} + \beta_1 \underbrace{\left(\hat{\mathbf{K}}_0^T\hat{\mathbf{U}}_0\hat{\mathbf{K}}_0 + \lambda\hat{\mathbf{K}}_0^T\hat{\mathbf{K}}_0 - \mathbf{I}\right)}_{=\hat{\mathbf{N}}_0}\Big]\hat{\mathbf{K}}_0^{-1} && (12)\\
&= \hat{\mathbf{K}}_0^{-T}\left[\mathbf{I} + \beta_1\hat{\mathbf{N}}_0\right]\hat{\mathbf{K}}_0^{-1} && (13)
\end{aligned}$$

Thus, the claim holds when $t = 1$.

Suppose, the claim holds when $t = n$. By the claim, we have

$$\bar{\mathbf{S}}_n = \hat{\mathbf{K}}_0^{-T}\left(\mathbf{I} + \beta_1\sum_{i=0}^{n-1}\hat{\mathbf{N}}_i\right)\hat{\mathbf{K}}_0^{-1} + O(\beta_1^2) \tag{14}$$

Now, we consider the case when $t = n + 1$. Notice that

$$\begin{aligned}
(1-\beta_1)\bar{\mathbf{S}}_n &= \hat{\mathbf{K}}_0^{-T}\left(\mathbf{I} + \beta_1\sum_{i=0}^{n-1}\hat{\mathbf{N}}_i - \beta_1\mathbf{I} + O(\beta_1^2)\right)\hat{\mathbf{K}}_0^{-1} + O(\beta_1^2)\\
&= \hat{\mathbf{K}}_0^{-T}\left(\mathbf{I} + \beta_1\sum_{i=0}^{n-1}\hat{\mathbf{N}}_i - \beta_1\mathbf{I}\right)\hat{\mathbf{K}}_0^{-1} + O(\beta_1^2)
\end{aligned}$$

By the definition of $\hat{\mathbf{S}}_{n+1}$, we have

$$\begin{aligned}
\bar{\mathbf{S}}_{n+1} &= (1-\beta_1)\bar{\mathbf{S}}_n + \beta_1\left(\hat{\mathbf{U}}_n + \lambda\mathbf{I}\right) && (15)\\
&= \hat{\mathbf{K}}_0^{-T}\left(\mathbf{I} + \beta_1\sum_{i=0}^{n-1}\hat{\mathbf{N}}_i \underbrace{-\beta_1\mathbf{I} + \beta_1\hat{\mathbf{K}}_0^T\hat{\mathbf{U}}_n\hat{\mathbf{K}}_0 + \beta_1\lambda\hat{\mathbf{K}}_0^T\hat{\mathbf{K}}_0}_{=\beta_1\hat{\mathbf{N}}_n}\right)\hat{\mathbf{K}}_0^{-1} + O(\beta_1^2) && (16)\\
&= \hat{\mathbf{K}}_0^{-T}\left(\mathbf{I} + \beta_1\sum_{i=0}^{n}\hat{\mathbf{N}}_i\right)\hat{\mathbf{K}}_0^{-1} + O(\beta_1^2) && (17)
\end{aligned}$$

which is exactly the claim when $t = n + 1$.

Thus, by induction, the claim holds.

D.2. Proof of Lemma D.2

We prove the lemma by induction We first show the base case when $t = 1$. By definition, we have

$$\mathbf{K}_1 = \mathbf{K}_0\Big(\mathbf{I} - \frac{\beta_1}{2}\underbrace{\Big(\mathbf{K}_0^\top\mathbf{U}_0\mathbf{K}_0 + \lambda\mathbf{K}_0^\top\mathbf{K}_0 - \mathbf{I}\Big)}_{=\mathbf{N}_0}\Big) \tag{18}$$

Thus, the claim holds when $t = 1$.

Suppose, the claim holds when $t = n$. By the claim, we have

$$\mathbf{K}_n = \mathbf{K}_0\left(\mathbf{I} - \frac{\beta_1}{2}\sum_{i=0}^{n-1}\mathbf{N}_i\right) + O(\beta_1^2) \tag{19}$$

Now, we consider the case when $t = n + 1$. Notice that

$$\mathbf{K}_{n+1} = \mathbf{K}_n\Big(\mathbf{I} - \frac{\beta_1}{2}\underbrace{\Big(\mathbf{K}_n^\top\mathbf{U}_n\mathbf{K}_n + \lambda\mathbf{K}_n^\top\mathbf{K}_n - \mathbf{I}\Big)}_{=\mathbf{N}_n}\Big) \tag{20}$$

$$= \underbrace{\mathbf{K}_0\left(\mathbf{I} - \frac{\beta_1}{2}\sum_{i=0}^{n-1}\mathbf{N}_i\right)}_{=\mathbf{K}_n - O(\beta_1^2)}\Big(\mathbf{I} - \frac{\beta_1}{2}\mathbf{N}_n\Big) + O(\beta_1^2) \tag{21}$$

$$= \mathbf{K}_0\left(\mathbf{I} - \frac{\beta_1}{2}\sum_{i=0}^{n-1}\mathbf{N}_i - \frac{\beta_1}{2}\mathbf{N}_n + O(\beta_1^2)\right) + O(\beta_1^2) \tag{22}$$

$$= \mathbf{K}_0\left(\mathbf{I} - \frac{\beta_1}{2}\sum_{i=0}^{n}\mathbf{N}_i\right) + O(\beta_1^2) \tag{23}$$

which is exactly the claim when $t = n + 1$.

Thus, by induction, the claim holds.

Notice that $\mathbf{N}_i$ by definition is symmetric. It is easy to see that

$$\mathbf{K}_t\mathbf{K}_t^\top = \mathbf{K}_0\left(\mathbf{I} - \frac{\beta_1}{2}\sum_{i=0}^{t-1}\mathbf{N}_i\right)\left(\mathbf{I} - \frac{\beta_1}{2}\sum_{i=0}^{t-1}\mathbf{N}_i\right)^\top\mathbf{K}_0^\top + O(\beta_1^2) \tag{24}$$

$$= \mathbf{K}_0\left(\mathbf{I} - \frac{\beta_1}{2}\sum_{i=0}^{t-1}\mathbf{N}_i\right)\left(\mathbf{I} - \frac{\beta_1}{2}\sum_{i=0}^{t-1}\mathbf{N}_i\right)\mathbf{K}_0^\top + O(\beta_1^2) \tag{25}$$

$$= \mathbf{K}_0\left(\mathbf{I} - \beta_1\sum_{i=0}^{t-1}\mathbf{N}_i\right)\mathbf{K}_0^\top + O(\beta_1^2) \tag{26}$$

Thus, the claim also holds.

D.3. Proof of Lemma D.3

We first show the base case when $t = 0$. By the assumption, we have $\mathbf{K}_0 = \hat{\mathbf{K}}_0$. Similarly, we have $\mathbf{U}_0 = \hat{\mathbf{U}}_0$ by the assumption.

By definition, we have

$$\mathbf{N}_0 = \mathbf{K}_0^\top\mathbf{U}_0\mathbf{K}_0 + \lambda\mathbf{K}_0^\top\mathbf{K}_0 - \mathbf{I} \tag{27}$$

$$= \hat{\mathbf{K}}_0^\top\hat{\mathbf{U}}_0\hat{\mathbf{K}}_0 + \lambda\hat{\mathbf{K}}_0^\top\hat{\mathbf{K}}_0 - \mathbf{I} \tag{28}$$

$$= \hat{\mathbf{N}}_0 \tag{29}$$

Thus, the claim holds when $t = 0$.

When $t > 0$, we can use Lemma D.2 to obtain the claim. Notice that

$$\mathbf{N}_{n+1} = \mathbf{K}_{n+1}^\top \mathbf{U}_{n+1} \mathbf{K}_{n+1} + \lambda \mathbf{K}_{n+1}^\top \mathbf{K}_{n+1} - \mathbf{I} \tag{30}$$

$$= \left(\mathbf{I} - \frac{\beta_1}{2} \sum_{i=0}^{n} \mathbf{N}_i\right)^\top \mathbf{K}_0^\top \big(\mathbf{U}_{n+1} + \lambda \mathbf{I}\big) \mathbf{K}_0 \left(\mathbf{I} - \frac{\beta_1}{2} \sum_{i=0}^{n} \mathbf{N}_i\right) - \mathbf{I} + O(\beta_1^2) \text{ (Lemma 2)} \tag{31}$$

$$= \mathbf{K}_0^\top \big(\mathbf{U}_{n+1} + \lambda \mathbf{I}\big) \mathbf{K}_0 + O(\beta_1) + O(\beta_1^2) \tag{32}$$

$$= \hat{\mathbf{K}}_0^\top \big(\hat{\mathbf{U}}_{n+1} + \lambda \mathbf{I}\big) \hat{\mathbf{K}}_0 + O(\beta_1) \text{ (Assumption)} \tag{33}$$

$$= \hat{\mathbf{N}}_{n+1} + O(\beta_1) \tag{34}$$

D.4. Proof of Theorem 1

It is sufficient to show that the following claim holds at iteration t since $\bar{\mathbf{S}}_t$ is non-singular.

$$\mathbf{K}_t \mathbf{K}_t^\top \bar{\mathbf{S}}_t = \mathbf{I} + O(\beta_1^2)$$

where we use $\bar{\mathbf{S}}_t$ to denote $\bar{\mathbf{S}}_K$ at iteration t.

By assumptions, we know that Lemmas D.1, D.2, D.3 hold. Moreover, we have $\mathbf{K}_0 = \hat{\mathbf{K}}_0$. Thus, we have

$$\mathbf{K}_t \mathbf{K}_t^\top \bar{\mathbf{S}}_t = \mathbf{K}_0 \left(\mathbf{I} - \beta_1 \sum_{i=0}^{t-1} \mathbf{N}_i\right) \mathbf{K}_0^\top \bar{\mathbf{S}}_t + O(\beta_1^2) \text{ (by Lemma D.2)} \tag{35}$$

$$= \mathbf{K}_0 \left(\mathbf{I} - \beta_1 \sum_{i=0}^{t-1} \mathbf{N}_i\right) \mathbf{K}_0^\top \hat{\mathbf{K}}_0^{-T} \left(\mathbf{I} + \beta_1 \sum_{i=0}^{t-1} \hat{\mathbf{N}}_i\right) \hat{\mathbf{K}}_0^{-1} + O(\beta_1^2) \text{ (by Lemma D.1)} \tag{36}$$

$$= \hat{\mathbf{K}}_0 \left(\mathbf{I} - \beta_1 \sum_{i=0}^{t-1} \hat{\mathbf{N}}_i + O(\beta_1^2)\right) \left(\mathbf{I} + \beta_1 \sum_{i=0}^{t-1} \hat{\mathbf{N}}_i\right) \hat{\mathbf{K}}_0^{-1} + O(\beta_1^2) \text{ (by Lemma D.3)} \tag{37}$$

$$= \hat{\mathbf{K}}_0 \mathbf{I} \hat{\mathbf{K}}_0^{-1} + O(\beta_1^2) \tag{38}$$

$$= \mathbf{I} + O(\beta_1^2) \tag{39}$$

E. Invariance of INGD and SINGD

INGD and SINGD are scale invariant to the choice of the Kronecker approximation while KFAC and IKFAC are not. Recall that we use the following Kronecker approximation to approximate the Hessian.

$$\mathbf{U} \otimes \mathbf{G} \approx \nabla_\mu^2 \ell(\boldsymbol{\mu})$$

However, such an approximation is not unique. We can consider an equivalent approximation such as

$$(\alpha \mathbf{U}) \otimes (\alpha^{-1} \mathbf{G}) \approx \nabla_\mu^2 \ell(\boldsymbol{\mu})$$

where $\alpha \neq 0$ can be any arbitrary non-zero scalar.

INGD is invariant since the update scheme involving the approximation is scale invariant: $\mathrm{Tr}(\mathbf{H}_C)\mathbf{H}_K = \mathrm{Tr}(\mathbf{C}^T \mathbf{G} \mathbf{C}) \mathbf{K}^T \mathbf{U} \mathbf{K} = \mathrm{Tr}(\mathbf{C}^T (\alpha^{-1} \mathbf{G}) \mathbf{C}) \mathbf{K}^T (\alpha \mathbf{U}) \mathbf{K}$. The invariance is also preserved in SINGD since structures and their subspace projection maps are closed under scalar multiplications.

In contrast, the updates of KFAC and IKFAC are not scale invariant. As an example, we consider using curvature approximations $\mathbf{U}$ and $(\alpha \mathbf{U})$ to update $\mathbf{S}_K^{-1}$ in KFAC, and denote the updated $\mathbf{S}_K^{-1}$ by $\hat{\mathbf{S}}_K^{-1}$ and $\bar{\mathbf{S}}_K^{-1}$, respectively. As shown below, we cannot recover $\hat{\mathbf{S}}_K^{-1}$ from $\bar{\mathbf{S}}_K^{-1}$ by scale transformations and thus, the KFAC update is not scale invariant.

$$\hat{\mathbf{S}}_K^{-1} = \big[(1 - \beta_1)\hat{\mathbf{S}}_K + \beta_1 \mathbf{U} + \lambda \mathbf{I}\big]^{-1} \neq \big[(1 - \beta_1)\bar{\mathbf{S}}_K + \beta_1 (\alpha \mathbf{U}) + \lambda \mathbf{I}\big]^{-1} = \bar{\mathbf{S}}_K^{-1}$$

An attempt to make the update of $\mathbf{S}_K$ invariant is to set the damping weight to be $\alpha\lambda$. However, the update of $\mathbf{S}_C$ requires us to set the damping weight to be $\alpha^{-1}\lambda$ as shown below. Thus, it is impossible to make KFAC invariant without introducing individual damping weights.

$$\hat{\mathbf{S}}_C^{-1} = \left[(1-\beta_1)\hat{\mathbf{S}}_C + \beta_1\mathbf{G} + \lambda\mathbf{I}\right]^{-1} \neq \left[(1-\beta_1)\bar{\mathbf{S}}_C + \beta_1(\alpha^{-1}\mathbf{G}) + \lambda\mathbf{I}\right]^{-1} = \bar{\mathbf{S}}_C^{-1}$$