CME 307 / MS&E 311: Optimization

Low rank approximation for faster optimization

Professor Udell

Management Science and Engineering
Stanford

May 17, 2023

Low rank approximation for faster optimization

thesis: randNLA allows $O(n)$ matvecs with n$\times$n matrix A
$\implies$ can speed up algorithms that use large matrices,

Low rank approximation for faster optimization

thesis: randNLA allows $O(n)$ matvecs with n×n matrix A
$\implies$ can speed up algorithms that use large matrices, e.g.,

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

Low rank approximation for faster optimization

thesis: randNLA allows $O(n)$ matvecs with n$\times$n matrix A
$\implies$ can speed up algorithms that use large matrices, e.g.,

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

Low rank approximation for faster optimization

thesis: randNLA allows $O(n)$ matvecs with n×n matrix A
$\implies$ can speed up algorithms that use large matrices, 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

thesis: randNLA allows $O(n)$ matvecs with n×n matrix A
$\implies$ can speed up algorithms that use large matrices, 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. SketchySGD for finite sum minimization $\sum_{i=1}^{n} f_i(x)$
   low rank approximation for Newton system improves
   - robustness (vs first-order methods) and
   - speed (vs other quasi-Newton methods)

Low rank approximation for faster optimization

thesis: randNLA allows $O(n)$ matvecs with n×n matrix A
$\implies$ can speed up algorithms that use large matrices, 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. SketchySGD for finite sum minimization $\sum_{i=1}^{n} f_i(x)$
   low rank approximation for Newton system improves
   - robustness (vs first-order methods) and
   - speed (vs other quasi-Newton methods)

   even works for deep learning!

Outline

Low rank approximation

Nyström PCG

SketchySGD

ADMM

NysADMM

Low rank approximation via eigenvalues

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

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

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

with $\Lambda = \mathbf{diag}(\lambda_1, \ldots, \lambda_n)$, $\lambda_1 \geq \cdots \geq \lambda_n$,
$UU^T = U^T U = I_n$,

- truncate to top s eigenvector/value pairs:

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

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

Nyström approximation

given $A \in \mathbf{S}_+^n$, approximate with the *Nyström method*:

- choose any test matrix $\Omega \in \mathbb{R}^{n \times s}$, $1 \le s \le n$
- *Nyström approximation* of A wrt Ω is [Tropp et al. (2017)]

$$A\langle\Omega\rangle = (A\Omega)(\Omega^T A\Omega)^\dagger (A\Omega)^T.$$

properties:

- $A\langle\Omega\rangle \in \mathbf{S}_+^n$
- $\mathbf{rank}(A\langle\Omega\rangle) \le s$
- $A\langle\Omega\rangle \preceq A$

Efficient eigs via randomized NLA

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

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

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

- 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 $\Omega \in \mathbb{R}^{n \times s}$
- compute randomized linear sketch $Y = A\Omega$.
- form *Nyström approximation*

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

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

properties:

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

Randomized Nyström approximation: guarantees

define the *p-stable rank* $\mathrm{sr}_p(A) = \lambda_p^{-1} \sum_{j=p}^{n} \lambda_j$

Randomized Nyström approximation: guarantees

define the *p-stable rank* $\mathrm{sr}_p(A) = \lambda_p^{-1} \sum_{j=p}^{n} \lambda_j$

Theorem (Randomized Nyström approximation)

Let $A \in \mathbf{S}_+^n$ *with eigenvalues* $\lambda_1 \geq \cdots \geq \lambda_n$. *Pick any* $p \geq 2$ *and set sketch size* $s = 2p - 1$. *Draw a Gaussian random test matrix* $\Omega \in \mathbb{R}^{n \times s}$. *Then* $\hat{A}_{\mathrm{nys}}$ *satisfies*

$$\mathbb{E}\|A - \hat{A}_{\mathrm{nys}}\| \leq \left(3 + \frac{4e^2}{p}\mathrm{sr}_p(A)\right)\lambda_p.$$

- error of randomized rank-s approximation is comparable with best error of any rank-$p = \frac{s+1}{2}$ approximation

Outline

Low rank approximation

Nyström PCG

SketchySGD

ADMM

NysADMM

Regularized linear system

find $x \in \mathbf{R}^n$ such that

$$(A + \mu I)x = b$$

where $A \in \mathbf{S}_+^n$ is symmetric psd and $\mu \geq 0$.

- eigenvalues of A $\lambda_1 \geq \cdots \geq \lambda_n$
- condition number $\kappa(A) = \lambda_1(A)/\lambda_n(A)$
- regularized matrix $A_\mu = A + \mu I$ has $\kappa(A_\mu) \leq \kappa(A)$
- matvec(A) time to compute matrix vector product Ax (often = nnz(A))

Sketch-and-solve

Given a rank-s (Nyström) approximation $A \approx \hat{A} = V\hat{\Lambda}V^T$, why not solve

$$(\hat{A} + \mu I)\hat{x} = b \qquad \text{instead of} \qquad (A + \mu I)x^\star = b?$$

- $(+)$ can apply inverse in $O(ns)$ time, since

$$(\hat{A} + \mu I)^{-1} = V(\hat{\Lambda} + \mu I)^{-1}V^T + \frac{1}{\mu}(I - VV^T)$$

Sketch-and-solve

Given a rank-s (Nyström) approximation $A \approx \hat{A} = V\hat{\Lambda}V^T$, why not solve

$$(\hat{A} + \mu I)\hat{x} = b \qquad \text{instead of} \qquad (A + \mu I)x^\star = b?$$

- $(+)$ can apply inverse in $O(ns)$ time, since

$$(\hat{A} + \mu I)^{-1} = V(\hat{\Lambda} + \mu I)^{-1}V^T + \frac{1}{\mu}(I - VV^T)$$

- $(+)$ works well if $b \in \textbf{span}(V)$

Sketch-and-solve

Given a rank-s (Nyström) approximation $A \approx \hat{A} = V\hat{\Lambda}V^T$, why not solve

$$(\hat{A} + \mu I)\hat{x} = b \qquad \text{instead of} \qquad (A + \mu I)x^\star = b?$$

- $(+)$ can apply inverse in $O(ns)$ time, since

$$(\hat{A} + \mu I)^{-1} = V(\hat{\Lambda} + \mu I)^{-1}V^T + \frac{1}{\mu}(I - VV^T)$$

- $(+)$ works well if $b \in \mathbf{span}(V)$
- (-) high accuracy requires $s \to n$

Preconditioning CG

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 CG

for any $P \succ 0$,

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

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

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

Preconditioning CG

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

how to precondition?

- common heuristic: Jacobi preconditioning $P = \mathbf{diag}(A)$
- incomplete Cholesky (best for structured sparsity)

Sketch-and-precondition

Sketch-and-precondition [Avron, Maymounkov, and Toledo (2010), Martinsson and Tropp (2020), X. Meng, Saunders, and Mahoney (2014), and Rokhlin and Tygert (2008)]: for an overdetermined problem $A = X^T X$ where $X \in \mathbf{R}^{N \times n}$, $N \gg n$,

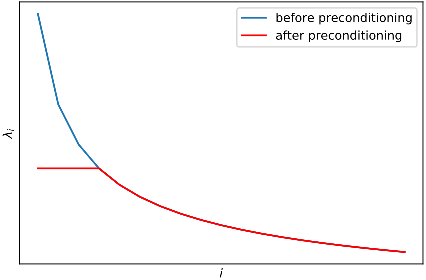
- pick *sketch size* $s = \Omega(n)$
- draw random matrix $S \in \mathbb{R}^{s \times n}$ (eg, iid normal entries)
- compute randomized *sketch* SX
- compute pivoted-QR factorization $SX = QR$
- precondition with $P = R^{-1}$

$O(n^3)$ flops, so only useful for moderate n

An optimal low-rank preconditioner

- suppose $\lfloor A \rfloor_s = V_s \Lambda_s V_s^T$ is a best rank-s apx to $A \in \mathbf{S}_+^n$.
- the best preconditioner 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

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

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

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

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

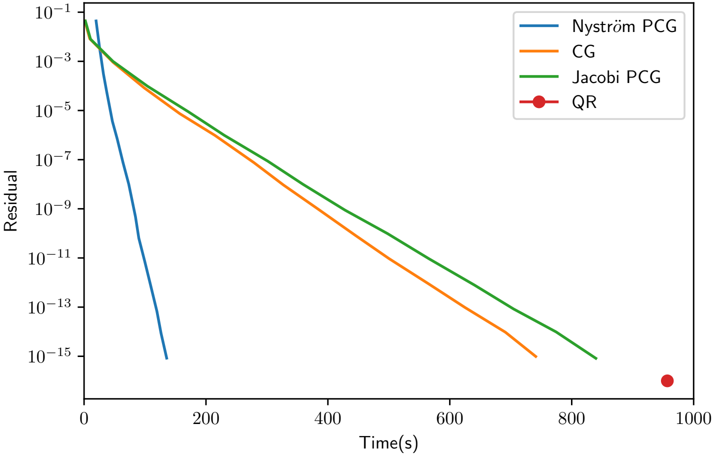
$$P_{\mathrm{nys}} = \frac{1}{\hat{\lambda}_s + \mu} V(\hat{\Lambda} + \mu I)V^T + (I - VV^T)$$

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

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

Source: Frangella, Tropp, and Udell, 2023

Nyström preconditioner is fast!



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

Nyström PCG controls the condition number

Theorem (Nyström condition number bound)

Let P be the Nyström preconditioner with regularization parameter $\mu \geq 0$ and let $M = P^{-1/2} A_\mu P^{-1/2}$ be the preconditioned matrix. Define the error $E = A - \hat{A}_{\mathrm{nys}}$. Then

$$\kappa(M) \leq \min \left\{ \frac{\hat{\lambda}_s + \mu + \|E\|}{\mu},\ 1 + \frac{\|E\|}{\hat{\lambda}_s + \mu} + \frac{\hat{\lambda}_s + \mu + \|E\|}{\lambda_n + \mu} \right\}.$$

Nyström PCG controls the condition number

Theorem (Nyström condition number bound)

Let P be the Nyström preconditioner with regularization parameter $\mu \geq 0$ and let $M = P^{-1/2} A_\mu P^{-1/2}$ be the preconditioned matrix. Define the error $E = A - \hat{A}_{\text{nys}}$. Then

$$\kappa(M) \leq \min\left\{ \frac{\hat{\lambda}_s + \mu + \|E\|}{\mu},\ 1 + \frac{\|E\|}{\hat{\lambda}_s + \mu} + \frac{\hat{\lambda}_s + \mu + \|E\|}{\lambda_n + \mu} \right\}.$$

corollary: for large enough s, $\hat{\lambda}_s \leq \mu$ and $\|E\| \leq \mu$, so

$$\kappa_2(P^{-1/2} A_\mu P^{-1/2}) \leq 3.$$

How to choose sketch size?

how to get $\|E\| \sim \mu$?

- fixed sketch size $s = 50$ (works surprisingly well!)
- adaptive: increase sketch size until (estimated) error is small enough
  - $\|E\| \approx \hat{\lambda}_\ell$
  - add one dimension to sketch for a-posteriori error guarantee [Tropp et al. (2019)]
- a priori, bound sketch size needed to ensure $\|E\| \sim \mu$

A priori bound via the effective dimension

the *effective dimension* at μ is a smoothed count of evs $\geq \mu$:

$$d_{\mathsf{eff}}(\mu) = \sum_{j=1}^{n} \frac{\lambda_j}{\lambda_j + \mu}.$$

A priori bound via the effective dimension

the *effective dimension* at μ is a smoothed count of evs $\geq \mu$:

$$d_{\text{eff}}(\mu) = \sum_{j=1}^{n} \frac{\lambda_j}{\lambda_j + \mu}.$$

the effective dimension bounds sketch size required for constant condition number

Theorem

Construct the randomized Nyström preconditioner P with rank $s = 2\lceil 1.5 d_{\text{eff}}(\mu) \rceil + 1$. Then

$$\mathbb{E}\left[\kappa(P^{-1/2} A_\mu P^{-1/2})\right] < 28.$$

So whp relative error is $< \epsilon$ after $T \leq \lceil 2.7 \log(\frac{2}{\epsilon}) \rceil$ iterations.

PCG converges fast when $s \sim d_{\text{eff}}$

plug in bound on condition number to CG convergence theory:

Corollary

Let $M = P^{-1/2} A_\mu P^{-1/2}$, and suppose

$$\kappa(M) < 28.$$

Then relative error $\delta_t := \|x_t - x_\star\|_M / \|x_\star\|_M$ of PCG iterate x_t, initialized with $x_0 = 0$, satisfies

$$\delta_t < 2\,(0.69)^t$$

and PCG attains relative error $\delta_t < \epsilon$ after $T \leq \lceil 2.7 \log(\frac{2}{\epsilon}) \rceil$ iterations.

Experimental results

Dataset	Method	# iterations	Runtime (s)
Higgs-rf	AdaIHS	55	$1,052.7$
	R&T	53	607.4
	Adaptive Nyström	**28**	**91.26**
YearMSD-rf	AdaIHS	44	$1,327.3$
	R&T	49	766.5
	Adaptive Nyström	**22**	**209.7**
EMNIST	Random features PCG	154	635.2
	Nyström	**32**	**268.4**
Santander	Random features PCG	160	810.4
	Nyström	**31**	**164.8**

Table: Nyström PCG is faster than other randomized preconditioners.

- For Higgs and YearMSD, s uses a posteriori error estimation.
- For EMNIST and Santander, $s = 1,000$
- R&T: sketch-and-precondition method [Rokhlin and Tygert (2008)]
- AdaIHS: Adaptive iterative Hessian sketch [Lacotte and Pilanci (2020)]
- Random features PCG [Avron, Clarkson, and Woodruff (2017)] uses $s = 1000$

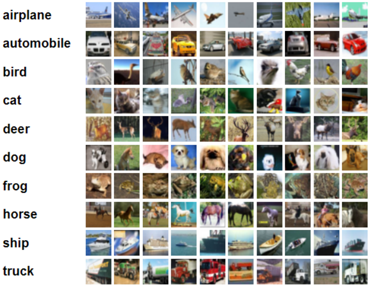
Numerics: details

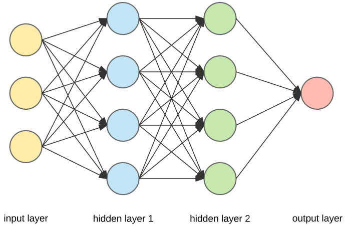
Dataset	n	d	# classes	μ	σ	PCG tolerance
Higgs-rf	800,000	10,000	2	1e-4	5	1e-10
YearMSD-rf	463,715	15,000	NA	1e-5	8	1e-10
EMNIST	105,280	784	47	1e-6	8	1e-3
Santander	160,000	200	2	1e-6	7	1e-3

Table: Datasets: statistics and parameters.

Outline

Classification with neural network

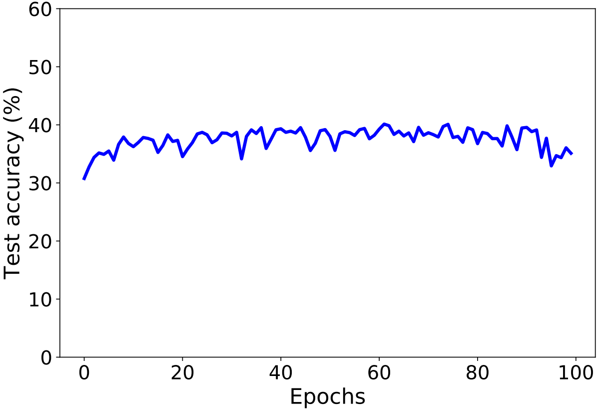




- CIFAR-10 dataset, tabular version
- basic MLP network
- use Adam to train the neural network

Adam needs babysitting

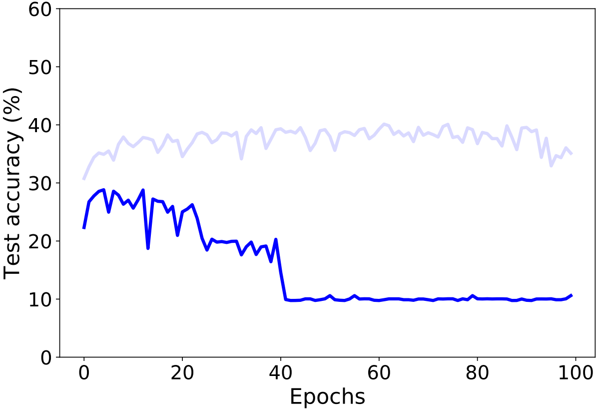
Adam is sensitive to hyperparameter settings



learning rate 0.005

Adam needs babysitting

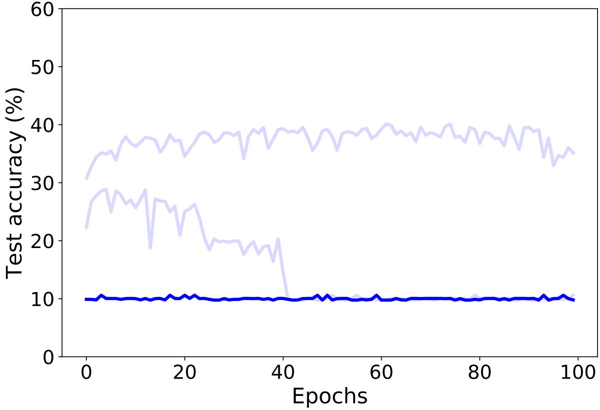
Adam is sensitive to hyperparameter settings



learning rate 0.01

Adam needs babysitting

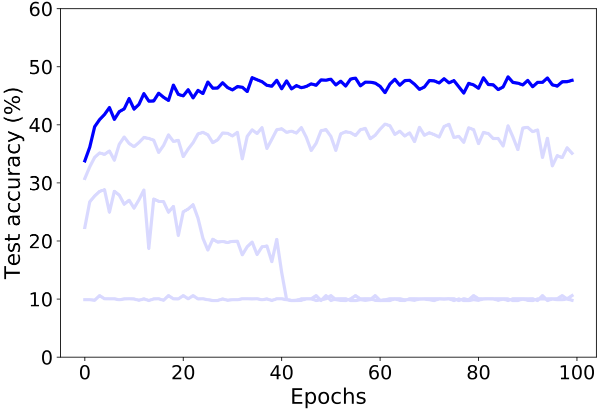
Adam is sensitive to hyperparameter settings



learning rate 0.02

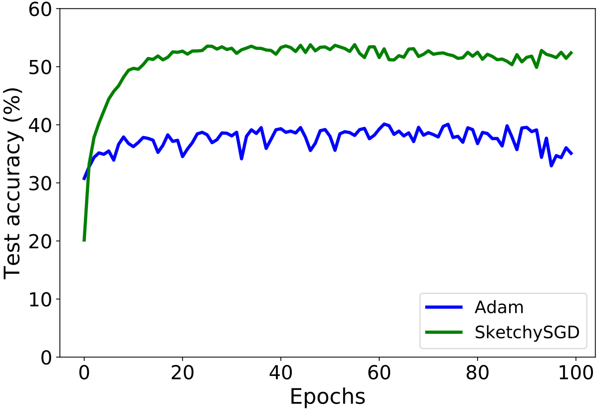
Adam needs babysitting

Adam is sensitive to hyperparameter settings



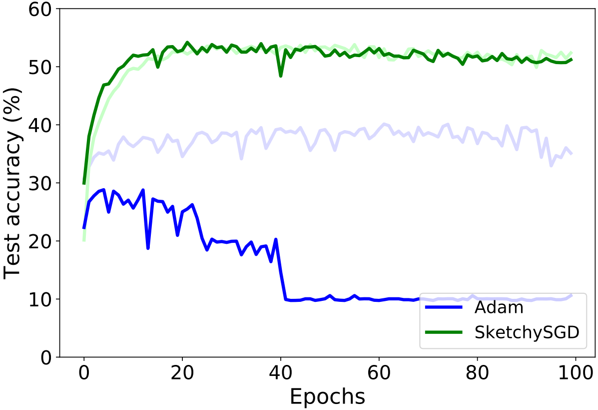
learning rate 0.001

Preconditioning improves robustness



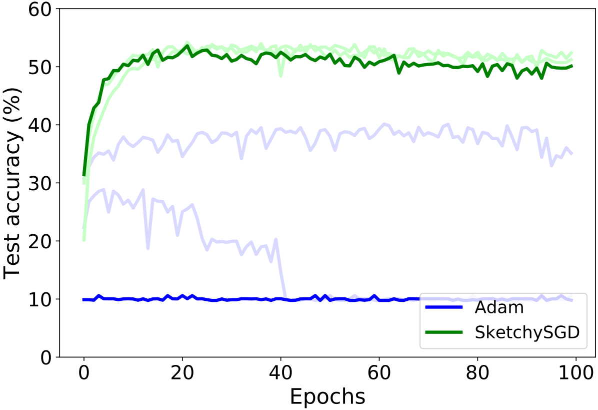
learning rate 0.005

Preconditioning improves robustness



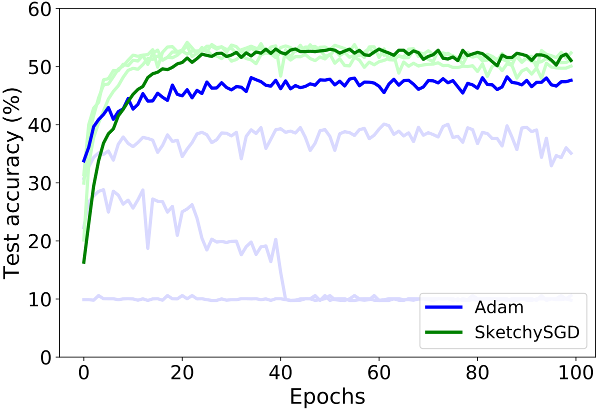
learning rate 0.01

Preconditioning improves robustness



learning rate 0.02

Preconditioning improves robustness



learning rate 0.001

Bad tuning $\implies$ slow convergence

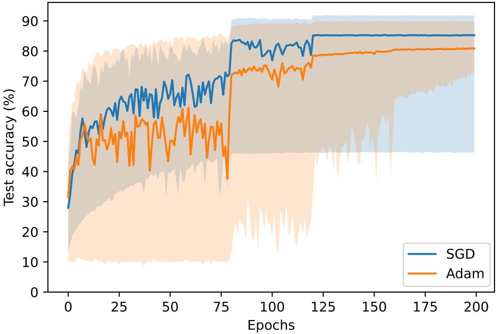
how does initial learning rate affect performance?

- ResNet-20 architecture
- CIFAR-10 dataset
  ($m_{\mathrm{tr}} = 50,000$, $m_{\mathrm{tst}} = 10,000$, $n = 3,072$)
- SGD and Adam optimizers
- initialize learning rate η at

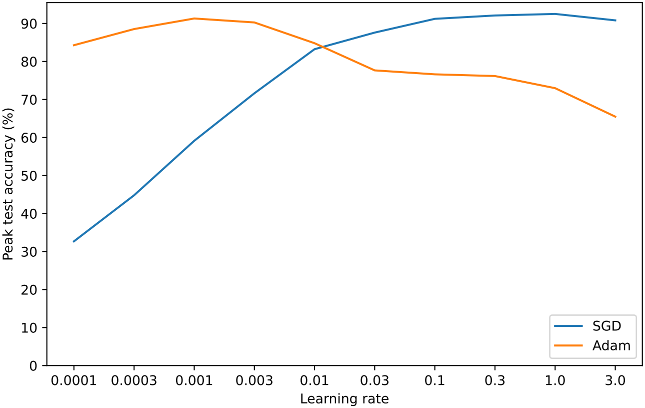
  $$\{10^{-4}, 3{\cdot}10^{-4}, 10^{-3}, 3{\cdot}10^{-3}, 10^{-2}, 3{\cdot}10^{-2}, 10^{-1}, 3{\cdot}10^{-1}, 10^{0}, 3{\cdot}10^{0}\}$$

- follow best practices to decay η throughout training

Bad tuning $\implies$ slow convergence



Bad tuning $\implies$ slow convergence

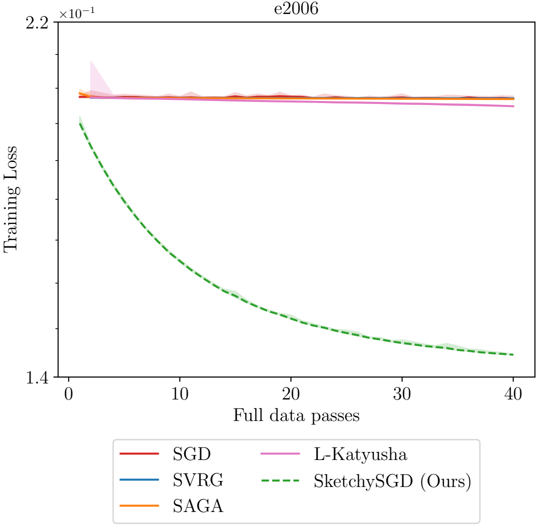


Ill-conditioning $\implies$ slow convergence

experiment on ill-conditioned dataset

- ridge regression on E2006-dataset ($m = 16,087, p = 150,360$)
- (small) l_2-regularization $\nu = \frac{10^{-2}}{m}$
- state of the art first order methods for this problem: SGD, SVRG, SAGA, L-Katyusha, tuned for best performance
- SketchySGD with default parameters

Ill-conditioning $\Longrightarrow$ slow convergence



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

Preconditioned stochastic optimization

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

Preconditioned stochastic optimization

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

Q: Why not use Quasi-Newton methods like (L-)BFGS?

Preconditioned stochastic optimization

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

Q: Why not use Quasi-Newton methods like (L-)BFGS?
A: Classical QN requires full gradient evaluations

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

- from a data subsample
- from stale data
- by the secant condition (BFGS, l-BFGS)
- by diagonal approximation (adaHessian)
- by block-diagonal kronecker approximation (Shampoo, KFAC, SENG, K-BFGS)
- by low rank approximation (sketchySGD)

Source: Erdogdu and Montanari, 2015, Shampoo Gupta, Koren, and Singer, 2018, Roosta-Khorasani and Mahoney, 2019, Bollapragada, Byrd, and Nocedal, 2019, AdaHessian Yao et al., 2021, R-SSN S. Y. Meng et al., 2020, KFAC Grosse and Martens, 2016, SENG Yang et al., 2020, Goldfarb, Ren, and Bahamou, 2020

Subsampling the Hessian

Hessian of $f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$ is

$$\nabla^2 f(w) = \frac{1}{m}\sum_{i=1}^{m} \nabla^2 f_i(w)$$

Subsampled Hessian is

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

where $S \subseteq \{1, \cdots, m\}$ is chosen uniformly at random.

Subsampling the Hessian

Hessian of $f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$ is

$$\nabla^2 f(w) = \frac{1}{m}\sum_{i=1}^{m} \nabla^2 f_i(w)$$

Subsampled Hessian is

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

where $S \subseteq \{1, \cdots, m\}$ is chosen uniformly at random.

Subsampled Newton method:

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

More approximations, more problems

1. *complexity.* Hessian of single loss $f_i : \mathbf{R}^p \to \mathbf{R}$ costs p^2 to compute and to store
2. *invertibility.* Hessian approximation may not be invertible $\widehat{\nabla}^2 f(w_k)$
3. *descent.* (stochastic quasi-)Newton search direction

$$\left(\widehat{\nabla}^2 f(x_k)\right)^{-1} \widehat{\nabla} f(w_k)$$

may not be a descent direction

More approximations, more problems

1. *complexity.* Hessian of single loss $f_i : \mathbf{R}^p \to \mathbf{R}$ costs p^2 to compute and to store
2. *invertibility.* Hessian approximation may not be invertible $\widehat{\nabla}^2 f(w_k)$
3. *descent.* (stochastic quasi-)Newton search direction

$$\left(\widehat{\nabla}^2 f(x_k)\right)^{-1} \widehat{\nabla} f(w_k)$$

   may not be a descent direction

solutions:

1. *complexity.* automatic differentiation
2. *invertibility.* use regularized Hessian approx $\widehat{\nabla}^2 + \rho I$
3. *descent.* harder ...

Complexity: how to access $\nabla^2 f(w)$?

our oracle: stochastic Hessian-vector products (HVPs)

1. minibatch loss $\tilde{f}(w) = \sum_{i \in S} f_i(w)$ for $S \subset \{1, \ldots, n\}$
2. compute minibatch gradient with automatic differentiation (AD) $\tilde{g}(w) = \nabla \tilde{f}(w)$
3. define minibatch Hessian vector product with vector v

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

   and compute using AD on $\tilde{g}(w) \cdot v$ (Pearlmutter's trick)

cost: two passes of AD $\approx 4\times$ cost of function evaluation

HVPs to find (stochastic quasi)-Newton direction

- CG (or MINRES, for indefinite $\widehat{\nabla}^2 f(x)$) to compute search direction
$$\left(\widehat{\nabla}^2 f(x_k)\right)^{-1} \widehat{\nabla} f(w_k)$$
uses only HVPs
  - problem: bad conditioning $\implies$ slow convergence of CG
- Nystrom approximation for regularized Hessian $\widehat{\nabla}^2 + \rho I$ uses only HVPs

SketchySGD

every now and then (e.g., each epoch),

- sample data batch S_k to sketch subsampled Hessian
$$H_{S_k}(w_k) = \frac{1}{|S_k|} \sum_{j \in S_k} \nabla^2 f_j(w_k)$$
- form rank r approximation $\hat{H}_{S_k}$

at each iteration k,

- sample data batch B_k
- form gradient estimate
$$g_{B_k}(w_k) = \frac{1}{|B_k|} \sum_{j \in B_k} g_j(w_k)$$
- take step
$$w_{k+1} = w_k - \eta_k (\hat{H}_{S_k} + \rho_k I)^{-1} g_{B_k}(w_k)$$

SketchySGD is fast

computing search direction v_k requires $O(pr)$ flops:

$$v_k = \hat{V}\left(\hat{\Lambda} + \rho_k I\right)^{-1} \hat{V}^T g_{B_k} + \frac{1}{\rho_k}(g_{B_k} - \hat{V}\hat{V}^T g_{B_k})$$

- the cost of a fresh low-rank Hessian approximation is $O((b_{h_k} + r^2)p)$.
- given Hessian approximation, per-iteration cost is $O\left((b_{g_k} + r)p\right)$.

Relative condition number

the *relative condition number* is $\hat{\kappa} = \hat{L}/\hat{\mu}$ where $\hat{L} \geq \hat{\mu} > 0$ are defined such that for all $w, w' \in \mathcal{X}$

$$f(w') \leq f(w) + \langle g(w), w' - w \rangle + \frac{\hat{L}}{2} \|w' - w\|_{H(w)}^2,$$
$$f(w') \geq f(w) + \langle g(w), w' - w \rangle + \frac{\hat{\mu}}{2} \|w' - w\|_{H(w)}^2.$$

(The condition number $\kappa = L/\mu$ is defined similarly, replacing $H(w)$ by I.)

Theory: convex

Suppose f_i are all smooth and convex and f is L-smooth and μ-strongly convex. Define

$$\lambda_{r+1}^{\star} = \sup_{w \in \mathcal{X}} \lambda_{r+1}(H(w)).$$

Observe $\lambda_{r+1}^{\star} \leq L$ and is often significantly smaller.

Corollary

Let $T_{\text{SketchySGD}}$ denote the iteration complexity of SketchySGD and T_{SGD} denote the iteration complexity of SGD given from Theorem 4.6 in Gower, Sebbouh, and Loizou, 2021. Then

$$\frac{T_{\text{SGD}}}{T_{\text{SketchySGD}}} \geq \frac{\hat{\mu}}{\hat{\kappa}} \frac{L}{30\lambda_{r+1}^{\star}}.$$

In particular, in the case of the least-squares loss we have

$$\frac{T_{\text{SGD}}}{T_{\text{SketchySGD}}} \geq \frac{L}{30\lambda_{r+1}^{\star}} = \frac{\lambda_1(H)}{30\lambda_{r+1}(H)}.$$

Theory: nonconvex

assumptions

- f and each f_i are twice differentiable and smooth
- f satisfies PL condition:

$$\|g(w)\|^2 \geq 2\theta(f(w) - f(w_\star)), \ \forall w$$

- interpolation: optimizer $w_\star \in \mathcal{W}_\star$ satisfies $\|g_i(w_\star)\| = 0$ for each $i \in \{1, \ldots n\}$

the SketchySGD iterate w_t after $t > 0$ iterations satisfies

$$\mathbb{E}[f(w_t)] - f(w_\star) \leq (1 - h(\theta))^t \, (f(w_0) - f(w_\star)).$$

- constant $h(\theta)$ has explicit analytical form
- linear convergence (optimality gap drops exponentially)

SketchySGD: simple parameter selection

For convex problems:

- Batch sizes set equal $b_g = b_h$. We used 256 in these examples.
- Fixed rank $r = 50$.
- Fixed regularization $\rho \in \{10^{-1}, 10^{-2}, 10^{-3}\}$ at every iteration. Here, $\rho = 10^{-3}$.
- Learning rate $\eta = \frac{1}{1+100\hat{\lambda}_r}$, where $\hat{\lambda}_r$ is the rth eigenvalue of the current subsampled Hessian approximation $\hat{H}$.
- Compute a fresh approximation $\hat{H}$ after each epoch or two.

For deep learning:

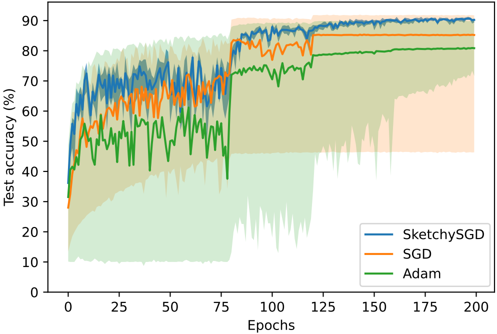
- Batch sizes set equal $b_g = b_h$. We used 128 in this paper.
- Fixed rank $r = 100$.
- Fixed learning rate $\eta = 10^{-2}$.
- Fixed regularization $\rho = \eta$ at every iteration.
- Compute a fresh approximation $\hat{H}$ after each epoch or two.

SOTAish results in deep learning

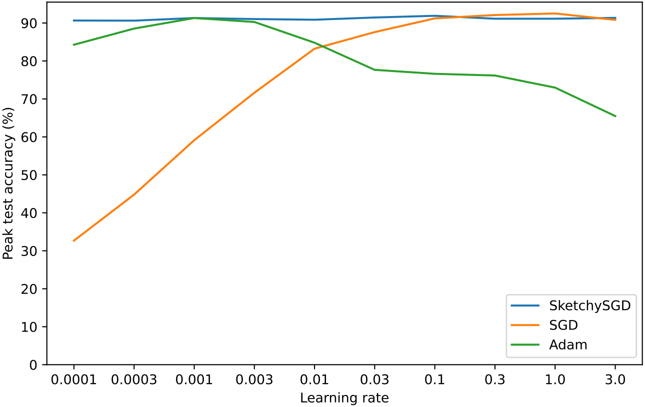
- SketchySGD uses the default parameter choices
- preconditioner is updated every 2 epochs
- all optimizers use the same learning rate decay

SketchySGD is reliable (CIFAR-10)

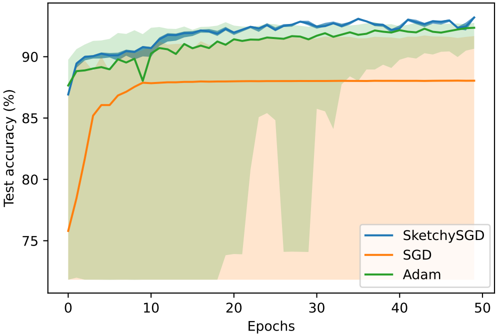
back to ResNet-20 on CIFAR-10



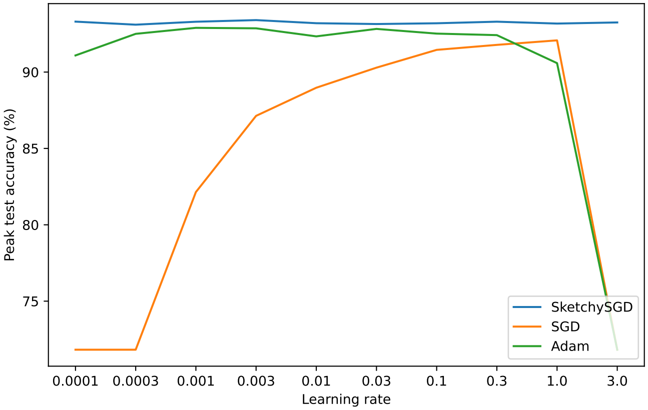
SketchySGD is near-optimal (CIFAR-10)



SketchySGD is reliable (Miniboone)

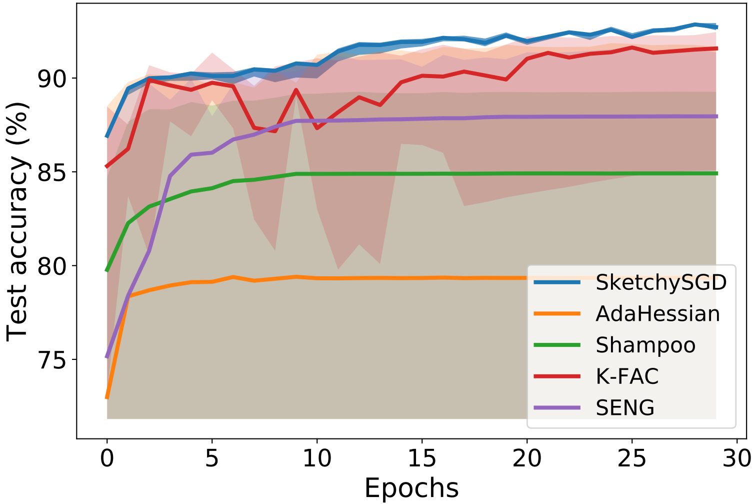


SketchySGD is near-optimal (Miniboone)

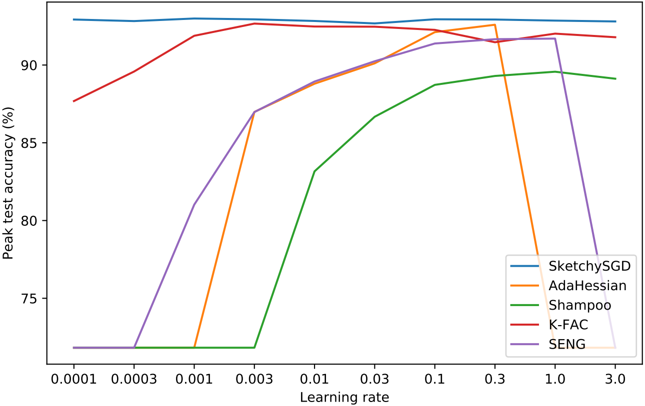


SketchySGD is more reliable than SQN competitors

stochastic quasi-Newton methods for DL on MiniBoone



SketchySGD outperforms SQN competitors



Outline

Composite optimization

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

- $A : \mathbf{R}^n \to \mathbf{R}^m$ linear
- $\ell : \mathbf{R}^m \to \mathbf{R}$ smooth
- $r : \mathbf{R}^n \to \mathbf{R}$ proxable
  - easy (often closed form) solution to $\mathbf{prox}_r(x) = \mathrm{argmin}_y \, r(y) + \frac{1}{2}\|x - y\|^2$
  - e.g., for $r(x) = \|x\|_1$, $\mathbf{prox}_r(x)$ is soft-thresholding operator

Example: Lasso

$$\text{minimize} \quad \frac{1}{2}\|Ax - b\|_2^2 + \gamma\|x\|_1$$

- $\ell(Ax) = \frac{1}{2}\|Ax - b\|_2^2$ smooth
- $r(x) = \gamma\|x\|_1$ proxable
- parameter $\gamma > 0$ controls strength of regularization

Example: Lasso

$$\text{minimize} \quad \frac{1}{2}\|Ax - b\|_2^2 + \gamma\|x\|_1$$

- $\ell(Ax) = \frac{1}{2}\|Ax - b\|_2^2$ smooth
- $r(x) = \gamma\|x\|_1$ proxable
- parameter $\gamma > 0$ controls strength of regularization

other examples: regularized logistic regression, SVM, ...

ADMM

consider the problem

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Ax + Bz = c \end{array}$$

Augmented Lagrangian for this problem (with dual variable y) is

$$L_t(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + t/2\|Ax + Bz - c\|^2$$

Alternating Directions Method of Multipliers (ADMM) iteration is

$$\begin{aligned} x^{(k+1)} &= \operatorname*{argmin}_x L_t(x, z^{(k)}, y^{(k)}) \\ z^{(k+1)} &= \operatorname*{argmin}_z L_t(x^{(k+1)}, z, y^{(k)}) \\ y^{(k+1)} &= y^{(k)} + \frac{1}{t}(Ax^{(k+1)} + Bz^{(k+1)} - c) \end{aligned}$$

ADMM

properties:

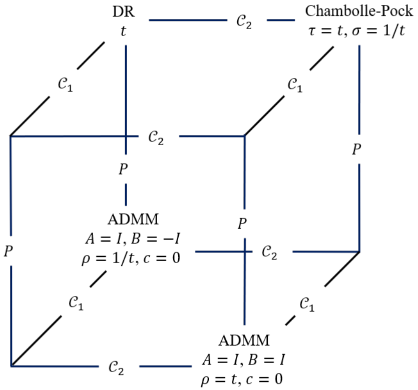
- converges for any $t > 0$ (but can be very slow)
- letting $y = tu$, equivalent to the iteration

$$\begin{aligned} x^{(k+1)} &= \underset{x}{\operatorname{argmin}}\, f(x) + t/2\|Ax + Bz^{(k)} - c + u^{(k)}\|^2 \\ z^{(k+1)} &= \underset{z}{\operatorname{argmin}}\, g(z) + t/2\|Ax^{(k+1)} + Bz - c + u^{(k)}\|^2 \\ u^{(k+1)} &= u^{(k)} + Ax^{(k+1)} + Bz^{(k+1)} - c \end{aligned}$$

- frequently used for distributed optimization:
  problems decouple if A or B is diagonal

Equivalence between iterative algorithms for optimization

Figure: relations between DR, ADMM, and Chambolle-Pock.



Source: [Zhao, Lessard, and Udell (2021)]

ADMM for statistical learning

Algorithm ADMM

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

ADMM for statistical learning

Algorithm ADMM

Input: loss function $\ell \circ A$, regularization r, stepsize ρ,
initial z^0, $u^0 = 0$
for $k = 0, 1, \dots$ **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 $\ell(Ax) + r(x)$

ADMM for statistical learning

Algorithm ADMM

```
Input: loss function ℓ ∘ A, regularization r, stepsize ρ,
initial z^0, u^0 = 0
for k = 0, 1, ... do
    x^{k+1} = argmin_x {ℓ(Ax) + ρ/2 ‖x − z^k + u^k‖_2^2}
    z^{k+1} = argmin_z {r(z) + ρ/2 ‖x^{k+1} − z + u^k‖_2^2}
    u^{k+1} = u^k + x^{k+1} − z^{k+1}
return x_⋆ (nearly) minimizing ℓ(Ax) + r(x)
```

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!

ADMM for statistical learning

Algorithm ADMM

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

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

solution: inexact ADMM

- solve x-min approximately with error ε^k
- converges if $\sum_k \varepsilon^k < \infty$ [Eckstein and Bertsekas (1992)]

ADMM for statistical learning

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: inexact ADMM

- solve x-min approximately with error ε^k
- converges if $\sum_k \varepsilon^k < \infty$ [Eckstein and Bertsekas (1992)]

add randNLA: use Nyström PCG to speed up x-min

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

Nyström PCG to solve ADMM subproblem

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

- $A^T H_\ell(x^k)A$ has data in it $\implies$ fast spectral decay
- stepsize ρ regularizes linear system
- if ℓ is quadratic (e.g., lasso and SVM), $H_\ell(x^k) = H_\ell$ is constant, so only need to sketch $A^T H_\ell A$ once

Nyström PCG to solve ADMM subproblem

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

- $A^T H_\ell(x^k) A$ has data in it $\implies$ fast spectral decay
- stepsize ρ regularizes linear system
- if ℓ is quadratic (e.g., lasso and SVM), $H_\ell(x^k) = H_\ell$ is constant, so only need to sketch $A^T H_\ell A$ once

in theory:

- solve to tolerance ϵ^k at iteration k, where $\sum_k \epsilon^k < \infty$
- if sketch size $s \approx d_{\text{eff}}(\rho)$, need $\leq O(\log(1/\epsilon^k))$ CG steps

Nyström PCG to solve ADMM subproblem

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

- $A^T H_\ell(x^k)A$ has data in it $\implies$ fast spectral decay
- stepsize ρ regularizes linear system
- if ℓ is quadratic (e.g., lasso and SVM), $H_\ell(x^k) = H_\ell$ is constant, so only need to sketch $A^T H_\ell A$ once

in theory:

- solve to tolerance ϵ^k at iteration k, where $\sum_k \epsilon^k < \infty$
- if sketch size $s \approx d_{\text{eff}}(\rho)$, need $\leq O(\log(1/\epsilon^k))$ CG steps

in practice:

- set $\epsilon^k = \text{geomean}(\text{primal resid}, \text{dual resid})$
- set sketch size $s = 50$

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 $\ell(Ax) + r(x)$

Source: Zhao, Frangella, and Udell, 2022

The competition

lasso:

- SSNAL, a Newton augmented Lagrangian method [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)]

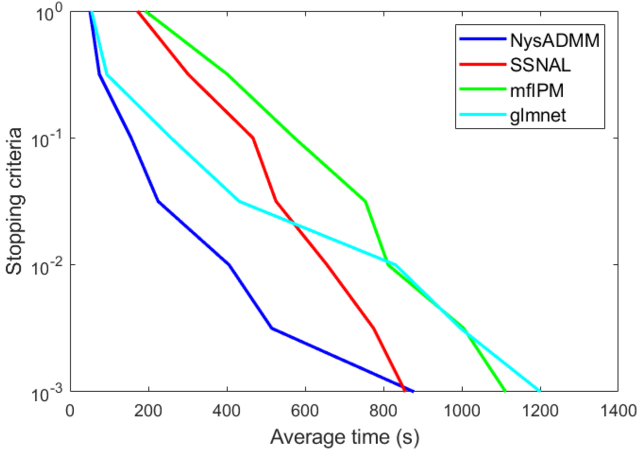
Numerical experiments: settings

- 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

What approximations are allowed?

Source: Frangella et al., 2023

Conclusion

low rank structure is everywhere! use it to accelerate

- top-k eigenvalue decomposition
- (regularized) linear system solve: $(A + \mu I)x = b$
- composite optimization: minimize $\ell(x) + r(x)$
- stochastic gradient descent
- ... your favorite problem ... ?