SINGD: Structured Inverse-Free Natural Gradient Descent

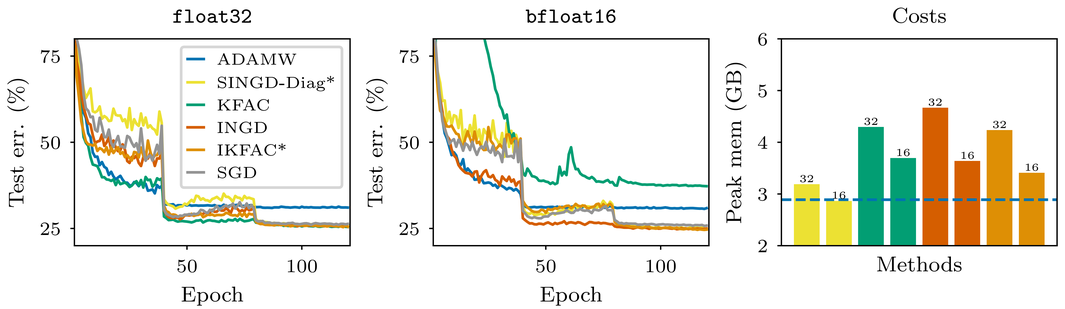
Wu Lin*, Felix Dangel*, Runa Eschenhagen, Kirill Neklyudov, Agustinus Kristiadi, Richard Turner, Alireza Makhzani

University of Toronto · Vector Institute · University of Cambridge

We present a **KFAC-style 2nd-order method** for DL that is **memory-efficient & numerically-stable in `bfloat16`** thanks to structured Kronecker factors and inverse-free update rule.

Paper | Code

Inefficiencies of KFAC: Numerically Unstable In `bfloat16` & Memory-Intensive due to Dense Kronecker Factors



Computational efficiency on VGG + CIFAR-100

Method	Memory	Time
SGD, `bfloat16`	1.00 x	1.00 x
AdamW, `bfloat16`	1.02 x	1.07 x
SINGD-Diag*, `bfloat16`	**1.02 x**	**1.29 x**
IKFAC*, `bfloat16`	1.21 x	1.84 x
INGD, `bfloat16`	1.29 x	1.84 x
KFAC, `float32`	**1.52 x**	**4.49 x**

SINGD = Inverse-free KFAC + Structures

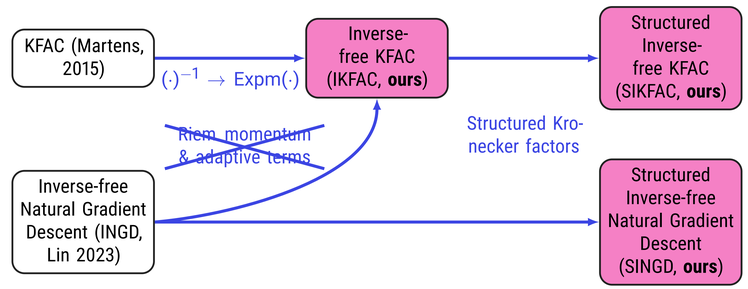
Goal: Approximate natural gradient $\mathbf{F}^{-1}\mathbf{g}$ with gradient $\mathbf{g}$, inverse Fisher approximation $\mathbf{F}^{-1}$, and damping λ. Per layer:

$$\textbf{KFAC:}\quad \mathbf{F}^{-1} = (\mathbf{S}_\mathbf{K} + \lambda\mathbf{I})^{-1} \otimes (\mathbf{S}_\mathbf{C} + \lambda\mathbf{I})^{-1}, \quad \text{updates } \mathbf{S}_\mathbf{K}, \mathbf{S}_\mathbf{C}$$

$$\textbf{INGD:}\quad \mathbf{F}^{-1} = \mathbf{K}\mathbf{K}^\top \otimes \mathbf{C}\mathbf{C}^\top, \quad \text{updates } \mathbf{K}, \mathbf{C}$$

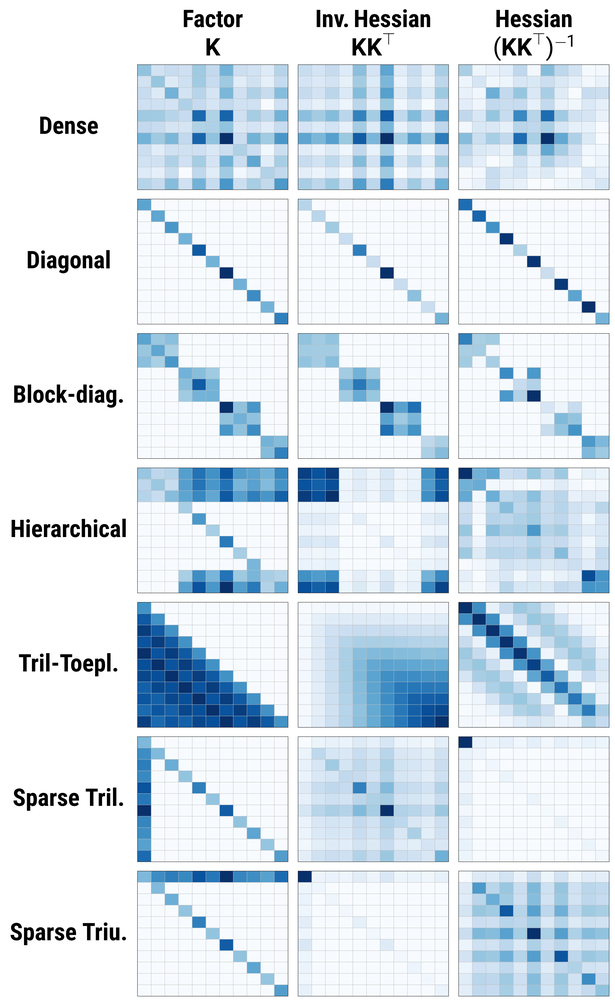
$$\textbf{SINGD:}\quad \mathbf{F}^{-1} = \mathbf{K}\mathbf{K}^\top \otimes \mathbf{C}\mathbf{C}^\top, \quad \text{with structured } \mathbf{K}, \mathbf{C}$$

Theorem 1 (informal): At each iteration, **IKFAC's update** of the Kronecker factors $\mathbf{K}/\mathbf{C}$ **matches KFAC's update** of $(\mathbf{S}_{\mathbf{K}/\mathbf{C}} + \lambda\mathbf{I})^{-1}$ **up to first order** in the exponential moving average (default $\lesssim 10^{-2}$), assuming same initialization and sequence of layer statistics.

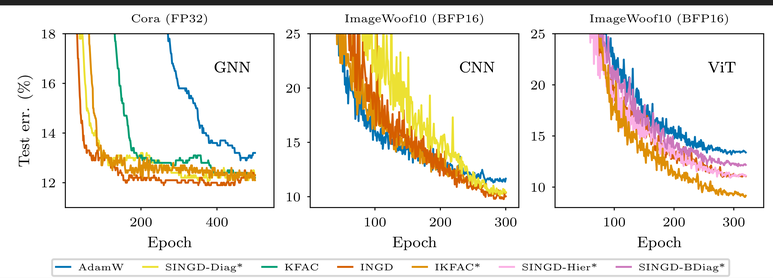


Available Structures

Different structures capture different information.



Performs Well on Many Problems & in `bfloat16`



Works like Any Optimizer (`pip install singd`)

```python
from singd.optim.optimizer import SINGD
from torch_utils import CrossEntropyLoss, alexnet, new_batch

# 1. Setup
model, loss_func = alexnet(), CrossEntropyLoss()
X, y = new_batch()

optim = SINGD(  # Optimizer setup
    model,
    kfac_like=False,  # Use True for IKFAC
    structures=("dense", "diagonal"),  # Structure of (K, C)
)

# 2. One training step
optim.zero_grad()  # Clear previous gradients
loss_func(model(X), y).backward()  # Forward + backward pass
optim.step()  # Update pre-conditioner and parameters
```