Determinant Estimation under Memory Constraints and Neural Scaling Laws

Siavash Ameli [1,2] Chris van der Heide [3] Liam Hodgkinson [4] Fred Roosta [5]
Michael W. Mahoney [1,2,6]

[1] *Department of Statistics, UC Berkeley*
[2] *International Computer Science Institute*
[3] *Dept. of Electrical and Electronic Eng., University of Melbourne*
[4] *School of Mathematics and Statistics, University of Melbourne*
[5] *CIRES and School of Mathematics and Physics, University of Queensland*
[6] *Lawrence Berkeley National Laboratory*

ICML 2025

Log-determinant is widely encountered in linear algebra and statistics:

- Gaussian process (kernel methods)
- Determinantal point process
- Volume form (Bayesian computation)

Challenges

- It is often **the most difficult term** to compute in these applications.
- **Memory-wall** (time complexity isn't the only bottleneck)

Outline

I. Large Matrices

- Neural Tangent Kernels
- Arithmetic Precision

II. MEMDET

- Compute exact log-det
- Out-of-core

III. FLODANCE

- Approximate log-det
- Utilize scale law

IIII. Results

- NTK matrices
- Matérn kernel

I. Large Matrices

Example of Extremely Challenging Matrices

Neural Tangent Kernel (NTK)

- Neural network $f_{\theta} : \mathcal{X} \to \mathbb{R}^{d}$
- θ: parameters
- $\mathbf{J}_{\theta}(f_{\theta}(x))$: Jacobian of f_{θ}
- NTK is Gramian of $\mathbf{J}_{\theta}$:

$$\kappa_{\theta}(x, x') := \mathbf{J}_{\theta}(f_{\theta}(x))\mathbf{J}_{\theta}(f_{\theta}(x'))^{\intercal}$$

Compute time of NTK (using NVIDIA H100 GPU)

		Compute Time (hrs)		
Dataset	Model	float16	float32	float64
MNIST	MobileNet	6	25	50
CIFAR-10	ResNet9	6	24	70
	ResNet18	14	63	65
	ResNet50	37	177	297
	ResNet101	107	442	1178

Challenges

Challenge I. Forming NTK

- Takes days/months to compute on H100 GPU
- Need large storage (from **Terabytes** to **Exabytes**)
- **Precision loss** when forming Gram matrix
- **double precision** to retain **positive-definiteness**

Challenge II. Computing LogDet

- **Cubic** complexity $\mathcal{O}(m^3)$
- NTK is nearly **singular**
- CIFAR-10: 10% of eigenvalues near zero
- Cannot load on **memory**

II. MEMDET

MEMDET

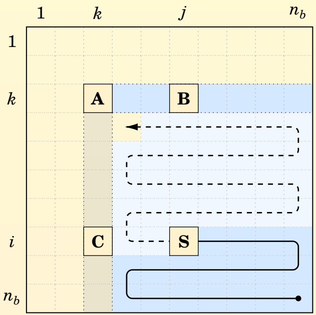
- Out-of-core algorithm
- Can process matrix of any scale
- Eliminates memory wall

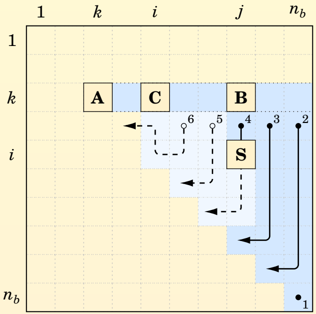
Block decompositions:

- LU decomposition: generic matrices

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} \\ \mathbf{M}_{21} & \mathbf{M}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{11} & \mathbf{0} \\ \mathbf{L}_{21} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} \\ \mathbf{0} & \mathbf{S} \end{bmatrix}$$

- Repeat decomposition on block **S**.
- LDL: for symmetric matrices
- Cholesky: for symmetric PD matrices

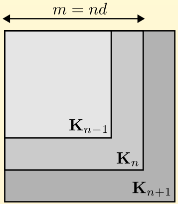




MEMDET Algorithm

- Only **four blocks A**, **B**, **C**, **S** on **memory**
- Blue blocks are written to disk (**scratch space**)
- Efficient **order of processing** of blocks
- *Figure*: LU (left) and LDL/Cholesky decompositions (right).

III. FLODANCE



$$\frac{\det(\mathbf{K}_n)}{\det(\mathbf{K}_{n-1})} \sim n^{\nu}$$

- n: num dataset
- d: num classes
- $m = nd$: matrix size

LEMMA

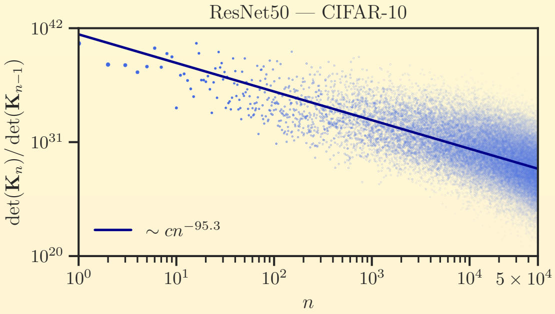
Let $f : \mathcal{X} \to \mathbb{R}^d$ be a zero-mean vector-valued m-dimensional Gaussian process with covariance kernel κ. For each $n \geq 2$, let

$$E(n) := \mathbb{E}[d^{-\frac{1}{2}} \|f(x_n)\|^2 \mid f(x_i) = 0$$

denote the mean-squared error of fitting the f to the zero function using $x_1, \ldots, x_{n-1}$. Then

$$\frac{\mathrm{pdet}(\mathbf{K}_n)}{\mathrm{pdet}(\mathbf{K}_{n-1})} \leq E(n)^d, \quad \forall n > 1,$$

with equality if $d = 1$.



- NTK of ResNet50 on CIFAR-10
- Number of classes: $d = 10$
- Dataset images: $n = 50\text{K}$
- Matrix size: $m = 500\text{K}$.

Proposition

Let $L_n := \frac{1}{n}\mathrm{logdet}(\mathbf{K}_n)$. *Then*

$$\hat{L}_n \approx L_1 + \left(1 - \frac{1}{n}\right) c_0 - \nu \frac{\log(n!)}{n}$$

- *Law of large numbers (LLN):*

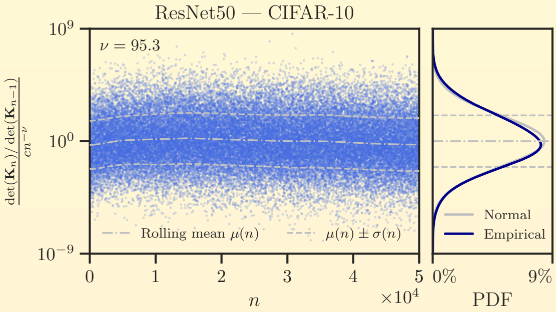
$$L_n = \hat{L}_n + o_p(1).$$

- *Central limit theorem (CLT):*

$$\frac{n}{\sqrt{n-1}}(L_n - \hat{L}_n) \xrightarrow{\mathcal{D}} \mathcal{N}(0, \sigma^2).$$

Algorithm:

- **Fit** $\hat{L}_n$ on submatrices $n = 1, \ldots, n_s \ll n$
- (Linear regression on parameters c_0, ν)
- **Extrapolate** to larger $n \gg n_s$

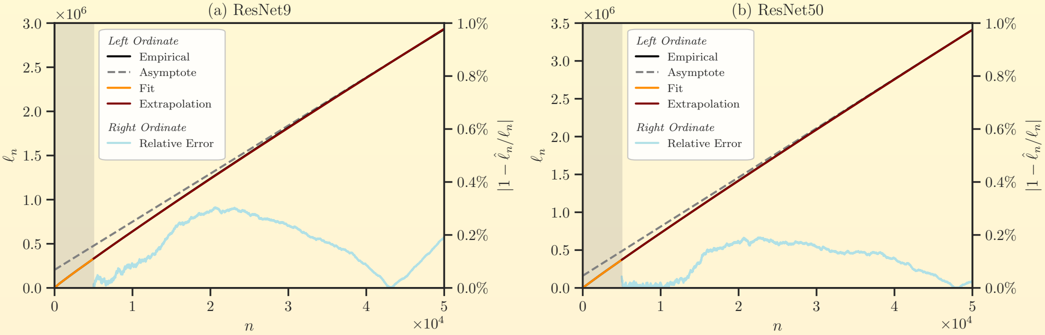


Assumptions:

- Stochastic process: $\dfrac{\det(\mathbf{K}_n)/\det(\mathbf{K}_{n-1})}{cn^{\nu}}$
- **Stationary** logarithmic process
- **Ergodic process**

IIII. Results

Estimating Log-Det — NTK Matrix



- Full CIFAR-10 data with all $n = 50\text{K}$ images
- Matrix size $m = 500{,}000$ dense matrix, **double precision**, **2TB** size.
- **Fit:** on 10% of total matrix size (shaded gray region, yellow curve)
- **Extrapolation:** in much larger interval (red curve)
- Error compared to MEMDET: (blue curve right axis in each panel), **0.2%** (left), **0.02%** (right).

Comparison of Methods

Method					
Name	Settings	TFLOPs	Rel. Error	Est. Cost	Wall Time
SLQ	$l = 100, s = 104$	5203	55%	\$83	1.8 days
MEMDET	LDL, $n_b = 32$	41,667	**0%**	\$601	13.8 days
FLODANCE	$n_s = 500, \quad q = 0$	**0.04**	4%	\$0.04	1 min
FLODANCE	$n_s = 5000, q = 4$	41.7	**0.02%**	\$4	1.5 hr

- **Largest NTK formation** and **exact logdet computation** to our knowledge
- ResNet50, full CIFAR-10 with all $n = 50\text{K}$ images
- Matrix size $m = 500{,}000$ dense matrix, **double precision**, **2TB** size.
- MEMDET computes the **exact** log-determinant, serves as **benchmark**.
- Costs and wall time are based on an NVIDIA H100 GPU (\$2/hour).
- Wall time include NTK formation.

Resources

Reference

Ameli, S., van der Heide, C., Hodgkinson, L., Roosta, F., Mahoney, M.W., (2025). Determinant Estimation under Memory Constraints and Neural Scaling Laws, *The 42nd International Conference on Machine Learning*.

Related Work

Ameli, S., van der Heide, C., Hodgkinson, L., Mahoney, M.W., (2025). Spectral Estimation with Free Decompression. *arXiv: 2506.11994*

Software

Package	Documentation	Install	Implements
detkit	`ameli.github.io/detkit`	`pip install detkit`	MEMDET FLODANCE
imate	`ameli.github.io/imate`	`pip install imate`	SLQ
freealg	`ameli.github.io/freealg`	`pip install freealg`	(Related work)