Hunting the Hessian: Randomized Methods

Madeleine Udell

Management Science and Engineering
Stanford University

Joint work with
Zachary Frangella, Pratik Rathore, Shaghayegh Fazliani, Weimu Lei,
Shipu Zhao (Cornell→Amazon), Theo Diamandis (MIT),
Bartolomeo Stellato (Princeton), Lu Lu (Yale), and Joel Tropp (Caltech)

January 12, 2026

Outline

Preconditioning

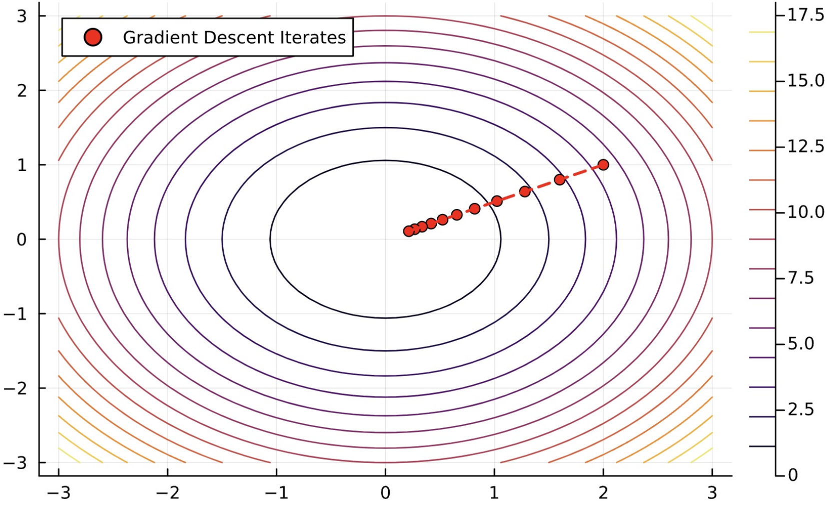
Nyström preconditioning

NysADMM

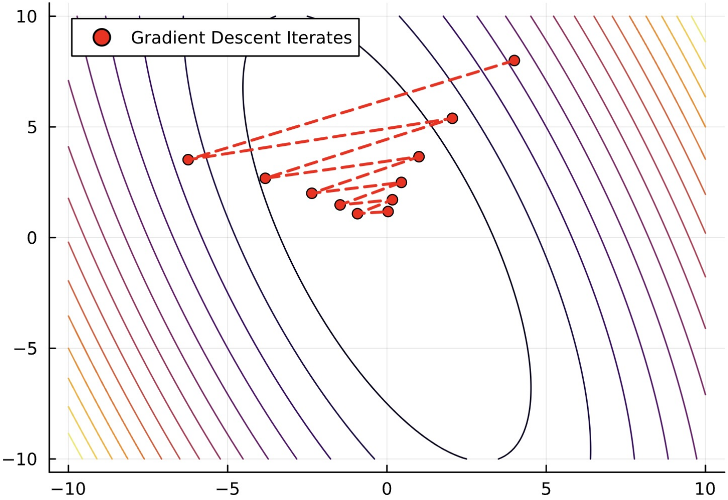
Optimization for deep learning

PINNs

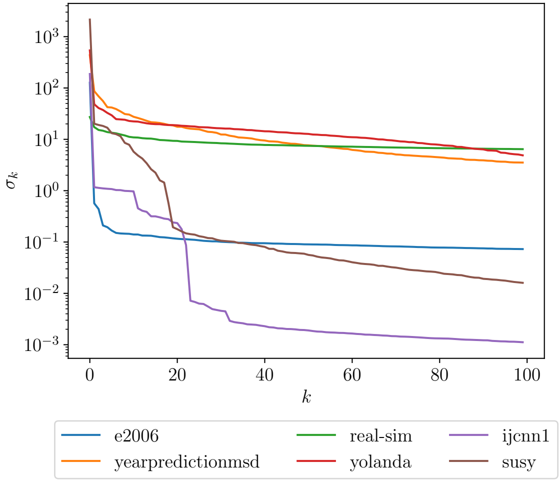
Gradient methods converge quickly on well-conditioned data



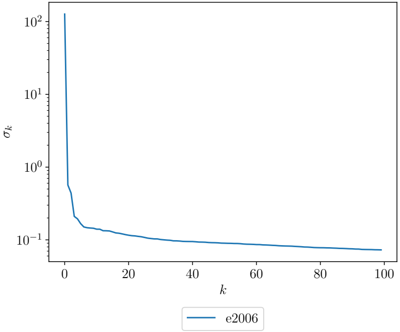
Gradient methods converge slowly on ill-conditioned data

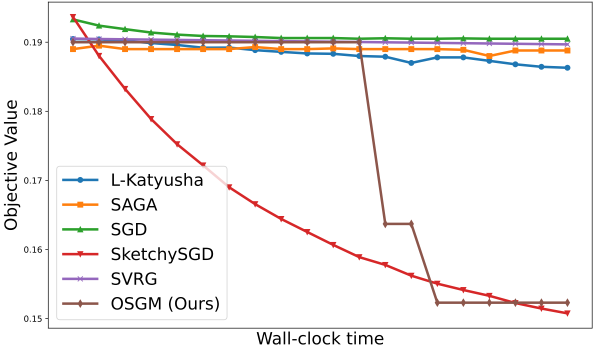


Ill-conditioning is common in ML data...



. . . and it makes optimization slower!





Recap: convergence analysis for gradient descent

$$\text{minimize} \quad f(x)$$

recall: we say (twice-differentiable) f is μ-strongly convex and L-smooth if

$$\mu I \preceq \nabla^2 f(x) \preceq LI$$

recall: if f is μ-strongly convex and L-smooth, gradient descent converges linearly

$$f(x^{K+1}) - p^\star \leq \frac{Lc^K}{2} \|x^1 - x^\star\|^2,$$

where $c = (\frac{\kappa-1}{\kappa+1})^2$, $\kappa = \frac{L}{\mu} \geq 1$ is condition number

Recap: convergence analysis for gradient descent

$$\text{minimize} \quad f(x)$$

recall: we say (twice-differentiable) f is μ-strongly convex and L-smooth if

$$\mu I \preceq \nabla^2 f(x) \preceq LI$$

recall: if f is μ-strongly convex and L-smooth, gradient descent converges linearly

$$f(x^{K+1}) - p^\star \leq \frac{Lc^K}{2}\|x^1 - x^\star\|^2,$$

where $c = (\frac{\kappa-1}{\kappa+1})^2$, $\kappa = \frac{L}{\mu} \geq 1$ is condition number $\implies$ want $\kappa \approx 1$

Recap: convergence analysis for gradient descent

$$\text{minimize} \quad f(x)$$

recall: we say (twice-differentiable) f is μ-strongly convex and L-smooth if

$$\mu I \preceq \nabla^2 f(x) \preceq LI$$

recall: if f is μ-strongly convex and L-smooth, gradient descent converges linearly

$$f(x^{K+1}) - p^\star \leq \frac{Lc^K}{2} \|x^1 - x^\star\|^2,$$

where $c = (\frac{\kappa-1}{\kappa+1})^2$, $\kappa = \frac{L}{\mu} \geq 1$ is condition number $\implies$ want $\kappa \approx 1$

idea: can we minimize another function with $\kappa \approx 1$ whose solution will tell us the minimizer of f?

Preconditioning

for invertible D, the two problems

$$\text{minimize} \quad f(x) \quad \text{and} \quad \text{minimize} \quad f(Dz)$$

have solutions related by $x^\star = Dz^\star$

Preconditioning

for invertible D, the two problems

$$\text{minimize} \quad f(x) \quad \text{and} \quad \text{minimize} \quad f(Dz)$$

have solutions related by $x^\star = Dz^\star$

- gradient of $f(Dz)$ is $D^T \nabla f(Dz)$
- the second derivative (Hessian) of $f(Dz)$ is $D^T \nabla^2 f(Dz) D$

Preconditioning

for invertible D, the two problems

$$\text{minimize} \quad f(x) \quad \text{and} \quad \text{minimize} \quad f(Dz)$$

have solutions related by $x^\star = Dz^\star$

- gradient of $f(Dz)$ is $D^T \nabla f(Dz)$
- the second derivative (Hessian) of $f(Dz)$ is $D^T \nabla^2 f(Dz) D$

a gradient step on $f(Dz)$ with step-size $t > 0$ is

$$\begin{aligned} z^+ &= z - tD^T \nabla f(Dz) \\ Dz^+ &= Dz - tDD^T \nabla f(Dz) \\ x^+ &= x - tDD^T \nabla f(x) \end{aligned}$$

this iteration is *preconditioned gradient descent* (PGD) with preconditioner $P = DD^T$.

Outline

Preconditioning

Nyström preconditioning

NysADMM

Optimization for deep learning

PINNs

Preconditioning a linear system

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

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

Low rank approximation via eigenvalues

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

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

$$A = \sum_{i=1}^{p} \lambda_i u_i u_i^T = U \Lambda U^T$$

with $\lambda_1 \geq \cdots \geq \lambda_p$, $\Lambda = \mathbf{diag}(\lambda_1, \ldots, \lambda_p)$, $u_i^T u_j = \delta_{ij}$.

- truncate to top s eigenvector/value pairs:

$$\hat{A} = \sum_{i=1}^{s} \lambda_i u_i u_i^T = U_s \Lambda_s U_s^T$$

where Λ_s and U_s are truncated versions of Λ and U.

Efficient eigs via randomized NLA

given $A \in \mathbf{S}_+^p$, 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 $A \in \mathbf{S}_+^p$, 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, and Cevher (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 $A \in \mathbf{S}_+^p$, 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, and Cevher (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 $A \in \mathbf{S}_+^p$, 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, and Cevher (2017)]

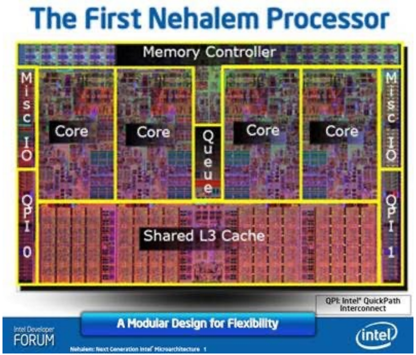
$$\hat{A}_{\text{nys}} = (A\Omega)(\Omega^T A\Omega)^\dagger (A\Omega)^T = Y(\Omega^T Y)^\dagger Y^T.$$

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

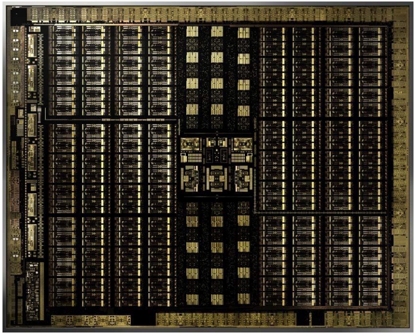
properties:

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

Speed depends on hardware



CPU: complex, sequential tasks



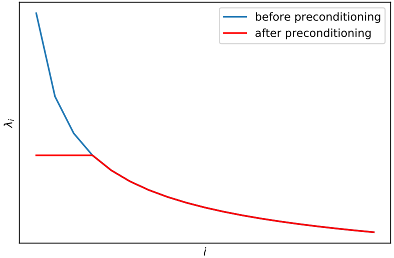
GPU: simple, parallel tasks

- traditional matrix decompositions: hopelessly serial (e.g., Gaussian elimination)
- randNLA: naturally parallel (mostly matvecs)

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

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

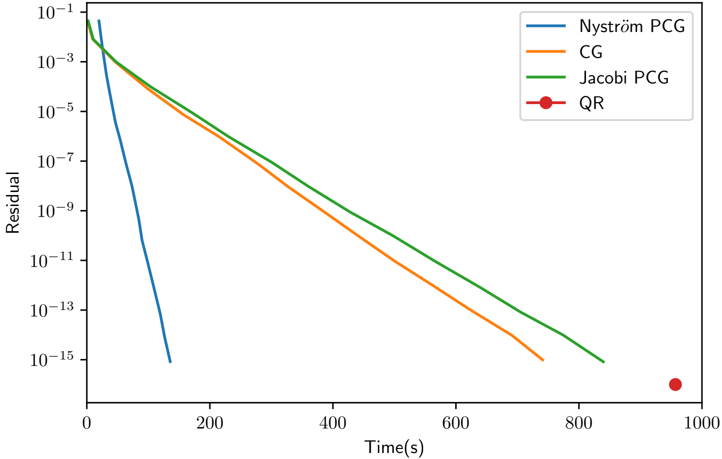
the *Nyström preconditioner* for $(A + \mu I)x = b$ is

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

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 $s = 500$.

Low rank approximation for faster optimization

randNLA allows approximate inverse of p$\times$p matrix A in $\mathcal{O}(p)$ time
$\implies$ can improve conditioning for many optimization problems.

Low rank approximation for faster optimization

randNLA allows approximate inverse of p×p matrix A in $\mathcal{O}(p)$ time
$\implies$ can improve conditioning for many optimization problems. e.g.,

1. Nyström PCG to solve $Ax = b$
   - randomized low rank approximation as preconditioner

Low rank approximation for faster optimization

randNLA allows approximate inverse of p×p matrix A in $\mathcal{O}(p)$ time
$\implies$ can improve conditioning for many optimization problems. e.g.,

1. Nyström PCG to solve $Ax = b$
   - randomized low rank approximation as preconditioner

Low rank approximation for faster optimization

randNLA allows approximate inverse of p$\times$p matrix A in $\mathcal{O}(p)$ time
$\implies$ can improve conditioning for many optimization problems. e.g.,

1. Nyström PCG to solve $Ax = b$
   - randomized low rank approximation as preconditioner
2. NysADMM for composite optimization minimize $f(Ax) + g(x)$, e.g.,
   - lasso
   - regularized logistic regression
   - support vector machine

   randNLA beats SOTA solver for all these problems!

Low rank approximation for faster optimization

randNLA allows approximate inverse of p×p matrix A in $\mathcal{O}(p)$ time
$\implies$ can improve conditioning for many optimization problems. e.g.,

1. Nyström PCG to solve $Ax = b$
   - randomized low rank approximation as preconditioner
2. NysADMM for composite optimization minimize $f(Ax) + g(x)$, e.g.,
   - lasso
   - regularized logistic regression
   - support vector machine

   randNLA beats SOTA solver for all these problems!
3. approximate Newton methods for deep learning and stochastic optimization
   low rank approximation for Newton system improves
   - robustness (vs first-order methods) and
   - speed (vs other quasi-Newton methods)

Outline

Preconditioning

Nyström preconditioning

NysADMM

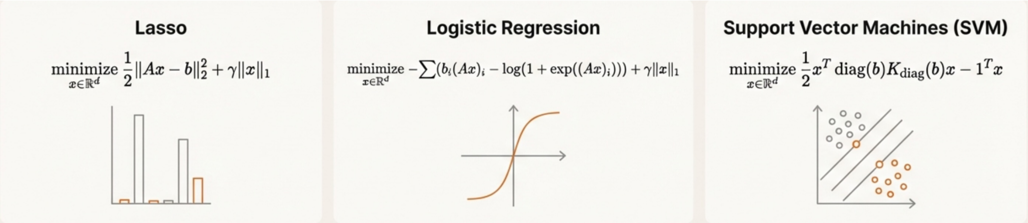
Optimization for deep learning

PINNs

Composite optimization

$$\text{minimize} \quad \ell(Ax) + r(x)$$

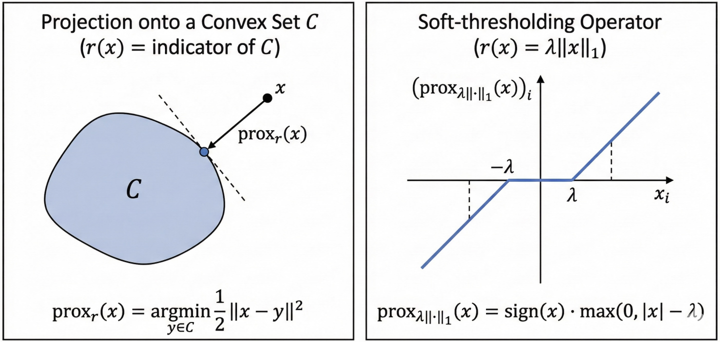
- $A : \mathbf{R}^p \to \mathbf{R}^m$ linear
- $\ell : \mathbf{R}^m \to \mathbf{R}$ smooth
- $r : \mathbf{R}^p \to \mathbf{R}$ *proxable*



Proximal operators

$r : \mathbf{R}^p \to \mathbf{R}$ is called *proxable* if it is easy to compute the *proximal operator*

$$\mathbf{prox}_r(x) := \operatorname*{argmin}_y r(y) + \frac{1}{2}\|x - y\|^2$$



Alternating Directions Method of Multipliers

Algorithm ADMM

Input: loss function $\ell \circ A$, regularization r, stepsize ρ,
initial z^0, $u^0 = 0$
for $k = 0, 1, \ldots$ **do**
    $x^{k+1} = \text{argmin}_x \{\ell(Ax) + \frac{\rho}{2}\|x - z^k + u^k\|_2^2\}$
    $z^{k+1} = \text{argmin}_z \{r(z) + \frac{\rho}{2}\|x^{k+1} - z + u^k\|_2^2\}$
    $u^{k+1} = u^k + x^{k+1} - z^{k+1}$
return $x_\star$ (nearly) minimizing $\ell(Ax) + r(x)$

Alternating Directions Method of Multipliers

Algorithm ADMM

Input: loss function $\ell \circ A$, regularization r, stepsize ρ,
initial z^0, $u^0 = 0$
for $k = 0, 1, \ldots$ **do**
    $x^{k+1} = \text{argmin}_x \{\ell(Ax) + \frac{\rho}{2}\|x - z^k + u^k\|_2^2\}$
    $z^{k+1} = \text{argmin}_z \{r(z) + \frac{\rho}{2}\|x^{k+1} - z + u^k\|_2^2\}$
    $u^{k+1} = u^k + x^{k+1} - z^{k+1}$
return $x_\star$ (nearly) minimizing $\ell(Ax) + r(x)$

problem: x-min involves the (large) data: not easy to solve!

Alternating Directions Method of Multipliers

Algorithm ADMM

Input: loss function $\ell \circ A$, regularization r, stepsize ρ,
initial z^0, $u^0 = 0$
for $k = 0, 1, \ldots$ **do**

$$x^{k+1} = \text{argmin}_x \{\ell(Ax) + \frac{\rho}{2}\|x - z^k + u^k\|_2^2\}$$
$$z^{k+1} = \text{argmin}_z \{r(z) + \frac{\rho}{2}\|x^{k+1} - z + u^k\|_2^2\}$$
$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

return $x_\star$ (nearly) minimizing $\ell(Ax) + r(x)$

problem: x-min involves the (large) data: not easy to solve!

solution: NysADMM [Zhao, Frangella, and Udell (2022)]

- *approximate* x-min with linear system
- solve (to moderate tolerance) with Nyström PCG

Quadratic approximation

if ℓ is twice diffable, approximate obj near prev iterate x^k

$$\ell(Ax) \approx \ell(Ax^k) + (x - x^k)^T A^T \nabla \ell(Ax^k) + \frac{1}{2}(x - x^k)^T A^T H_\ell(Ax^k) A(x - x^k)$$

where H_ℓ is the Hessian of ℓ.

Quadratic approximation

if ℓ is twice diffable, approximate obj near prev iterate x^k

$$\ell(Ax) \approx \ell(Ax^k) + (x - x^k)^T A^T \nabla \ell(Ax^k) + \frac{1}{2}(x - x^k)^T A^T H_\ell(Ax^k) A(x - x^k)$$

where H_ℓ is the Hessian of ℓ.

with this approximation, x-min becomes linear system: find x so

$$(A^T H_\ell(Ax^k)A + \rho I)x = r^k$$

where $r^k = \rho z^k - \rho u^k + A^T H_\ell(Ax^k)Ax^k - A^T \nabla \ell(Ax^k)$

NysADMM algorithm

Algorithm NysADMM

input loss function $\ell \circ A$, regularization r, stepsize ρ, positive summable sequence $\{\varepsilon^k\}_{k=0}^{\infty}$, initial z^0, $u^0 = 0$

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

  compute $r^k = \rho z^k - \rho u^k + A^T H_\ell(Ax^k)Ax^k - A^T \nabla \ell(Ax^k)$

  use Nyström PCG to find ε^k-apx solution x^{k+1} to

$$(A^T H_\ell(Ax^k)A + \rho I)x^{k+1} = r^k$$

  $z^{k+1} = \text{argmin}_z\{r(z) + \frac{\rho}{2}\|x^{k+1} - z + u^k\|_2^2\}$

  $u^{k+1} = u^k + x^{k+1} - z^{k+1}$

return $x_\star$ (nearly) minimizing $\ell(Ax) + r(x)$

The competition

lasso:

- SSNAL, a Newton augmented Lagrangian method [X. Li, Sun, and Toh (2018)]
- mfIPM, a matrix-free interior point method [Fountoulakis, Gondzio, and Zhlobich (2014)]
- glmnet, a coordinate-descent method [Friedman, Hastie, and Tibshirani (2010)]

logistic regression:

- SAGA, a stochastic average gradient method [Defazio, Bach, and Lacoste-Julien (2014)]

SVM:

- LIBSVM, a sequential minimal optimization (pairwise coordinate descent) method [Chang and Lin (2011)]

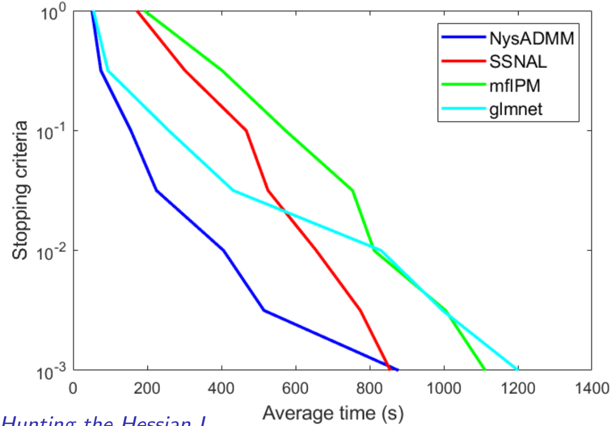
Numerical experiments: settings

- pick datasets with $n > 10,000$ or $d > 10,000$ from LIBSVM, UCI, and OpenML.
- use random feature map to generate more features
- use same stopping criterion and parameter settings as the standard solver for each problem class
- constant sketch size $s = 30$

Lasso results

stl10 dataset. stop iteration when

$$\frac{\|x - \mathrm{prox}_{\gamma\|\cdot\|_1}(x - A^T(Ax - b))\|}{1 + \|x\| + \|Ax - b\|} \leq \epsilon.$$



Lasso results

Task	Time for $\epsilon = 10^{-1}$ (s)			
	NysADMM	mfIPM	SSNAL	glmnet
STL-10	**165**	573	467	278
CIFAR-10-rf	**251**	655	692	391
smallNorb-rf	**219**	552	515	293
E2006.train	**313**	875	903	554
sector	**235**	678	608	396
realsim-rf	**193**	–	765	292
rcv1-rf	**226**	563	595	273
cod-rna-rf	**208**	976	865	324

ℓ_1-regularized logistic regression results

Table: Results for ℓ_1-regularized logistic regression experiment.

Task	NysADMM time (s)	SAGA (sklearn) time (s)
STL-10	**3012**	6083
CIFAR-10-rf	**7884**	21256
p53-rf	**528**	2116
connect-4-rf	**866**	4781
smallnorb-rf	**1808**	6381
rcv1-rf	**1237**	3988
con-rna-rf	**7528**	21513

Support vector machine results

NysADMM is $\geq 5\times$ faster, although code is pure python!

Table: Results of SVM experiment.

Task	NysADMM time (s)	LIBSVM time (s)
STL-10	**208**	11573
CIFAR-10	**1636**	8563
p53-rf	**291**	919
connect-4-rf	**7073**	42762
realsim-rf	**17045**	52397
rcv1-rf	**564**	32848
cod-rna-rf	**4942**	36791

Outline

Preconditioning

Nyström preconditioning

NysADMM

Optimization for deep learning

PINNs

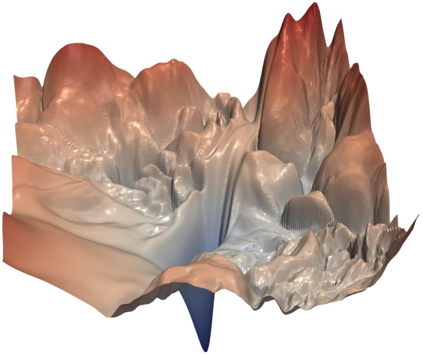
Optimization landscape

best methods for optimization depend on the landscape

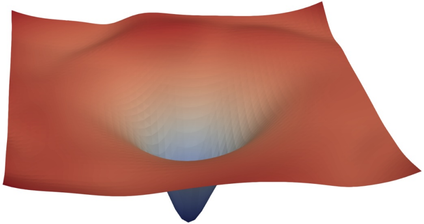
- local minima?
- saddle points?
- ill-conditioning?

what landscapes should we expect in modern problems (eg, 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.

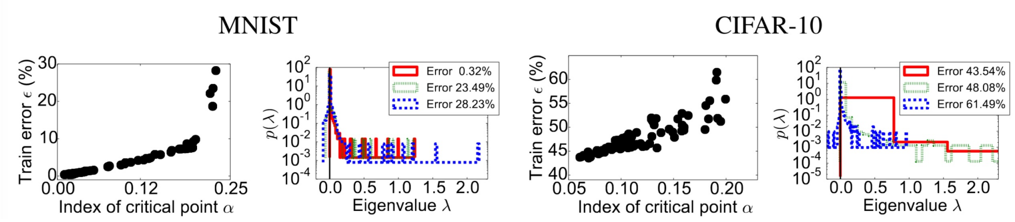
Source: H. Li, Xu, Taylor, et al., 2018

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! or just degenerate local minima?
  - "negative" eigenvalues are all nearly 0



Source: MLP experiments from Dauphin, Pascanu, Gulcehre, et al., 2014; for a modern take, see Sun, Li, Liang, et al., 2020.

Landscape-aware optimization

agenda:

1. **local minima.** ignore them: they are rarely a problem in modern architectures
    - or try random restarts / judicious initialization ...

Landscape-aware optimization

agenda:

1. **local minima.** ignore them: they are rarely a problem in modern architectures
   - or try random restarts / judicious initialization ...
2. **saddles.**
   - seek and follow directions of negative curvature? [Royer, O'Neill, and Wright (2020)]
   - nah, ignore them: associated eigenvalues are small [Alain, Roux, and Manzagol (2019) and Rathore, Lei, Frangella, et al. (2024)]

Landscape-aware optimization

agenda:

1. **local minima.** ignore them: they are rarely a problem in modern architectures
   - or try random restarts / judicious initialization ...
2. **saddles.**
   - seek and follow directions of negative curvature? [Royer, O'Neill, and Wright (2020)]
   - nah, ignore them: associated eigenvalues are small [Alain, Roux, and Manzagol (2019) and Rathore, Lei, Frangella, et al. (2024)]
3. **ill-conditioning.** precondition!

Landscape-aware optimization

agenda:

1. **local minima.** ignore them: they are rarely a problem in modern architectures
   - or try random restarts / judicious initialization ...
2. **saddles.**
   - seek and follow directions of negative curvature? [Royer, O'Neill, and Wright (2020)]
   - nah, ignore them: associated eigenvalues are small [Alain, Roux, and Manzagol (2019) and Rathore, Lei, Frangella, et al. (2024)]
3. **ill-conditioning.** precondition!

but how to query and use the $p \times p$ Hessian of $f : \mathbf{R}^p \to \mathbf{R}$?

Outline

Preconditioning

Nyström preconditioning

NysADMM

Optimization for deep learning

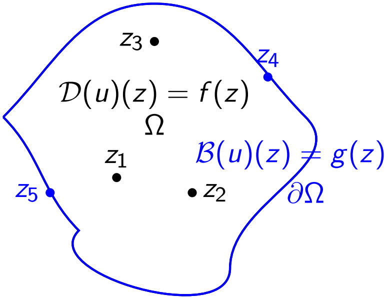
PINNs

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.

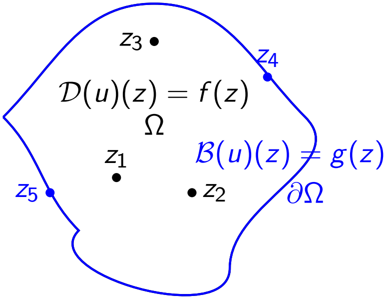


Physics-Informed Neural Networks (PINNs)

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

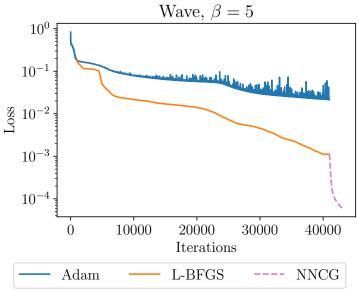


Figure: Even after running L-BFGS, the loss can be improved.

Source: [Rathore, Lei, Frangella, et al. (2024)]

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, $O(p)$)

Newton-CG: a matvec-only nearly-second-order optimizer

Newton-CG: repeat

- approximate f locally as a quadratic with $A = \nabla^2 f(x_0)$

$$f(x) \approx f(x_0) + \nabla f(x_0)^T (x - x_0) + \frac{1}{2}(x - x_0)^T A(x - x_0).$$

- (optionally) find a good preconditioner for A
- solve linear system $Ax = Ax_0 - \nabla f(x_0)$ with PCG.

algorithm only uses gradient evaluations and matrix-vector products with A
$\implies$ compatible with AD

Source: [Rathore, Lei, Frangella, et al. (2024)]

Preconditioners can improve conditioning

plot spectral density of PINN Hessian for different PDEs

- blue: original function
- orange: after preconditioning

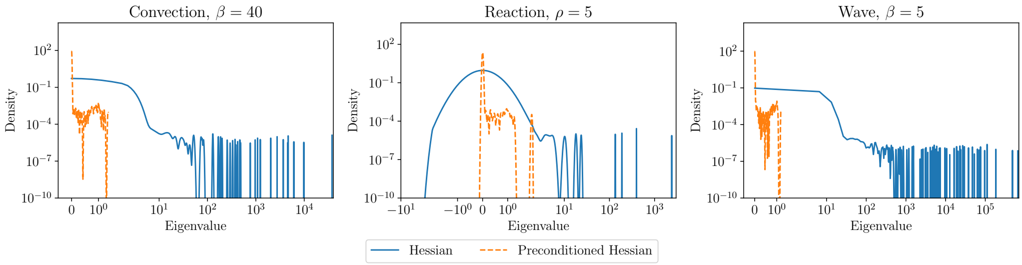
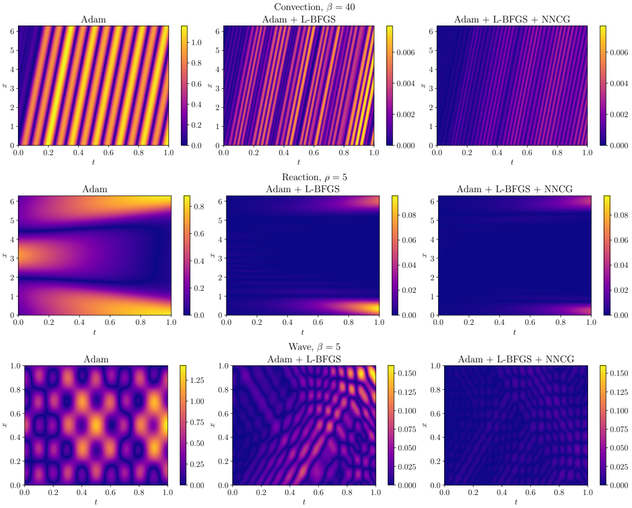


Figure: The total loss is ill-conditioned for all three PDEs. [Rathore, Lei, Frangella, et al. (2024)]

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

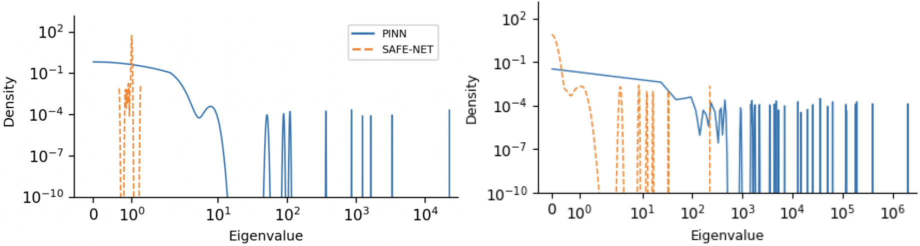
Preconditioned optimizers improve fits



Architectural choices can improve conditioning

plot spectral density of PINN Hessian for wave PDEs

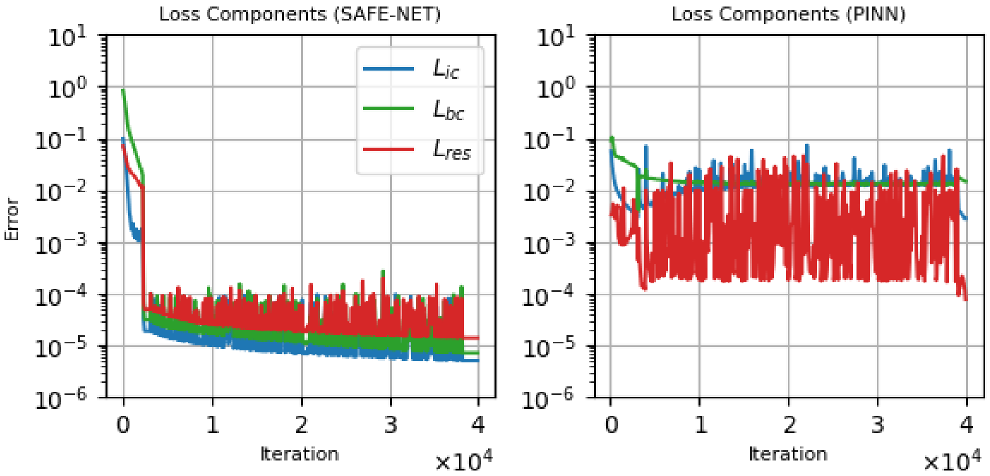
- blue: standard MLP architecture
- orange: with SAFE-NET architecture (single layer with fourier features)



(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

rlaopt: Randomized Linear Algebra for Scalable Optimization

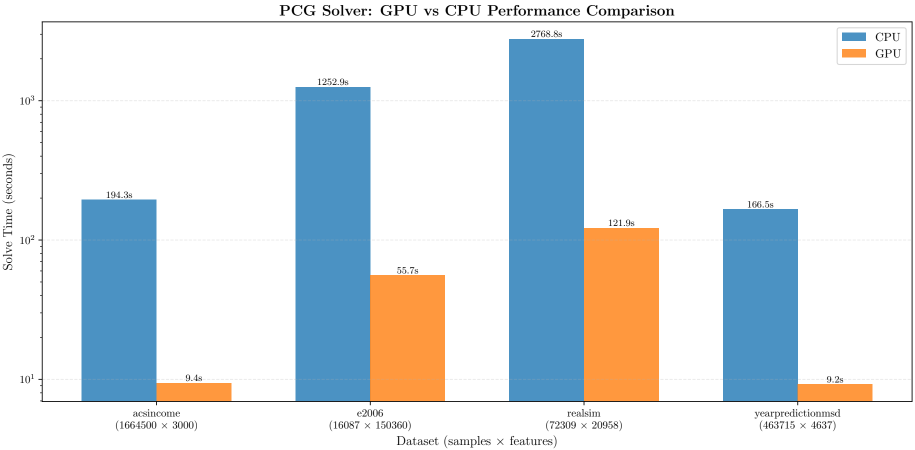
problems:

- solve linear systems $Ax = b$ faster
- composite optimization: minimize $f(Ax) + g(x)$
- stochastic optimization: minimize $\sum_{i=1}^{n} f_i(x)$

algorithms:

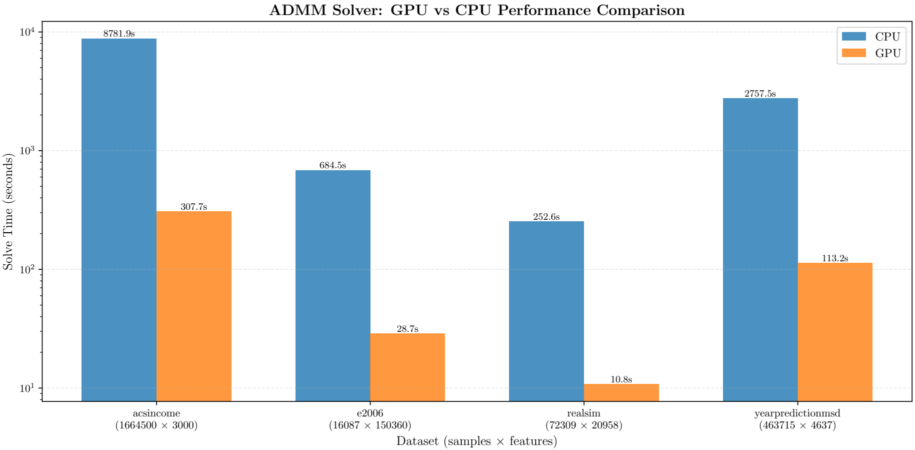
- Nyström PCG for solving linear systems
- Nyström ADMM for composite optimization
- PROMISE: low-rank stochastic optimization
- SAPPHIRE: stochastic proximal gradient method

rlaopt delivers 20× speedups solving dense linear system



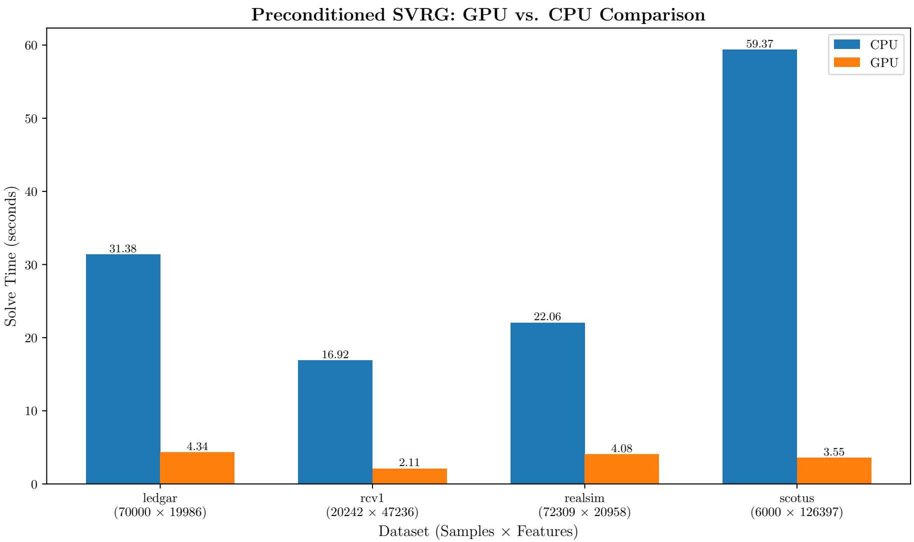
ridge regression solved with PCG + Nyström preconditioner

rlaopt delivers 25× speedups solving elastic net



elastic net with a box constraint solved with NysADMM

rlaopt delivers 8× speedups solving logistic regression



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
  - randomized Nyström approximation
  - autodiff from Hessian-vector products

Conclusion

does your optimization suffer from ill-conditioning?

preconditioners can help!

- spectral preconditioning is feasible at large scale
  - randomized Nyström approximation
  - autodiff from Hessian-vector products
- randomized preconditioners can speed up
  - composite optimization (e.g.NysADMM)
  - deep learning (e.g.NysNewton-CG)
  - stochastic optimization (e.g.SketchySGD, PROMISE, SAPPHIRE)

Conclusion

does your optimization suffer from ill-conditioning?

preconditioners can help!

- spectral preconditioning is feasible at large scale
  - randomized Nyström approximation
  - autodiff from Hessian-vector products
- randomized preconditioners can speed up
  - composite optimization (e.g.NysADMM)
  - deep learning (e.g.NysNewton-CG)
  - stochastic optimization (e.g.SketchySGD, PROMISE, SAPPHIRE)
- (tomorrow) 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
- NysADMM for composite optimization minimize $\ell(x) + r(x)$:
  - algorithm (NysADMM): https://arxiv.org/abs/2202.11599
  - convergence (GeNI-ADMM): https://arxiv.org/abs/2302.03863
  - solver (GeNIOS): https://github.com/tjdiamandis/GeNIOS.jl
- 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
- PyTorch implementation of all these methods: rlaopt https:www.github.com/udellgroup/rlaopt