Hunting the Hessian: How and Why

Madeleine Udell

Management Science and Engineering
Stanford University

Joint work with
Zachary Frangella, Pratik Rathore, Shaghayegh Fazliani, Weimu Lei,
Wenzhi Gao, Ya-Chi Chu (Stanford),
Lu Lu (Yale), Yinyu Ye (Stanford), and Joel Tropp (Caltech)

February 12, 2025

Outline

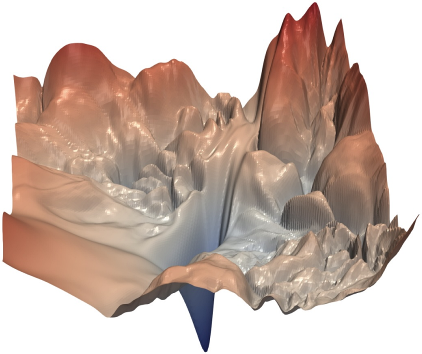
Optimization landscape

best methods for optimization depend on the landscape

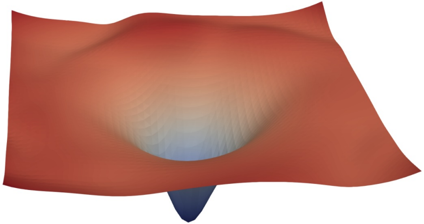
- local minima?
- saddle points?
- ill-conditioning?

what landscapes should we expect in modern deep learning?

Architectural choices govern optimization landscape



(a) without skip connections



(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

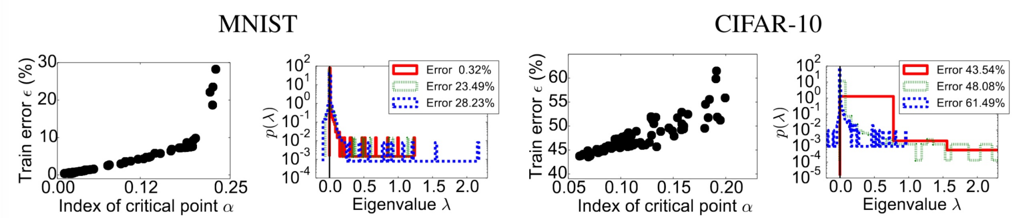
Source: Li, Xu, Taylor, et al., 2018

Saddle points vs local minima in deep learning

- **index** of critical point is

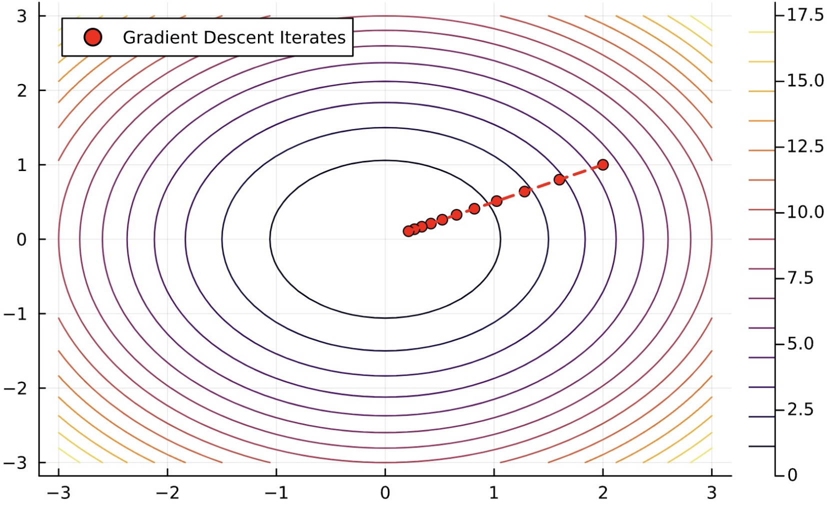
$$\# \text{ negative Hessian eigenvalues} = \text{directions of negative curvature}$$

- observation: all local minima are (nearly) global minima
- but there are plenty of saddles! (actually, might just be 0 eigenvalues. . .)

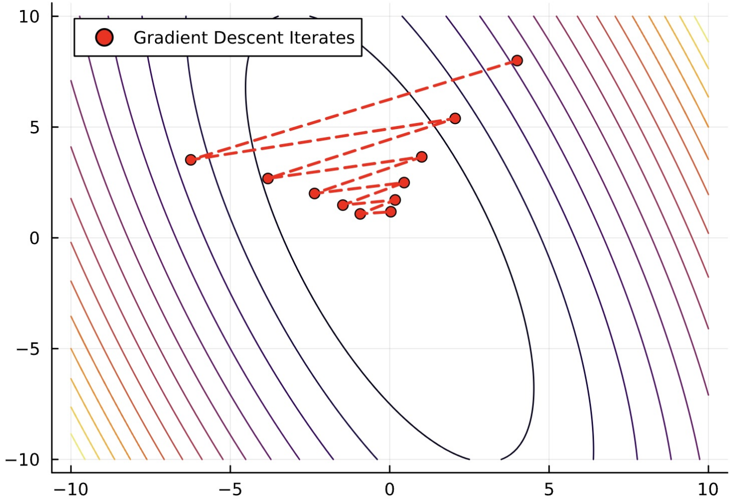


Source: MLP experiments from Dauphin, Pascanu, Gulcehre, et al., 2014

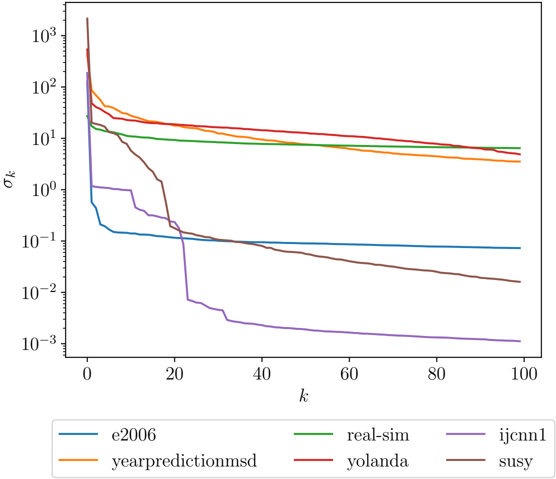
Gradient methods converge quickly on well-conditioned data



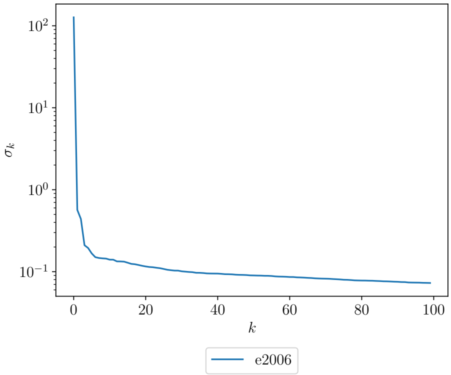
Gradient methods converge slowly on ill-conditioned data

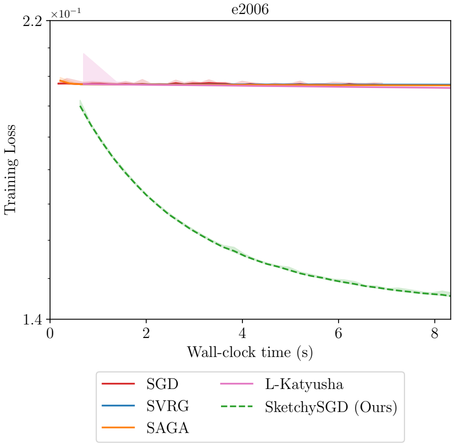


Ill-conditioning is common in ML data. . .



. . . and it makes optimization slower!





Landscape-aware optimization

agenda:

1. **local minima.** ignore them: they aren't so bad!
   - or try random restarts / judicious initialization ...
2. **ill-conditioning.** precondition!
3. **saddles.**
   - seek and follow directions of negative curvature? eg Clement Royer
   - actually, these are all small and not worth pursuing (Nicolas LeRoux, Jeremy Cohen)

Landscape-aware optimization

agenda:

1. **local minima.** ignore them: they aren't so bad!
   - or try random restarts / judicious initialization ...
2. **ill-conditioning.** precondition!
3. **saddles.**
   - seek and follow directions of negative curvature? eg Clement Royer
   - actually, these are all small and not worth pursuing (Nicolas LeRoux, Jeremy Cohen)

how to query and use the $p \times p$ Hessian of deep neural net?

Outline

Ill-conditioning is a property of the Hessian

for a twice-differentiable loss $f : \mathbf{R}^p \to \mathbf{R}$, write its Taylor expansion at a point w as

$$f(w+h) = f(w) + \nabla f(w)^T h + \frac{1}{2} h^T \nabla^2 f(w) h + \cdots$$

where $\nabla f(w)$ is the gradient and $\nabla^2 f(w)$ is the Hessian.

Ill-conditioning is a property of the Hessian

for a twice-differentiable loss $f : \mathbf{R}^p \to \mathbf{R}$, write its Taylor expansion at a point w as

$$f(w+h) = f(w) + \nabla f(w)^T h + \frac{1}{2} h^T \nabla^2 f(w) h + \cdots$$

where $\nabla f(w)$ is the gradient and $\nabla^2 f(w)$ is the Hessian.

convergence rate near a minimum depends on the condition number of local Hessian:

Definition (Condition number)

The *condition number* of a symmetric positive definite matrix A is

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$$

where $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$ are the largest and smallest eigenvalues of A.

(locally, approximately): $\kappa(\nabla^2 f(w)) =$ ratio of long to short axes of the level set

We can access $\nabla^2 f(w)$ with automatic differentiation!

automatic differentiation (AD) on $f : \mathbf{R}^p \to \mathbf{R}$ can compute gradients $\nabla f(w)$ and Hessian-vector products (hvp) $(\nabla^2 f(w))v$ in $O(p)$ time!

1. compute gradient with automatic differentiation (AD) $g(w) = \nabla f(w)$
2. define Hessian vector product with vector v

$$(\nabla^2 f(w))v = \nabla(g(w) \cdot v)$$

   and compute using AD on $g(w) \cdot v$ (Pearlmutter's trick)
3. cost: two passes of AD $\approx 4\times$ cost of function evaluation (usually, $O(p)$)

Low rank approximation via eigenvalues

given $A \in \mathbf{S}^p_+$ (symmetric positive definite), find the best rank-s approximation:

- compute the eigenvalue decomposition ($O(p^3)$ flops)

$$A = U \Lambda U^T$$

  with $\Lambda = \mathbf{diag}(\lambda_1, \ldots, \lambda_p)$, $\lambda_1 \geq \cdots \geq \lambda_p$, $UU^T = U^T U = I_p$,
- truncate to top s eigenvector/value pairs:

$$\hat{A} = U_s \Lambda_s U_s^T$$

  where
  - $U_s \in \mathbf{R}^{p \times s}$ is first s columns of $U \in \mathbf{R}^{p \times p}$, so $U_s^T U_s = I_s$
  - $\Lambda_s = \mathbf{diag}(\lambda_1, \ldots, \lambda_s)$,

Efficient eigs via randomized NLA

given $A \in \mathbf{S}_+^n$, find a good rank-s approximation:

- draw random Gaussian matrix $\Omega \in \mathbb{R}^{p \times s}$
- compute randomized linear sketch $Y = A\Omega$.

Efficient eigs via randomized NLA

given $A \in \mathbf{S}_+^n$, find a good rank-s approximation:

- draw random Gaussian matrix $\Omega \in \mathbb{R}^{p \times s}$
- compute randomized linear sketch $Y = A\Omega$.
- form *Nyström approximation* [Tropp, Yurtsever, Udell, et al. (2017)]

$$\hat{A}_{\text{nys}} = (A\Omega)(\Omega^T A\Omega)^\dagger (A\Omega)^T = Y(\Omega^T Y)^\dagger Y^T.$$

Efficient eigs via randomized NLA

given $A \in \mathbf{S}_+^n$, find a good rank-s approximation:

- draw random Gaussian matrix $\Omega \in \mathbb{R}^{p \times s}$
- compute randomized linear sketch $Y = A\Omega$.
- form *Nyström approximation* [Tropp, Yurtsever, Udell, et al. (2017)]

$$\hat{A}_{\text{nys}} = (A\Omega)(\Omega^T A\Omega)^\dagger (A\Omega)^T = Y(\Omega^T Y)^\dagger Y^T.$$

- in practice, construct apx eigs $\hat{A} = V\hat{\Lambda}V^T$ using tall-skinny QR, small SVD

Efficient eigs via randomized NLA

given $A \in \mathbf{S}_+^n$, find a good rank-s approximation:

- draw random Gaussian matrix $\Omega \in \mathbb{R}^{p \times s}$
- compute randomized linear sketch $Y = A\Omega$.
- form *Nyström approximation* [Tropp, Yurtsever, Udell, et al. (2017)]

$$\hat{A}_{\text{nys}} = (A\Omega)(\Omega^T A\Omega)^\dagger (A\Omega)^T = Y(\Omega^T Y)^\dagger Y^T.$$

- in practice, construct apx eigs $\hat{A} = V\hat{\Lambda}V^T$ using tall-skinny QR, small SVD

properties:

- total computation: s matvecs + $O(ps^2)$
- total storage: $O(ps)$
- $\hat{A}_{\text{nys}}$ is spd, **rank**$(\hat{A}_{\text{nys}}) \leq s$, and $\hat{A}_{\text{nys}} \preceq A$
- requires only matvecs with A, streaming ok.

Preconditioning a linear system

for any $P \succ 0$,

$$Ax = b \quad \Longleftrightarrow \quad \begin{aligned} P^{-1/2}Ax &= P^{-1/2}b \\ P^{-1/2}AP^{-1/2}z &= P^{-1/2}b \end{aligned}$$

where $x = P^{-1/2}z$.

Preconditioning a linear system

for any $P \succ 0$,

$$Ax = b \quad \Longleftrightarrow \quad \begin{aligned} P^{-1/2}Ax &= P^{-1/2}b \\ P^{-1/2}AP^{-1/2}z &= P^{-1/2}b \end{aligned}$$

where $x = P^{-1/2}z$.

- preconditioning works well when $\kappa(P^{-1/2}AP^{-1/2}) \ll \kappa(A)$

Preconditioning a linear system

for any $P \succ 0$,

$$Ax = b \quad \iff \quad \begin{aligned} P^{-1/2}Ax &= P^{-1/2}b \\ P^{-1/2}AP^{-1/2}z &= P^{-1/2}b \end{aligned}$$

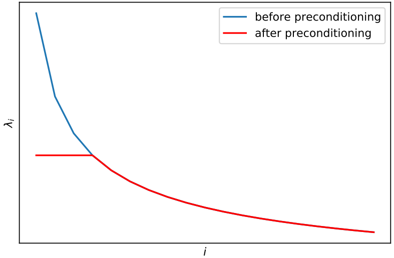
where $x = P^{-1/2}z$.

- preconditioning works well when $\kappa(P^{-1/2}AP^{-1/2}) \ll \kappa(A)$

An optimal low-rank preconditioner

- suppose $\lfloor A \rfloor_s = V_s \Lambda_s V_s^T$ is a best rank-s apx to $A \in \mathbf{S}_+^p$.
- the best preconditioner (e.g., for PCG) using this information is

$$P_\star = \frac{1}{\lambda_{s+1}} V_s(\Lambda_s)V_s^T + (I - V_s V_s^T)$$



Nyström preconditioner

Given a rank-s Nyström approximation

$$\hat{A}_{\text{nys}} = V \hat{\Lambda} V^T \qquad \approx \qquad A \in \mathbf{S}_+^p,$$

the *Nyström preconditioner* for $(A + \mu I)x = b$ is

$$P_{\text{nys}} = \frac{1}{\hat{\lambda}_s + \mu} V(\hat{\Lambda} + \mu I)V^T + (I - VV^T)$$

Nyström preconditioner

Given a rank-s Nyström approximation

$$\hat{A}_{\text{nys}} = V\hat{\Lambda}V^T \qquad \approx \qquad A \in \mathbf{S}_+^p,$$

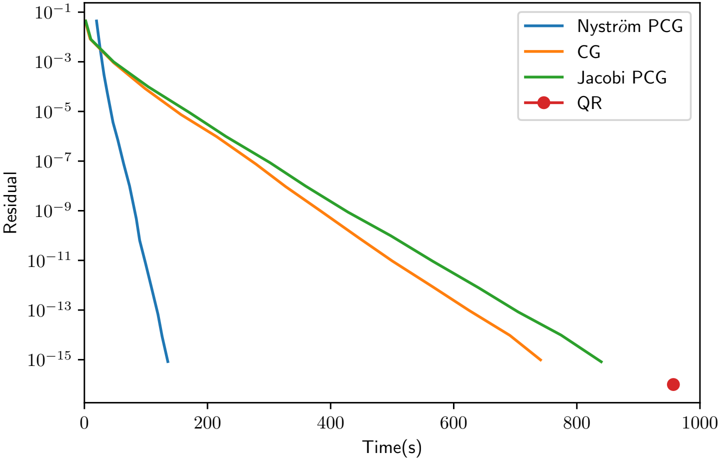
the *Nyström preconditioner* for $(A + \mu I)x = b$ is

$$P_{\text{nys}} = \frac{1}{\hat{\lambda}_s + \mu} V(\hat{\Lambda} + \mu I)V^T + (I - VV^T)$$

inverse can be applied in $O(ps)$:

$$P^{-1} = (\hat{\lambda}_s + \mu)V(\hat{\Lambda} + \mu I)^{-1}V^T + (I - VV^T)$$

Nyström preconditioner is fast!



Random features regression on YearMSD dataset $(463,715 \times 15,000)$. Regularization $\mu = 10^{-5}$; sketch size $s = 500$.

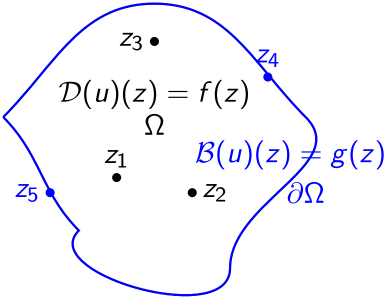
Outline

Physics-Informed Neural Networks (PINNs)

goal: solve PDE to find solution $u : \Omega \to \mathbf{R}$

$$\mathcal{D}(u)(z) = f(z), \quad z \in \Omega$$
$$\mathcal{B}(u)(z) = g(z), \quad z \in \partial\Omega,$$

where $\mathcal{D}$ is a differential operator, f is a forcing function, $\mathcal{B}$ is initial condition/boundary condition operator, and g is boundary function.

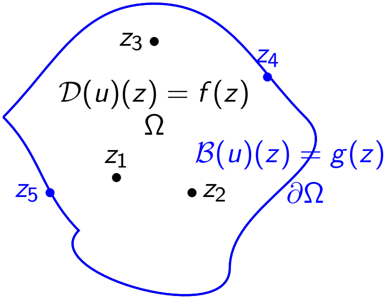


Physics-Informed Neural Networks (PINNs)

goal: solve PDE to find solution $u : \Omega \to \mathbf{R}$

$$\mathcal{D}(u)(z) = f(z), \quad z \in \Omega$$
$$\mathcal{B}(u)(z) = g(z), \quad z \in \partial\Omega,$$

where $\mathcal{D}$ is a differential operator, f is a forcing function, $\mathcal{B}$ is initial condition/boundary condition operator, and g is boundary function.



PINNs train a neural network $u_\theta(z)$ to approximate the PDE solution by minimizing a loss function that includes both data and physics-based terms

$$\frac{1}{N_{\mathrm{r}}} \sum_{i=1}^{N_{\mathrm{r}}} \|\mathcal{D}(u_\theta(z_i)) - f(z_i)\|^2 + \frac{1}{N_{\mathcal{B}}} \sum_{i=1}^{N_{\mathcal{B}}} \|\mathcal{B}(u_\theta(z_i)) - g(z_i)\|^2$$

PINNs suffer from under-optimization

- After training, gradient norm is typically on the order 10^{-2} or 10^{-3}
- L-BFGS stops early because PyTorch detects instability in the preconditioner
- Our proposal: fine-tune with NysNewton-CG (NNCG), i.e.,
  use Newton's method and solve linear system with NyströmPCG

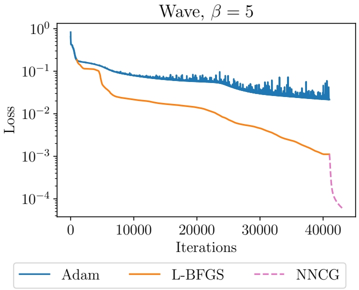


Figure: Even after running L-BFGS, the loss can be improved.

Preconditioners can improve conditioning

plot spectral density of PINN Hessian for different PDEs

- blue: original function
- orange: after preconditioning

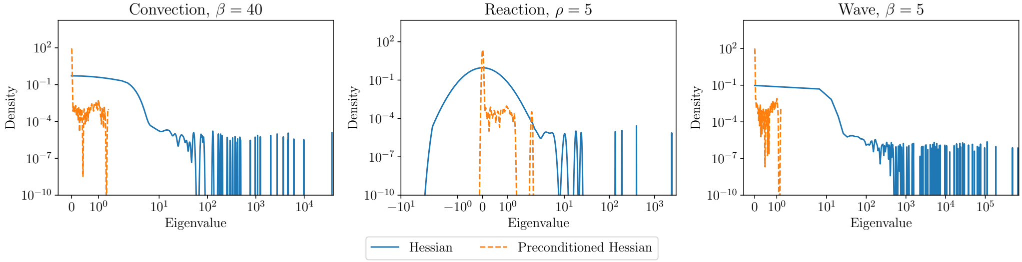
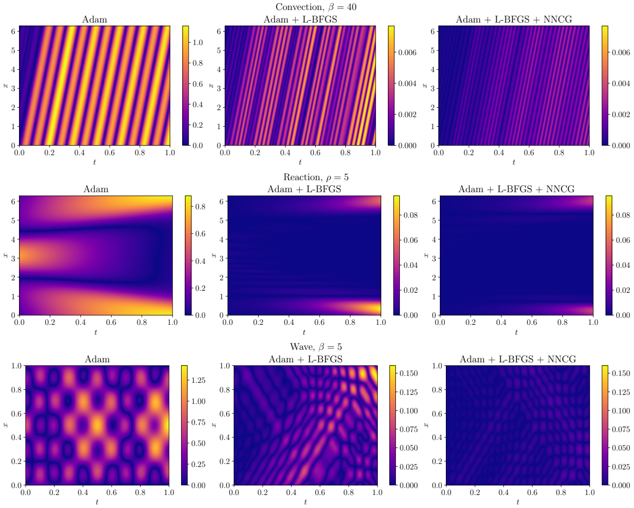


Figure: The total loss is ill-conditioned for all three PDEs.

Source: Approximate spectral density with kernel smoothing + stochastic trace estimation + Gaussian quadrature + Lanczos [Ubaru, Chen, and Saad (2017) and Yao, Gholami, Keutzer, et al. (2020)]

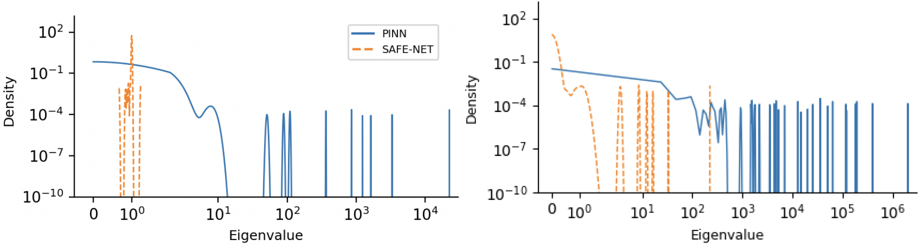
Preconditioned optimizers improve fits



Architectural choices can improve conditioning

plot spectral density of PINN Hessian for wave PDEs

- blue: standard MLP architecture
- orange: with SAFE-NET architecture (single layer with fourier features)

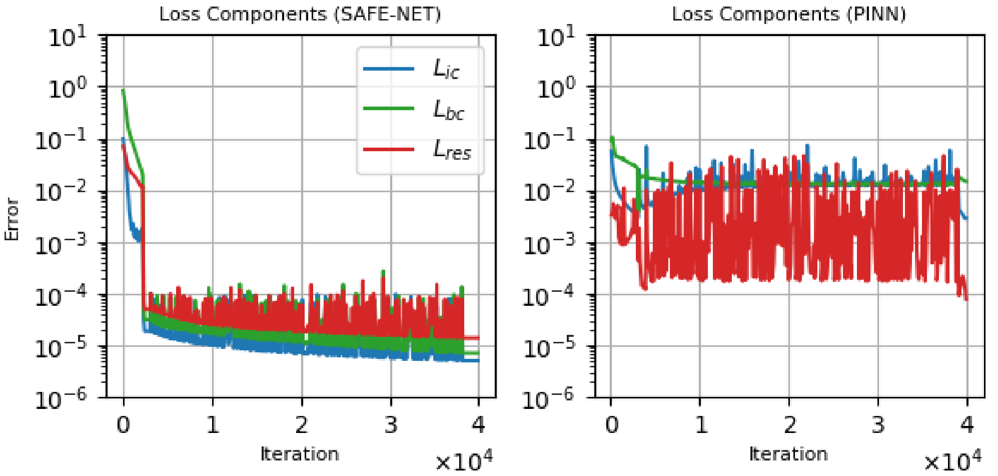


(a) Early into Training

(b) End of Training

Figure: Spectral density for the wave PDE using SAFE-NET and PINN at the early stages of training and at the end of training.

Well-conditioned architectures improve fits



(a) Wave

Outline

Stochastic optimization

consider the empirical risk minimization problem for $w \in \mathbf{R}^p$

$$\text{minimize} \quad \frac{1}{n}\sum_{i=1}^{n} f_i(w)$$

stochastic gradient method (SGD):

$$w \leftarrow w - \eta g \quad \text{where} \quad g \approx \nabla f(w)$$

works if $\mathbf{E}\, g = \nabla f(w)$

Stochastic quasi-Newton approximates the Hessian

Newton's method converges in one step on a deterministic quadratic problem:

$$w \leftarrow w - \eta H^{-1} g \quad \text{where} \quad g = \nabla f(w),\ H = \nabla^2 f(w)$$

Stochastic quasi-Newton approximates the Hessian

Newton's method converges in one step on a deterministic quadratic problem:

$$w \leftarrow w - \eta H^{-1} g \quad \text{where} \quad g = \nabla f(w),\ H = \nabla^2 f(w)$$

stochastic quasi-Newton method:

$$w \leftarrow w - \eta H^{-1} g \quad \text{where} \quad g \approx \nabla f(w),\ H \approx \nabla^2 f(w)$$

pros:

- faster convergence
- more robust to ill-conditioned problems (= all ML problems)
- easier to choose hyperparameters (learning rate η)

cons:

- $\nabla^2 f(x)$ is expensive to compute and apply

How to approximate $\nabla^2 f(x)$?

- from a data subsample
- from stale data
- by the secant condition (BFGS, L-BFGS)
- by diagonal approximation (Adam, AdaHessian)
- by block-diagonal kronecker approximation (Shampoo, KFAC, SENG, K-BFGS)
- by low rank approximation (SketchySGD)

Source: Erdogdu and Montanari, 2015, Shampoo (Anil, Gupta, Koren, et al., 2020; Gupta, Koren, & Singer, 2018), Roosta-Khorasani and Mahoney, 2019, Bollapragada, Byrd, and Nocedal, 2019, AdaHessian (Yao, Gholami, Shen, et al., 2021), R-SSN (Meng, Vaswani, Laradji, et al., 2020), KFAC (Grosse & Martens, 2016), SENG (Yang, Xu, Wen, et al., 2020), Goldfarb, Ren, and Bahamou, 2020, SketchySGD (Frangella, Rathore, Zhao, et al., 2023; Rathore, Frangella, & Udell, 2023)

Subsampling the Hessian

Subsampled loss where $S \subseteq \{1, \cdots, m\}$ is random uniform:

$$\tilde{f}(w) = \frac{1}{|S|} \sum_{i \in S} f_i(w)$$

Hessian of subsampled loss is

$$\nabla^2 \tilde{f}(w) = \frac{1}{|S|} \sum_{i \in S} \nabla^2 f_i(w)$$

Subsampling the Hessian

Subsampled loss where $S \subseteq \{1, \cdots, m\}$ is random uniform:

$$\tilde{f}(w) = \frac{1}{|S|} \sum_{i \in S} f_i(w)$$

Hessian of subsampled loss is

$$\nabla^2 \tilde{f}(w) = \frac{1}{|S|} \sum_{i \in S} \nabla^2 f_i(w)$$

- Subsampled Newton (SSN) method uses *inverse of regularized hvp*

  $$w_{k+1} = w_k - \eta_k \left(\nabla^2 \tilde{f}(x_k) + \rho I \right)^{-1} \nabla \tilde{f}(w_k)$$

  where $\rho > 0$ ensures invertibility and stability.

Subsampling the Hessian

Subsampled loss where $S \subseteq \{1, \cdots, m\}$ is random uniform:

$$\tilde{f}(w) = \frac{1}{|S|} \sum_{i \in S} f_i(w)$$

Hessian of subsampled loss is

$$\nabla^2 \tilde{f}(w) = \frac{1}{|S|} \sum_{i \in S} \nabla^2 f_i(w)$$

- Subsampled Newton (SSN) method uses *inverse of regularized hvp*

  $$w_{k+1} = w_k - \eta_k \left(\nabla^2 \tilde{f}(x_k) + \rho I\right)^{-1} \nabla \tilde{f}(w_k)$$

  where $\rho > 0$ ensures invertibility and stability.
- Can replace $\nabla^2 \tilde{f}(w)$ with a low-rank (e.g., Nyström) approximation $\implies$
- Can implement inverse with Woodbury formula in $O(p|S|^2)$ time

PROMISE algorithms

PROMISE (**Pr**econditioned Stochastic **O**ptimization **M**ethods by **I**ncorporating **S**calable Curvature **E**stimates): preconditioned versions of SGD, SVRG, SAGA, and Katyusha with default hyperparameters that work out-of-the-box, with provable linear convergence at improved rates.

Algorithm	Base Algorithm	Variance reduction	Acceleration	Stochastic gradients only?
SketchySGD	SGD	✗	✗	✓
SketchySVRG	SVRG	✓	✗	✗
SketchySAGA	b-nice SAGA	✓	✗	✓
SketchyKatyusha	Loopless Katyusha	✓	✓	✗

- SketchySGD (improves SGD): https://arxiv.org/abs/2211.08597
- PROMISE (improves SVRG, SAGA, Katyusha): https://arxiv.org/abs/2309.02014

Comparison of preconditioned stochastic gradient methods

Algorithm	b_g	b_H/ Sketch size	Lazy preconditioner updates	Fast local-linear convergence	Source
SketchySGD	$\tau_\star^\nu$	$\tilde{\mathcal{O}}(d_{\text{eff}}^\nu(A))$	✓	✗	PROMISE
SketchySVRG	$\tau_\star^\nu$	$\tilde{\mathcal{O}}(d_{\text{eff}}^\nu(A))$	✓	✓	PROMISE
Subsampled Newton	Exponentially increasing	$\tilde{\mathcal{O}}(\kappa/\epsilon^2)$	✗	✓	(Roosta-Khorasani & Mahoney, 2019)
Newton Sketch	Full	$\tilde{\mathcal{O}}(d_{\text{eff}}^\nu(A))$	✗	✓	(Lacotte, Wang, & Pilanci, 2021)
Stochastic Variance Reduced Newton	$\tilde{\mathcal{O}}(\kappa)$	$\tilde{\mathcal{O}}(\kappa/\epsilon^2)$	✗	✓	(Dereziński, 2022)
SLBFGS	Constant	Constant	✗	✗	(Moritz, Nishihara, & Jordan, 2016)
Progressive Batching L-BFGS	Increasing	Increasing	✗	✗	(Bollapra-gada, Nocedal, Mudigere, et al., 2018)

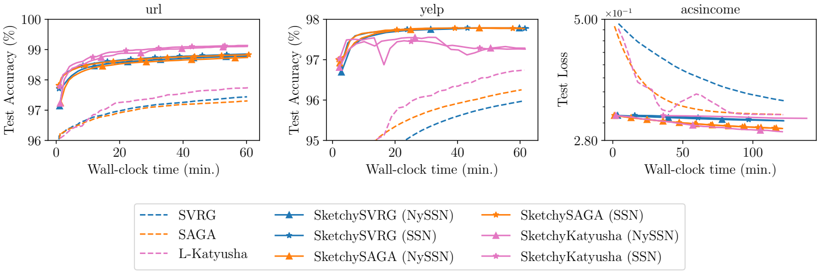
Table: Methods for finite sum optimization when F is a GLM.

Comparison of preconditioned stochastic gradient methods

Algorithm	b_g	b_H/ Sketch size	Lazy preconditioner updates	Fast local-linear convergence	Source
SketchySGD	$\tau_\star^\nu$	$\tilde{\mathcal{O}}(d_{\text{eff}}^\nu(A))$	✓	✗	PROMISE
SketchySVRG	$\tau_\star^\nu$	$\tilde{\mathcal{O}}(d_{\text{eff}}^\nu(A))$	✓	✓	PROMISE
Subsampled Newton	Exponentially increasing	$\tilde{\mathcal{O}}(\kappa/\epsilon^2)$	✗	✓	(Roosta-Khorasani & Mahoney, 2019)
Newton Sketch	Full	$\tilde{\mathcal{O}}(d_{\text{eff}}^\nu(A))$	✗	✓	(Lacotte, Wang, & Pilanci, 2021)
Stochastic Variance Reduced Newton	$\tilde{\mathcal{O}}(\kappa)$	$\tilde{\mathcal{O}}(\kappa/\epsilon^2)$	✗	✓	(Dereziński, 2022)
SLBFGS	Constant	Constant	✗	✗	(Moritz, Nishihara, & Jordan, 2016)
Progressive Batching L-BFGS	Increasing	Increasing	✗	✗	(Bollapragada, Nocedal, Mudigere, et al., 2018)

Table: Methods for finite sum optimization when F is a GLM.

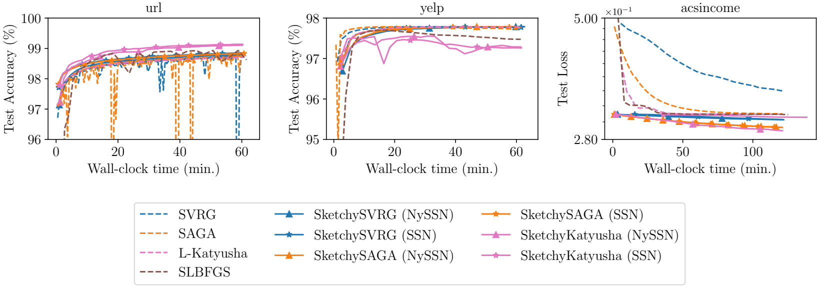
Showcase experiments (competitors not tuned)



random features regression; l^2-regularized logistic regression on url and yelp and ridge regression on acsincome

PROMISE methods provide better generalization in less time!

Showcase experiments (competitors tuned)



random features regression; l^2-regularized logistic regression on url and yelp and ridge regression on acsincome

Competitors perform better after tuning; but tuning is slow.

Outline

Gradient methods with online preconditioning

For smooth, strongly convex optimization,

$$x^{k+1} = x^k - P\nabla f(x^k) \implies f(x^{k+1}) - f(x^\star) \leq (1 - \frac{1}{\kappa_P})[f(x^k) - f(x^\star)]$$

Gradient methods with online preconditioning

For smooth, strongly convex optimization,

$$x^{k+1} = x^k - P\nabla f(x^k) \implies f(x^{k+1}) - f(x^\star) \leq (1 - \tfrac{1}{\kappa_P})[f(x^k) - f(x^\star)]$$

Can we learn a preconditioner P with $\kappa_P \ll \kappa$ during gradient descent?

$$\begin{aligned} \text{Gradient descent} \quad & x^{k+1} = x^k - P_k \nabla_x f(x^k) \\ \text{Preconditioner update} \quad & P_{k+1} = \text{Learn}(P_k, x^k) \end{aligned}$$

Gradient methods with online preconditioning

For smooth, strongly convex optimization,

$$x^{k+1} = x^k - P\nabla f(x^k) \implies f(x^{k+1}) - f(x^\star) \le (1 - \tfrac{1}{\kappa_P})[f(x^k) - f(x^\star)]$$

Can we learn a preconditioner P with $\kappa_P \ll \kappa$ during gradient descent?

$$\begin{aligned} \text{Gradient descent} \quad & x^{k+1} = x^k - P_k \nabla_x f(x^k) \\ \text{Preconditioner update} \quad & P_{k+1} = P_k - \eta \nabla_P \ell_{x^k}(P_k) \end{aligned}$$

Yes! By gradient descent on a surrogate $\ell_x(P)$

- invented 25 years ago [Almeida, Langlois, Amaral, et al. (1999)] and re-discovered as hypergradient descent [Baydin, Cornish, Rubio, et al. (2018)]
- good performance after tuning, but often unstable and almost no theory

Optimize the convergence rate with online learning

Better $P \Rightarrow$ smaller condition number $\kappa_P \Rightarrow$ better contraction factor

$$f(x - P\nabla f(x)) - f(x^\star) \leq (1 - \frac{1}{\kappa_P})[f(x) - f(x^\star)]$$

Optimize the convergence rate with online learning

Better $P \Leftarrow$ smaller condition number $\kappa_P \Leftarrow$ better contraction factor

$$\ell_x(P) = \frac{f(x - P\nabla f(x)) - f(x^\star)}{f(x) - f(x^\star)}$$

Optimize the convergence rate with online learning

Better $P \Leftarrow$ smaller condition number $\kappa_P \Leftarrow$ better contraction factor

$$\ell_x(P) = \frac{f(x - P\nabla f(x)) - f(x^\star)}{f(x) - f(x^\star)}$$

$$\frac{f(x^{K+1}) - f(x^\star)}{f(x^1) - f(x^\star)} = \prod_{k=1}^{K} \frac{f(x^{k+1}) - f(x^\star)}{f(x^k) - f(x^\star)} = \prod_{k=1}^{K} \ell_{x^k}(P_k) \leq \left(\frac{1}{K} \sum_{k=1}^{K} \ell_{x^k}(P_k)\right)^K$$

Optimize the convergence rate with online learning

Better $P \Leftarrow$ smaller condition number $\kappa_P \Leftarrow$ better contraction factor

$$\ell_x(P) = \frac{f(x - P\nabla f(x)) - f(x^\star)}{f(x) - f(x^\star)}$$

$$\frac{f(x^{K+1}) - f(x^\star)}{f(x^1) - f(x^\star)} = \prod_{k=1}^K \frac{f(x^{k+1}) - f(x^\star)}{f(x^k) - f(x^\star)} = \prod_{k=1}^K \ell_{x^k}(P_k) \leq \left(\frac{1}{K}\sum_{k=1}^K \ell_{x^k}(P_k)\right)^K$$

- $P_{k+1} = P_k - \eta \nabla_P \ell_{x^k}(P_k)$ is online gradient descent

Optimize the convergence rate with online learning

Better $P \Leftarrow$ smaller condition number $\kappa_P \Leftarrow$ better contraction factor

$$\ell_x(P) = \frac{f(x - P\nabla f(x)) - f(x^\star)}{f(x) - f(x^\star)}$$

$$\frac{f(x^{K+1}) - f(x^\star)}{f(x^1) - f(x^\star)} = \prod_{k=1}^K \frac{f(x^{k+1}) - f(x^\star)}{f(x^k) - f(x^\star)} = \prod_{k=1}^K \ell_{x^k}(P_k) \le \left(\frac{1}{K}\sum_{k=1}^K \ell_{x^k}(P_k)\right)^K$$

- $P_{k+1} = P_k - \eta \nabla_P \ell_{x^k}(P_k)$ is online gradient descent
- $\sum_{k=1}^K \ell_{x^k}(P_k) \le \sum_{k=1}^K \ell_{x^k}(P^\star) + O(\sqrt{K})$ guaranteed by online learning

Since $\ell_{x^k}(P^\star) \le 1 - \frac{1}{\kappa^\star}$, combining these relations gives

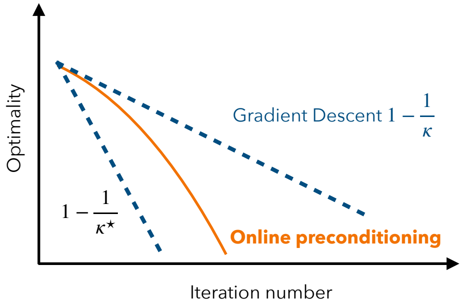
$$\frac{f(x^{K+1}) - f(x^\star)}{f(x^1) - f(x^\star)} \le \left(1 - \frac{1}{\kappa^\star} + O\left(\frac{1}{\sqrt{K}}\right)\right)^K \approx \left(1 - \frac{1}{\kappa^\star}\right)^K,$$

an online acceleration of gradient descent!

Convergence behavior

$$x^{k+1} = x^k - P_k \nabla_x f(x^k)$$
$$P_{k+1} = P_k - \eta \nabla_P \ell_{x^k}(P_k)$$

What does $\nabla_P \ell_{x^k}(P_k)$ look like?



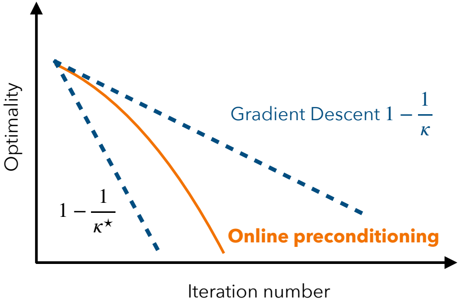
Convergence behavior

$$x^{k+1} = x^k - P_k \nabla_x f(x^k)$$

$$P_{k+1} = P_k - \eta \nabla_P \ell_{x^k}(P_k)$$

What does $\nabla_P \ell_{x^k}(P_k)$ look like?

- $\nabla \ell_x(P) = \frac{\nabla f(x - P\nabla f(x))\nabla f(x)^\top}{f(x) - f(x^\star)}$ or



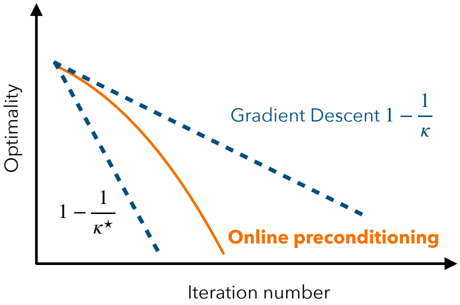
Convergence behavior

$$x^{k+1} = x^k - P_k \nabla_x f(x^k)$$

$$P_{k+1} = P_k - \eta \nabla_P \ell_{x^k}(P_k)$$

What does $\nabla_P \ell_{x^k}(P_k)$ look like?

- $\nabla \ell_x(P) = \frac{\nabla f(x - P\nabla f(x)) \nabla f(x)^\top}{f(x) - f(x^\star)}$ or
- $\nabla \ell_x(P) = \frac{\nabla f(x - P\nabla f(x)) \circ \nabla f(x)}{f(x) - f(x^\star)}$ or

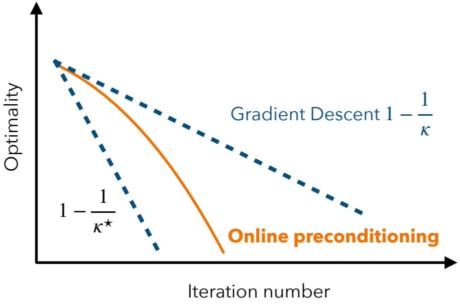


Convergence behavior

$$x^{k+1} = x^k - P_k \nabla_x f(x^k)$$
$$P_{k+1} = P_k - \eta \nabla_P \ell_{x^k}(P_k)$$

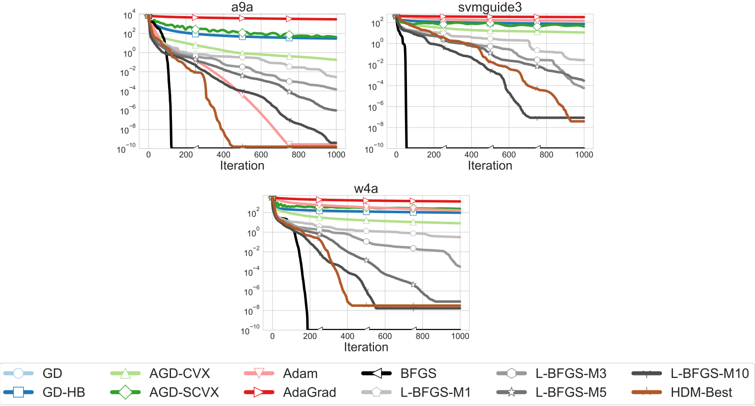
What does $\nabla_P \ell_{x^k}(P_k)$ look like?

- $\nabla \ell_x(P) = \frac{\nabla f(x - P\nabla f(x)) \nabla f(x)^\top}{f(x) - f(x^\star)}$ or
- $\nabla \ell_x(P) = \frac{\nabla f(x - P\nabla f(x)) \circ \nabla f(x)}{f(x) - f(x^\star)}$ or
- $\nabla \ell_x(P) = \frac{\nabla f(x - P\nabla f(x)) \circ \nabla f(x)}{\|\nabla f(x)\|^2}$



Numerical experiments

HDM-best includes heavy-ball momentum



Algorithm	SVM	Log. Reg
GD	5	2
GD-HB	9	7
AGD-CVX	8	3
AGD-SCVX	7	6
Adam	26	11
AdaGrad	9	8
L-BFGS-M1	13	11
L-BFGS-M3	20	14
L-BFGS-M5	26	16
L-BFGS-M10	31	18
BFGS	32	26
OSGM	32	21

solved instances in LIBSVM

On deterministic convex problems

- comparable performance to L-BFGS-M5/M10
- same memory as L-BFGS-1 and cheaper iterations

Conclusion

does your optimization suffer from ill-conditioning?

Conclusion

does your optimization suffer from ill-conditioning?

preconditioners can help!

Conclusion

does your optimization suffer from ill-conditioning?

preconditioners can help!

- spectral preconditioning is feasible at large scale
  - autodiff from Hessian-vector products
  - randomized Nyström approximation

Conclusion

does your optimization suffer from ill-conditioning?

preconditioners can help!

- spectral preconditioning is feasible at large scale
  - autodiff from Hessian-vector products
  - randomized Nyström approximation
- online scaled gradient method
  - provably competes with the best offline methods
  - flexible framework can improve many optimization algorithms

Where can I learn more?

- randomized Nyström approximation to a psd matrix: https://arxiv.org/abs/1706.05736 NeurIPS 2017
- Nyström PCG to solve $Ax = b$: https://arxiv.org/abs/2110.02820 SIMAX 2023
- almost-second-order stochastic optimization:
  - SketchySGD (improves SGD): https://arxiv.org/abs/2211.08597 SIMODS 2024
  - PROMISE (improves SVRG etc.): https://arxiv.org/abs/2309.02014 JMLR 2024
  - NNCG for PINNs: https://arxiv.org/abs/2402.01868 ICML 2024
  - SAFE-NET for PINNs: http://arxiv.org/abs/2502.07209
- online scaled gradient method: https://arxiv.org/abs/2411.01803