TransferLab

TransferLab Seminar

Second-Order Information and Applications

Kristof Schröder

11.04.2024

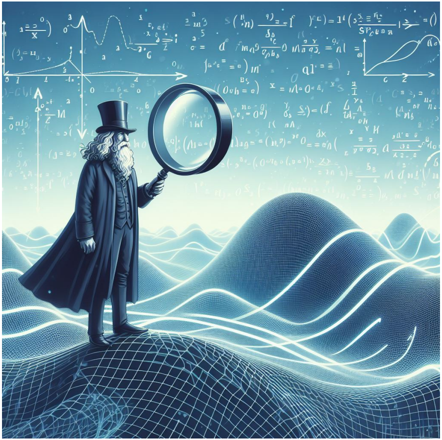
appliedAI Institute for Europe

Overview

Introduction

Why Second-Order?

- Encodes geometry/curvature information about the optimization objective
- Can reduce number of iterations
- Can improve the quality of the optimization result



Created with https://www.bing.com/chat

Gradient descent

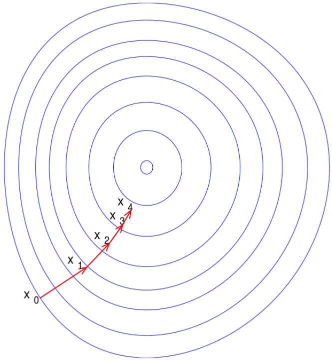
In order to solve (for $\mathcal{L}$ smooth, strongly convex)

$$\min_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta),$$

iterate

$$\theta_{t+1} = \theta_t - \gamma_t \nabla_\theta \mathcal{L}(\theta_t).$$

- convergence rate is linear
- convergence depends on the condition number of the Hessian at the solution



Source: Wikipedia

Newton's Method in Optimization

In order to solve

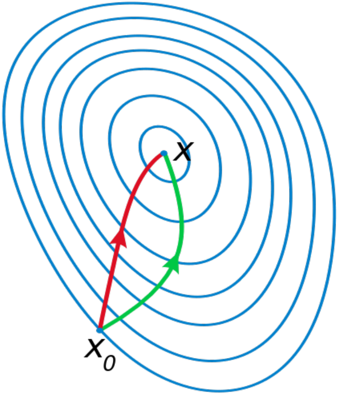
$$\min_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta),$$

iterate

$$\theta_{t+1} = \theta_t - H_{\mathcal{L}}(\theta_t)^{-1} \nabla_\theta \mathcal{L}(\theta_t)$$

where $H_{\mathcal{L}}(\theta_t) = \nabla_\theta^2 \mathcal{L}(\theta_t)$.

- quadratic convergence rate
- think of $H_{\mathcal{L}}(\theta_t)^{-1}$ as a preconditioner, i.e. $\mathrm{cond}(H_{\mathcal{L}}(\theta_t)^{-1} H_{\mathcal{L}}(\theta^\star))$ is small



Source: Wikipedia

Neural Networks

In this case

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i; \theta), y_i)$$

where $f(x, \theta)$ is a parametrized model and ℓ is some loss function.

Neural Networks

In this case

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i;\theta), y_i)$$

where $f(x,\theta)$ is a parametrized model and ℓ is some loss function.

Problems:

Neural Networks

In this case

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i; \theta), y_i)$$

where $f(x, \theta)$ is a parametrized model and ℓ is some loss function.

Problems:

- $\mathcal{L}$ is non-convex, Newton could even converge to a maximum

Neural Networks

In this case

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i; \theta), y_i)$$

where $f(x, \theta)$ is a parametrized model and ℓ is some loss function.

Problems:

- $\mathcal{L}$ is non-convex, Newton could even converge to a maximum
- loss-landscape is highly ill-conditioned, i.e. very heterogeneous curvature

Neural Networks

In this case

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i; \theta), y_i)$$

where $f(x, \theta)$ is a parametrized model and ℓ is some loss function.

Problems:

- $\mathcal{L}$ is non-convex, Newton could even converge to a maximum
- loss-landscape is highly ill-conditioned, i.e. very heterogeneous curvature
- memory cost is $\mathcal{O}(d^2)$ and naive inversion has complexity $\mathcal{O}(d^3)$

In this case

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i;\theta), y_i)$$

where $f(x, \theta)$ is a parametrized model and ℓ is some loss function.

Problems:

- $\mathcal{L}$ is non-convex, Newton could even converge to a maximum
- loss-landscape is highly ill-conditioned, i.e. very heterogeneous curvature
- memory cost is $\mathcal{O}(d^2)$ and naive inversion has complexity $\mathcal{O}(d^3)$

Strategies

Approximate the preconditioned gradients

$$H_{\mathcal{L}}(\theta)^{-1}\nabla_\theta \mathcal{L}(\theta)$$

What to do?

Strategies

Approximate the preconditioned gradients

$$H_{\mathcal{L}}(\theta)^{-1}\nabla_{\theta}\mathcal{L}(\theta)$$

What to do?

- Implicit inversion, block approximation, diagonal approximation,

Strategies

Approximate the preconditioned gradients

$$H_{\mathcal{L}}(\theta)^{-1}\nabla_{\theta}\mathcal{L}(\theta)$$

What to do?

- Implicit inversion, block approximation, diagonal approximation,
- Randomization: stochastic estimators + smoothing (EMA)

Strategies

Approximate the preconditioned gradients

$$H_{\mathcal{L}}(\theta)^{-1}\nabla_\theta \mathcal{L}(\theta)$$

What to do?

- Implicit inversion, block approximation, diagonal approximation,
- Randomization: stochastic estimators + smoothing (EMA)
- Low-rank approximations

Strategies

Approximate the preconditioned gradients

$$H_{\mathcal{L}}(\theta)^{-1}\nabla_{\theta}\mathcal{L}(\theta)$$

What to do?

- Implicit inversion, block approximation, diagonal approximation,
- Randomization: stochastic estimators + smoothing (EMA)
- Low-rank approximations
- Decomposition based approximations: Gauss-Newton approximation, Fisher information matrix (Natural Gradient Method)

LLM Pre-Training

LLM Pre-Training

Pre-training of a large language model typically incur significant expenses. Often the chosen state of the state-of-the-art solver is **Adam**.

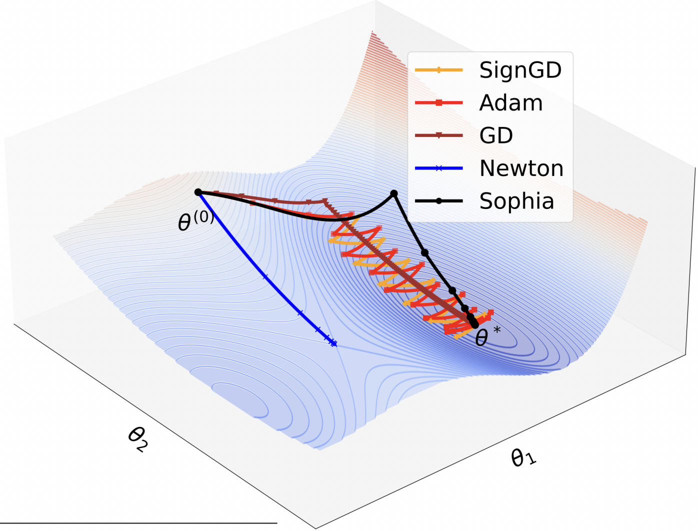
LLM Pre-Training

Pre-training of a large language model typically incur significant expenses. Often the chosen state of the state-of-the-art solver is **Adam**.

How to incorporate second-order information in order to

- reduce the number of iterations needed,
- only adding a small over-head per iteration,
- keep memory footprint comparable?

Motivating Example



[1] Liu et al., Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training, 2023 [LLH+23]

Sophia: Second-order Clipped Stochastic Optimization[2]

- Construct stochastic estimators for the diagonal of the Hessian.
- Exponential smoothing of minibatch gradients at each iteration.
- Exponential smoothing of the second-order information every $k = 10$ iteration.
- Per-coordinate clipping to handle negative curvature.

Algorithm 3 Sophia

Input: θ_1, learning rate $\{\eta_t\}_{t=1}^{T}$, hyperparameters $\lambda, \beta_1, \beta_2, \epsilon$, and estimator choice Estimator $\in$ {Hutchinson,Gauss-Newton-Bartlett}

Set $m_0 = 0$, $v_0 = 0$, $h_{1-k} = 0$
for $t = 1$ **to** T **do**
  Compute minibach loss $L_t(\theta_t)$.
  Compute $g_t = \nabla L_t(\theta_t)$.
  $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  if $t \bmod k = 1$ **then**
    Compute $\hat{h}_t = \text{Estimator}(\theta_t)$.
    $h_t = \beta_2 h_{t-k} + (1 - \beta_2)\hat{h}_t$
  else
    $h_t = h_{t-1}$
  $\theta_t = \theta_t - \eta_t \lambda \theta_t$ (weight decay)
  $\theta_{t+1} = \theta_t - \eta_t \cdot \text{clip}(m_t / \max\{h_t, \epsilon\}, \rho)$

[2] Liu et al., Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training, 2023 [LLH$^+$23]

Iterations

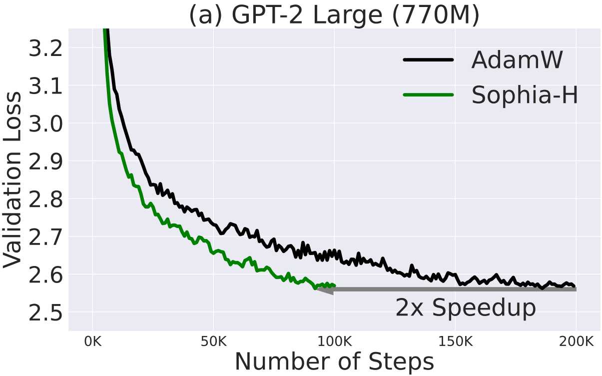


Figure 4: Loss evolution for training GPT-2 on OpenWebText[3]

[3] Liu et al., Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training, 2023 [LLH$^+$23]

Scaling Laws

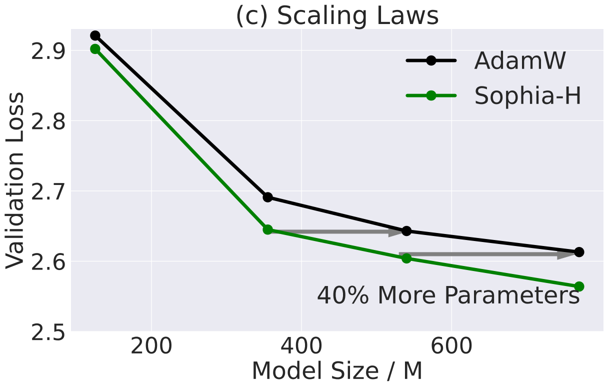


Figure 5: Validation loss vs. number of parameters[4]

[4] Liu et al., Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training, 2023 [LLH+23]

Computation Time

- same memory cost as AdamW,
- overall wall-clock time overhead less than 5%
- halving the number of iteration results in almost halving the total wall-clock time

Table 1: Wall-clock time and compute.

Algorithm	Model Size	T(step)	T(Hessian)	Compute
AdamW	770M	3.25s	–	2550
Sophia-H	770M	3.40s	0.12s	2708
Sophia-G	770M	3.42s	0.17s	2678
AdamW	355M	1.77s	–	1195
Sophia-H	355M	1.88s	0.09s	1249
Sophia-G	355M	1.86s	0.09s	1255

Figure 6: Computation Time[a]

[a] Liu et al., Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training, 2023 [LLH+23]

LLM Fine-Tuning

LLM Fine-Tuning

- Fine-tuning language models has been effective for various tasks, but the memory demands of backpropagation in large models are still significant.
- The limitations of available compute environments typically constrain the fine-tuning.
- Recently, zero-order methods have been adapted to fine-tuning LLMs[5](with up to 12x reduced memory consumption), but are known to converge slowly.

[5] Malladi et al., Fine-Tuning Language Models with Just Forward Passes, 2023 [MGN$^+$23]

Zero-Order Method

Definition (Simultaneous Perturbation Stochastic Approximation or SPSA)
For $\theta \in \mathbb{R}^d$ and loss function L estimate the gradient on a mini-batch B as

$$g_\varepsilon(\theta) = \frac{L(\theta + \varepsilon z, B) - L(\theta - \varepsilon z, B)}{2\varepsilon} \cdot z \approx zz^T \nabla_\theta L(\theta, B),$$

for $z \sim \mathcal{N}(0, \mathrm{Id}_d)$.

- Requires two forward passes per mini-natch, no back-propagation.
- Can be averaged over several samplings from $\mathcal{N}(0, \mathrm{Id}_d)$ (n-SPSA).

Motivation

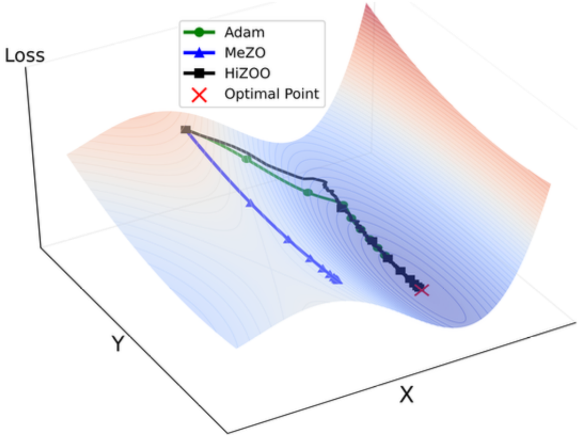


Figure 7: Toy example in figure 1[6]

[6] Zhao et al., Second-Order Fine-Tuning without Pain for LLMs: A Hessian Informed Zeroth-Order Optimizer, 2024 [ZDY^{+}24]

Hessian Informed Zero-Order Optimization (HiZOO)

For $\theta \in \mathbb{R}^d$ and loss function L estimate the gradient on a mini-batch B as

$$g_\varepsilon(\theta) = \frac{L(\theta + \varepsilon\Sigma_t^{1/2} z, B) - L(\theta - \varepsilon\Sigma_t^{1/2} z, B)}{2\varepsilon} \cdot \Sigma_t^{1/2} z,$$

for $z \sim \mathcal{N}(0, \mathrm{Id}_d)$ and Σ_t is an approximation to the diagonal of the inverse Hessian matrix, which gets updated simultanously during iteration.

Overall computation requires three forward passes of the model[7].

[7] Zhao et al., Second-Order Fine-Tuning without Pain for LLMs: A Hessian Informed Zeroth-Order Optimizer, 2024 [ZDY$^+$24]

HiZOO vs. MeZO

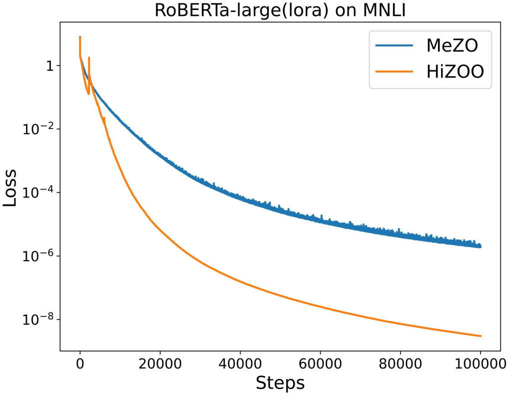


Figure 8: Loss evolution for LoRa training RoBERTa on the MultiNLI dataset[8], see figure 3[9]

[8] https://paperswithcode.com/dataset/multinli

[9] Zhao et al., Second-Order Fine-Tuning without Pain for LLMs: A Hessian Informed Zeroth-Order Optimizer, 2024 [ZDY+24]

Physics Informed Neural Networks (PINNs)

Physics Informed Neural Networks (PINNs)

Consider a partial differential equation of the form

$$\begin{aligned} \mathcal{D}[u(x), x] &= 0, \quad x \in \Omega \\ \mathcal{B}[u(x), x] &= 0, \quad x \in \partial\Omega \end{aligned}$$

where $\Omega \subseteq \mathbb{R}^d$, $\mathcal{D}$ is a differential operator and $\mathcal{B}$ is the boundary value operator.

Physics Informed Neural Networks (PINNs)

Consider a partial differential equation of the form

$$\mathcal{D}[u(x), x] = 0, \quad x \in \Omega$$
$$\mathcal{B}[u(x), x] = 0, \quad x \in \partial\Omega$$

where $\Omega \subseteq \mathbb{R}^d$, $\mathcal{D}$ is a differential operator and $\mathcal{B}$ is the boundary value operator. Let $u(x; \theta)$ be a neural network and minimize

$$\mathcal{L}(\theta) = \frac{1}{2n_{res}} \sum_{i=1}^{n_{res}} (\mathcal{D}[u(x_i^{res}; \theta), x_i^{res}])^2 + \frac{1}{2n_{bc}} \sum_{i=1}^{n_{bc}} (\mathcal{B}[u(x_i^{bc}; \theta), x_i^{bc}])^2$$

Key Observations[10]

- PINNs must be trained to near-zero loss to obtain an adequate solution (in ℓ_2 sense) to the PDE.
- The loss-landscape is ill-conditioned, i.e. the Hessian $H_{\mathcal{L}}(\theta)$ has a large condition number. In other words, the loss is very steep in some directions and very flat in others.
- Pre-conditioning with second-order information improves the conditioning significantly.

[10] Rathore et al., Challenges in Training PINNs: A Loss Landscape Perspective, 2024 [RLF$^+$24]

Spectral Density

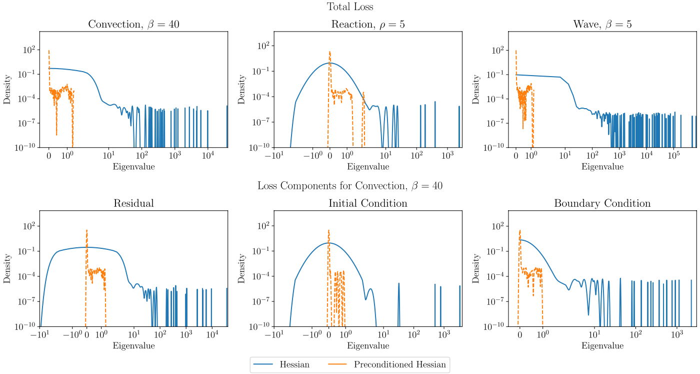


Figure 9: Optmizing with Adam + L-BFGS, Hessian spectral density after 41k iterations with and without preconditioning with quasi-Newton matrix[11]

[11] Rathore et al., Challenges in Training PINNs: A Loss Landscape Perspective, 2024 [RLF+24]

NysNewton-CG (NNCG)

Motivation: L-BFGS may stop early and leaves the loss under-optimized (see Section 7.1[12]).

[12] Rathore et al., Challenges in Training PINNs: A Loss Landscape Perspective, 2024 [RLF$^+$24]
[13] Frangella, et al., Randomized Nyström Preconditioning, 2023 [FTU23]

NysNewton-CG (NNCG)

Motivation: L-BFGS may stop early and leaves the loss under-optimized (see Section 7.1[12]).

Idea: Use a different approximate Newton update step, which allow for further progress.

[12] Rathore et al., Challenges in Training PINNs: A Loss Landscape Perspective, 2024 [RLF+24]

[13] Frangella, et al., Randomized Nyström Preconditioning, 2023 [FTU23]

NysNewton-CG (NNCG)

Motivation: L-BFGS may stop early and leaves the loss under-optimized (see Section 7.1[12]).
Idea: Use a different approximate Newton update step, which allow for further progress.

Introduce a positive-definite rank-r approximation (Nyström approximation)

$$H_{nys} = (H_{\mathcal{L}}S)(S^T H_{\mathcal{L}}S)^{\dagger}(H_{\mathcal{L}}S)^T, \quad S \in \mathbb{R}^{d\times r} \text{ standard normal,}$$

of the Hessian as preconditioner and use Conjugate Gradient. [13]

[12] Rathore et al., Challenges in Training PINNs: A Loss Landscape Perspective, 2024 [RLF$^+$24]
[13] Frangella, et al., Randomized Nyström Preconditioning, 2023 [FTU23]

NysNewton-CG (NNCG)

Algorithm 4 NysNewton-CG (NNCG)

input Initialization w_0, max. learning rate η, number of iterations K, preconditioner sketch size s, preconditioner update frequency F, damping parameter μ, CG tolerance ϵ, CG max. iterations M, backtracking parameters α, β

$d_{-1} = 0$

for $k = 0, \ldots, K-1$ **do**

  if k is a multiple of F **then**

    $[U, \hat{\Lambda}] =$ RandomizedNyströmApproximation$(H_L(w_k), s)$ ▷ Update Nyström preconditioner every F iterations

  end if

  $d_k =$ NyströmPCG$(H_L(w_k), \nabla L(w_k), d_{k-1}, U, \hat{\Lambda}, s, \mu, \epsilon, M)$ ▷ Damped Newton step $(H_L(w_k) + \mu I)^{-1} \nabla L(w_k)$

  $\eta_k =$ Armijo$(L, w_k, \nabla L(w_k), -d_k, \eta)$ ▷ Compute step size via line search

  $w_{k+1} = w_k - \eta_k d_k$ ▷ Update parameters

end for

Figure 10: NysNewton-CG optimizer[14]

[14] Rathore et al., Challenges in Training PINNs: A Loss Landscape Perspective, 2024 [RLF+24]

NysNewton-CG (NNCG)

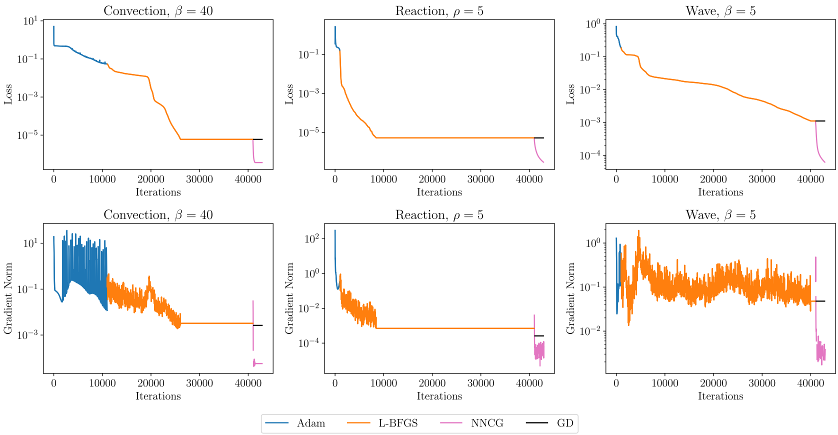


Figure 11: Loss evolution Adam + L-BFGS + NNCG[15]

[15]Rathore et al., Challenges in Training PINNs: A Loss Landscape Perspective, 2024 [RLF+24]

Influence Functions

Influence Functions

How to estimate the influence of single training points on model parameters or model output on test points?

Influence Functions

How to estimate the influence of single training points on model parameters or model output on test points?

Definition (Influence Function)

$$\mathcal{I}(z_t, z) = \nabla_\theta \ell(z_t, \theta)^T (H_\theta + \lambda I_d)^{-1} \nabla_\theta \ell(z, \theta)$$

Influence Functions

How to estimate the influence of single training points on model parameters or model output on test points?

Definition (Influence Function)

$$\mathcal{I}(z_t, z) = \nabla_\theta \ell(z_t, \theta)^T (H_\theta + \lambda I_d)^{-1} \nabla_\theta \ell(z, \theta)$$

Again, the bottleneck is the computation of the preconditioned gradients

$$(H_\theta + \lambda I_d)^{-1} \nabla_\theta \ell(z, \theta)$$

Recent Implementations

- EKFAC (Eigenvalue Corrected Kronecker Factorization)[GBA+23]:
  - Block inversion (per layer), Gauss-Newton approximation, Kronecker factorization
  - Implementation: pyDVL
  - Seminar talk, paper pill
- Nyström Preconditioned CG[FTU23]
  - Randomized low-rank approximation as preconditioner, Woodbury matrix identity
  - Implementation: pyDVL, implemented for the next minor release
- DataInf[KWWZ23]:
  - Block inversion (per layer), Gauss-Newton approximation, harmonic mean estimator, Woodbury matrix identity
  - Implementation: planned for next major release, github issue
  - paper pill

Software

Software

- Sophia: Second-order Clipped Stochastic Optimization[LLH+23]
  - https://github.com/Liuhong99/Sophia
  - Torch optimizer implementation, no package
  - paper pill
- MeZO: Memory-efficient Zeroth-order[MGN+23]
  - https://github.com/princeton-nlp/MeZO
  - HuggingFace trainer implementation, no package
- HiZOO: Hessian informed zeroth-order optimization[ZDY+24]
  - https://anonymous.4open.science/r/HiZOO-27F8
  - HuggingFace trainer implementation, no package
- NysNewton-CG[RLF+24]
  - https://anonymous.4open.science/r/opt_for_pinns-9246
  - Torch optimizer implementation, no package

Thank you!

Zachary Frangella, Joel A. Tropp, and Madeleine Udell.
Randomized Nyström Preconditioning.
SIAM Journal on Matrix Analysis and Applications, 44(2):718–752, June 2023.

Roger Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, Evan Hubinger, Kamilė Lukošiūtė, Karina Nguyen, Nicholas Joseph, Sam McCandlish, Jared Kaplan, and Samuel R. Bowman.
Studying Large Language Model Generalization with Influence Functions, August 2023.

- Yongchan Kwon, Eric Wu, Kevin Wu, and James Zou.
  DataInf: Efficiently Estimating Data Influence in LoRA-tuned LLMs and Diffusion Models, October 2023.
- Hong Liu, Zhiyuan Li, David Hall, Percy Liang, and Tengyu Ma.
  Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training, May 2023.
- Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D. Lee, Danqi Chen, and Sanjeev Arora.
  Fine-Tuning Language Models with Just Forward Passes, May 2023.

- Pratik Rathore, Weimu Lei, Zachary Frangella, Lu Lu, and Madeleine Udell.
  Challenges in Training PINNs: A Loss Landscape Perspective, February 2024.
- Yanjun Zhao, Sizhe Dang, Haishan Ye, Guang Dai, Yi Qian, and Ivor W. Tsang.
  Second-Order Fine-Tuning without Pain for LLMs:A Hessian Informed Zeroth-Order Optimizer, February 2024.