Big Data Analytics: Optimization and Randomization

Tianbao Yang[†], Qihang Lin[♮], Rong Jin[*‡]

Tutorial@SIGKDD 2015
Sydney, Australia

[†]Department of Computer Science, The University of Iowa, IA, USA
[♮]Department of Management Sciences, The University of Iowa, IA, USA
[*]Department of Computer Science and Engineering, Michigan State University, MI, USA
[‡]Institute of Data Science and Technologies at Alibaba Group, Seattle, USA

August 10, 2015

URL

http://www.cs.uiowa.edu/~tyng/kdd15-tutorial.pdf

Some Claims

No

- This tutorial is not an exhaustive literature survey
- It is not a survey on different machine learning/data mining algorithms

Yes

- It is about how to efficiently solve machine learning/data mining (formulated as optimization) problems for big data

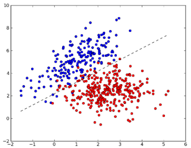
Outline

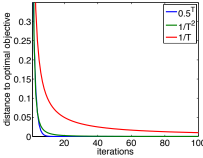
Big Data Analytics: Optimization and Randomization

Part I: Basics

Outline

Three Steps for Machine Learning





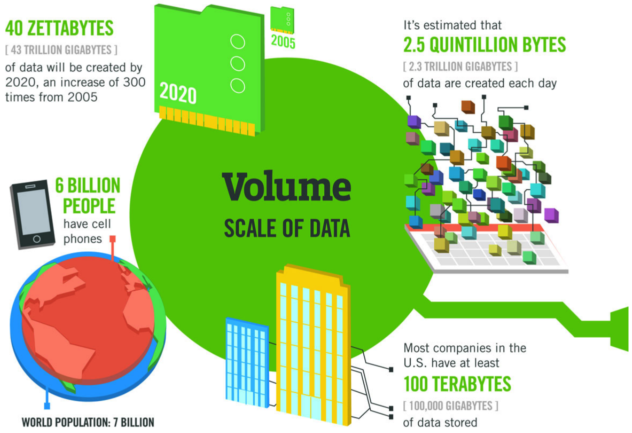
Data

Model

Optimization

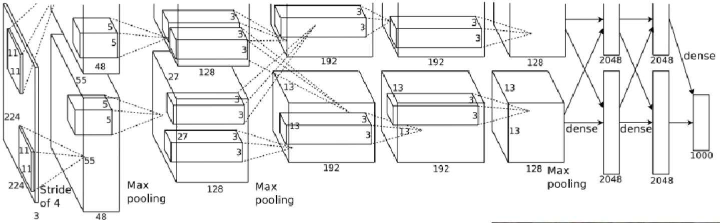
Big Data Challenge

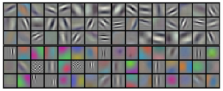
Big Data

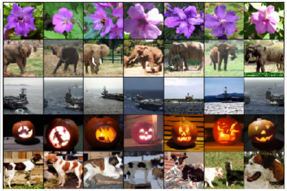

40 ZETTABYTES
[43 TRILLION GIGABYTES]
of data will be created by 2020, an increase of 300 times from 2005
2020
2005
It's estimated that
2.5 QUINTILLION BYTES
[2.3 TRILLION GIGABYTES]
of data are created each day
6 BILLION PEOPLE
have cell phones
Volume
SCALE OF DATA
WORLD POPULATION: 7 BILLION
Most companies in the U.S. have at least
100 TERABYTES
[100,000 GIGABYTES]
of data stored

Big Data Challenge

Big Model





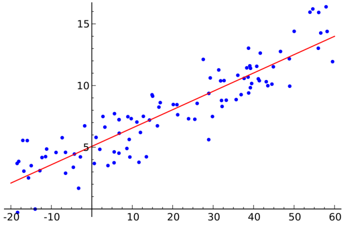


60 million parameters

Learning as Optimization

Ridge Regression Problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

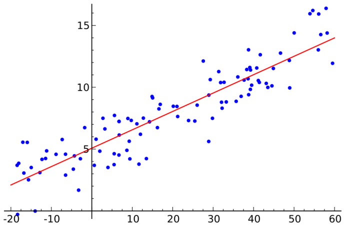


- $\mathbf{x}_i \in \mathbb{R}^d$: d-dimensional feature vector
- $y_i \in \mathbb{R}$: target variable
- $\mathbf{w} \in \mathbb{R}^d$: model parameters
- n: number of data points

Learning as Optimization

Ridge Regression Problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \underbrace{\frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2}_{\text{Empirical Loss}} + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

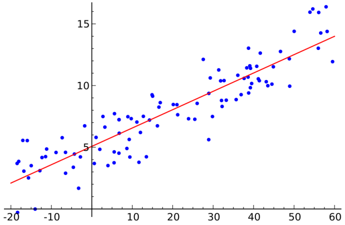


- $\mathbf{x}_i \in \mathbb{R}^d$: d-dimensional feature vector
- $y_i \in \mathbb{R}$: target variable
- $\mathbf{w} \in \mathbb{R}^d$: model parameters
- n: number of data points

Learning as Optimization

Ridge Regression Problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + \underbrace{\frac{\lambda}{2}\|\mathbf{w}\|_2^2}_{\text{Regularization}}$$

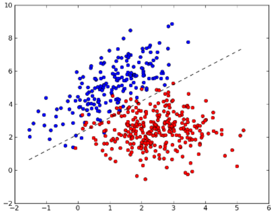


- $\mathbf{x}_i \in \mathbb{R}^d$: d-dimensional feature vector
- $y_i \in \mathbb{R}$: target variable
- $\mathbf{w} \in \mathbb{R}^d$: model parameters
- n: number of data points

Learning as Optimization

Classification Problems:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n}\ell(y_i\mathbf{w}^\top\mathbf{x}_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

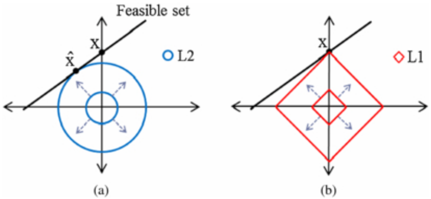


- $y_i \in \{+1, -1\}$: label
- Loss function $\ell(z)$: $z = y\mathbf{w}^\top\mathbf{x}$
  1. SVMs: (squared) hinge loss $\ell(z) = \max(0, 1-z)^p$, where $p = 1, 2$
  2. Logistic Regression: $\ell(z) = \log(1 + \exp(-z))$

Learning as Optimization

Feature Selection:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \lambda\|\mathbf{w}\|_1$$

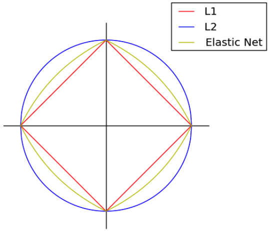


- ℓ_1 regularization $\|\mathbf{w}\|_1 = \sum_{i=1}^{d} |w_i|$
- λ controls sparsity level

Learning as Optimization

Feature Selection using Elastic Net:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{w}^\top\mathbf{x}_i, y_i)+\lambda\left(\|\mathbf{w}\|_1+\gamma\|\mathbf{w}\|_2^2\right)$$



- Elastic net regularizer, more robust than ℓ_1 regularizer

Learning as Optimization

Multi-class/Multi-task Learning:

$$\min_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{W}\mathbf{x}_i, y_i) + \lambda r(\mathbf{W})$$

- $\mathbf{W} \in \mathbb{R}^{K\times d}$
- $r(\mathbf{W}) = \|\mathbf{W}\|_F^2 = \sum_{k=1}^{K}\sum_{j=1}^{d} W_{kj}^2$: Frobenius Norm
- $r(\mathbf{W}) = \|\mathbf{W}\|_* = \sum_i \sigma_i$: Nuclear Norm (sum of singular values)
- $r(\mathbf{W}) = \|\mathbf{W}\|_{1,\infty} = \sum_{j=1}^{d} \|W_{:j}\|_\infty$: $\ell_{1,\infty}$mixed norm

Learning as Optimization

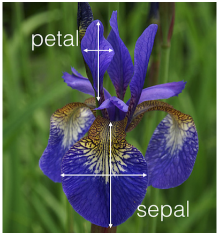
Regularized Empirical Loss Minimization

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + R(\mathbf{w})$$

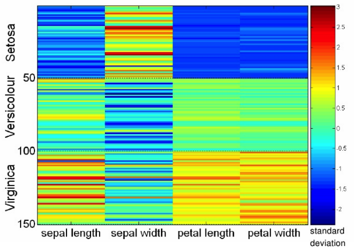
- Both ℓ and R are convex functions
- Extensions to Matrix Cases are possible (sometimes straightforward)
- Extensions to Kernel methods can be combined with randomized approaches
- Extensions to Non-convex (e.g., deep learning) are in progress

Data Matrices and Machine Learning

The Instance-feature Matrix: $X \in \mathbb{R}^{n \times d}$



$$X = \begin{pmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{x}_n^\top \end{pmatrix}$$



Data Matrices and Machine Learning

The output vector: $\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_n \end{pmatrix} \in \mathbb{R}^{n\times 1}$

- continuous $y_i \in \mathbb{R}$: regression (e.g., house price)
- discrete, e.g., $y_i \in \{1, 2, 3\}$: classification (e.g., species of iris)

Setosa

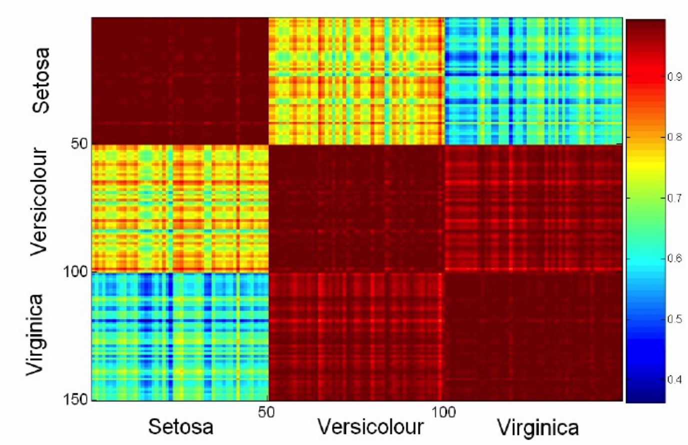
Versicolour

Virginica

Data Matrices and Machine Learning

The Instance-Instance Matrix: $K \in \mathbb{R}^{n \times n}$

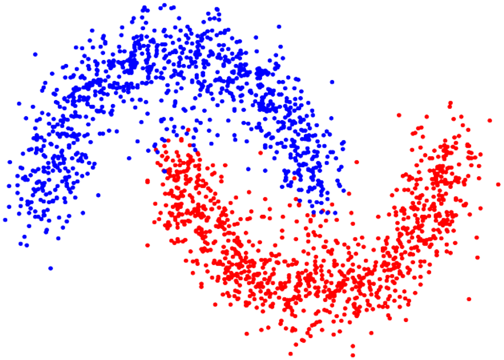
- Similarity Matrix
- Kernel Matrix



Data Matrices and Machine Learning

Some machine learning tasks are formulated on the kernel matrix

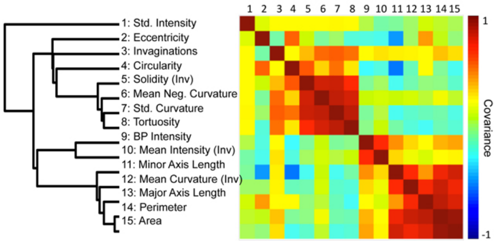
- Clustering
- Kernel Methods



Data Matrices and Machine Learning

The Feature-Feature Matrix: $C \in \mathbb{R}^{d \times d}$

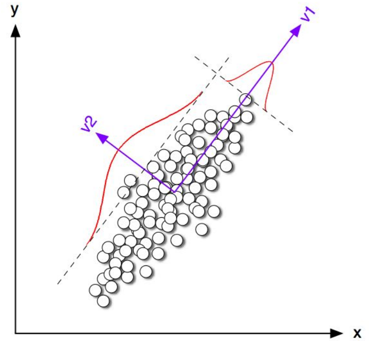
- Covariance Matrix
- Distance Metric Matrix



Data Matrices and Machine Learning

Some machine learning tasks requires the covariance matrix

- Principal Component Analysis
- Top-k Singular Value (Eigen-Value) Decomposition of the Covariance Matrix



Why Learning from Big Data is Challenging?

- High per-iteration cost
- High memory cost
- High communication cost
- Large iteration complexity

Outline

Norms

Vector $\mathbf{x} \in \mathbb{R}^d$

- Euclidean vector norm: $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^\top \mathbf{x}} = \sqrt{\sum_{i=1}^d x_i^2}$
- ℓ_p-norm of a vector: $\|\mathbf{x}\|_p = \left(\sum_{i=1}^d |x_i|^p\right)^{1/p}$ where $p \geq 1$
  1. ℓ_2 norm $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^d x_i^2}$
  2. ℓ_1 norm $\|\mathbf{x}\|_1 = \sum_{i=1}^d |x_i|$
  3. ℓ_∞ norm $\|\mathbf{x}\|_\infty = \max_i |x_i|$

Norms

Vector $\mathbf{x} \in \mathbb{R}^d$

- Euclidean vector norm: $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^\top \mathbf{x}} = \sqrt{\sum_{i=1}^d x_i^2}$
- ℓ_p-norm of a vector: $\|\mathbf{x}\|_p = \left(\sum_{i=1}^d |x_i|^p\right)^{1/p}$ where $p \geq 1$
  1. ℓ_2 norm $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^d x_i^2}$
  2. ℓ_1 norm $\|\mathbf{x}\|_1 = \sum_{i=1}^d |x_i|$
  3. ℓ_∞ norm $\|\mathbf{x}\|_\infty = \max_i |x_i|$

Norms

Vector $\mathbf{x} \in \mathbb{R}^d$

- Euclidean vector norm: $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^\top \mathbf{x}} = \sqrt{\sum_{i=1}^d x_i^2}$

- ℓ_p-norm of a vector: $\|\mathbf{x}\|_p = \left(\sum_{i=1}^d |x_i|^p\right)^{1/p}$ where $p \geq 1$
  1. ℓ_2 norm $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^d x_i^2}$
  2. ℓ_1 norm $\|\mathbf{x}\|_1 = \sum_{i=1}^d |x_i|$
  3. ℓ_∞ norm $\|\mathbf{x}\|_\infty = \max_i |x_i|$

Matrix Factorization

Matrix $X \in \mathbb{R}^{n \times d}$

- Singular Value Decomposition $X = U \Sigma V^\top$
  1. $U \in \mathbb{R}^{n \times r}$: orthonormal columns ($U^\top U = I$): span column space
  2. $\Sigma \in \mathbb{R}^{r \times r}$: diagonal matrix $\Sigma_{ii} = \sigma_i > 0$, $\sigma_1 \geq \sigma_2 \ldots \geq \sigma_r$
  3. $V \in \mathbb{R}^{d \times r}$: orthonormal columns ($V^\top V = I$): span row space
  4. $r \leq \min(n, d)$: max value such that $\sigma_r > 0$: rank of X
  5. $U_k \Sigma_k V_k^\top$: top-k approximation
- Pseudo inverse: $X^\dagger = V \Sigma^{-1} U^\top$
- QR factorization: $X = QR$ ($n \geq d$)
  - $Q \in \mathbb{R}^{n \times d}$: orthonormal columns
  - $R \in \mathbb{R}^{d \times d}$: upper triangular matrix

Matrix Factorization

Matrix $X \in \mathbb{R}^{n\times d}$

- Singular Value Decomposition $X = U\Sigma V^\top$
  1. $U \in \mathbb{R}^{n\times r}$: orthonormal columns ($U^\top U = I$): span column space
  2. $\Sigma \in \mathbb{R}^{r\times r}$: diagonal matrix $\Sigma_{ii} = \sigma_i > 0$, $\sigma_1 \geq \sigma_2 \ldots \geq \sigma_r$
  3. $V \in \mathbb{R}^{d\times r}$: orthonormal columns ($V^\top V = I$): span row space
  4. $r \leq \min(n, d)$: max value such that $\sigma_r > 0$: rank of X
  5. $U_k\Sigma_k V_k^\top$: top-k approximation
- Pseudo inverse: $X^\dagger = V\Sigma^{-1}U^\top$

- QR factorization: $X = QR$ ($n \geq d$)
  - $Q \in \mathbb{R}^{n\times d}$: orthonormal columns
  - $R \in \mathbb{R}^{d\times d}$: upper triangular matrix

Matrix Factorization

Matrix $X \in \mathbb{R}^{n\times d}$

- Singular Value Decomposition $X = U\Sigma V^\top$
  1. $U \in \mathbb{R}^{n\times r}$: orthonormal columns ($U^\top U = I$): span column space
  2. $\Sigma \in \mathbb{R}^{r\times r}$: diagonal matrix $\Sigma_{ii} = \sigma_i > 0$, $\sigma_1 \geq \sigma_2 \ldots \geq \sigma_r$
  3. $V \in \mathbb{R}^{d\times r}$: orthonormal columns ($V^\top V = I$): span row space
  4. $r \leq \min(n, d)$: max value such that $\sigma_r > 0$: rank of X
  5. $U_k \Sigma_k V_k^\top$: top-k approximation
- Pseudo inverse: $X^\dagger = V\Sigma^{-1}U^\top$

- QR factorization: $X = QR$ $(n \geq d)$
  - $Q \in \mathbb{R}^{n\times d}$: orthonormal columns
  - $R \in \mathbb{R}^{d\times d}$: upper triangular matrix

Norms

Matrix $X \in \mathbb{R}^{n\times d}$

- Frobenius norm: $\|X\|_F = \sqrt{tr(X^\top X)} = \sqrt{\sum_{i=1}^n \sum_{j=1}^d X_{ij}^2}$
- Spectral (induced norm) of a matrix: $\|X\|_2 = \max_{\|\mathbf{u}\|_2=1} \|X\mathbf{u}\|_2$
    - $\|A\|_2 = \sigma_1$ (maximum singular value)

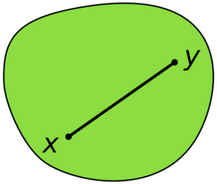
Norms

Matrix $X \in \mathbb{R}^{n \times d}$

- Frobenius norm: $\|X\|_F = \sqrt{tr(X^\top X)} = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{d} X_{ij}^2}$
- Spectral (induced norm) of a matrix: $\|X\|_2 = \max_{\|\mathbf{u}\|_2=1} \|X\mathbf{u}\|_2$
  - $\|A\|_2 = \sigma_1$ (maximum singular value)

Convex Optimization

$$\min_{x\in\mathcal{X}} f(x)$$

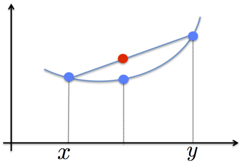


- $\mathcal{X}$ is a convex domain
  - for any $x, y \in \mathcal{X}$, their convex combination $\alpha x + (1-\alpha)y \in \mathcal{X}$

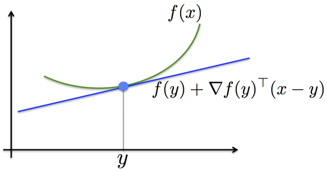
- $f(x)$ is a convex function

Convex Function

Characterization of Convex Function



$$f(\alpha x + (1-\alpha)y) \le \alpha f(x) + (1-\alpha) f(y), \\ \forall x, y \in \mathcal{X}, \alpha \in [0,1]$$

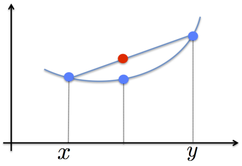


$$f(x) \ge f(y) + \nabla f(y)^\top (x-y) \; \forall x, y \in \mathcal{X}$$

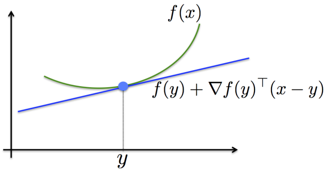
local optimum is global optimum

Convex Function

Characterization of Convex Function



$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y),$$
$$\forall x, y \in \mathcal{X}, \alpha \in [0,1]$$



$$f(x) \geq f(y) + \nabla f(y)^\top (x-y) \; \forall x, y \in \mathcal{X}$$

local optimum is global optimum

Convex vs Strongly Convex

Convex function:

$$f(x) \geq f(y) + \nabla f(y)^\top (x - y) \, \forall x, y \in \mathcal{X}$$

Strongly Convex function:

$$f(x) \geq f(y) + \nabla f(y)^\top (x - y) + \frac{\lambda}{2}\|x - y\|_2^2 \, \forall x, y \in \mathcal{X}$$

Global optimum is unique

Convex vs Strongly Convex

Convex function:

$$f(x) \geq f(y) + \nabla f(y)^\top (x - y) \, \forall x, y \in \mathcal{X}$$

strong convexity constant

Strongly Convex function:

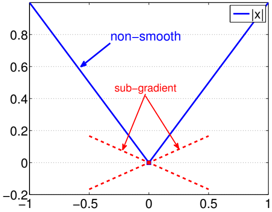
$$f(x) \geq f(y) + \nabla f(y)^\top (x - y) + \frac{\lambda}{2}\|x - y\|_2^2 \, \forall x, y \in \mathcal{X}$$

Global optimum is unique

Non-smooth function vs Smooth function

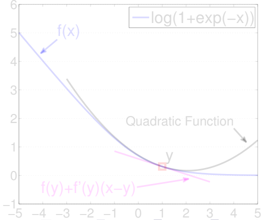
Non-smooth function

- Lipschitz continuous: e.g. absolute loss $f(x) = |x|$
- $|f(x) - f(y)| \leq G\|x - y\|_2$
- Subgradient: $f(x) \geq f(y) + \partial f(y)^\top (x - y)$



Smooth function

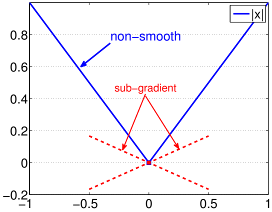
- e.g. logistic loss $f(x) = \log(1 + \exp(-x))$
- $\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2$



Non-smooth function vs Smooth function

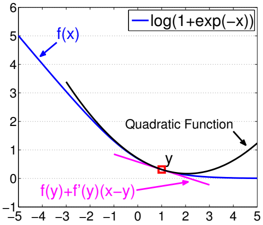
Non-smooth function

- Lipschitz continuous: e.g. absolute loss $f(x) = |x|$
- $|f(x) - f(y)| \leq G\|x - y\|_2$
- Subgradient: $f(x) \geq f(y) + \partial f(y)^\top (x - y)$



Smooth function

- e.g. logistic loss $f(x) = \log(1 + \exp(-x))$
- $\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2$

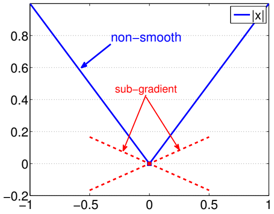


Non-smooth function vs Smooth function

Non-smooth function

Lipschitz constant

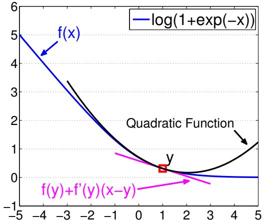
- Lipschitz continuous, e.g. absolute loss $f(x) = |x|$
- $|f(x) - f(y)| \leq G\|x - y\|_2$
- Subgradient: $f(x) \geq f(y) + \partial f(y)^\top (x - y)$



Smooth function

smoothness constant

- e.g. logistic loss $f(x) = \log(1 + \exp(-x))$
- $\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2$



Next ...

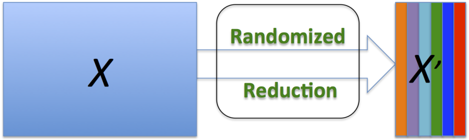
$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + R(\mathbf{w})$$

Part II: Optimization

- stochastic optimization
- distributed optimization

Reduce Iteration Complexity: utilizing properties of functions

Next ...



Part III: Randomization

- Classification, Regression
- SVD, K-means, Kernel methods

Reduce Data Size: utilizing properties of data

Please stay tuned!

Big Data Analytics: Optimization and Randomization

Part II: Optimization

Outline

Learning as Optimization

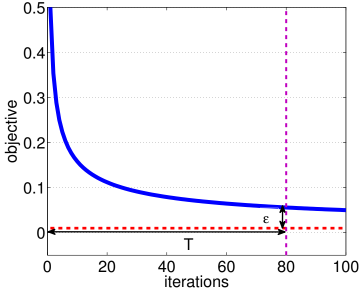
Regularized Empirical Loss Minimization

$$\min_{\mathbf{w}\in\mathbb{R}^d} \underbrace{\frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + R(\mathbf{w})}_{F(\mathbf{w})}$$

Convergence Measure

- Most optimization algorithms are iterative

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \Delta \mathbf{w}_t$$



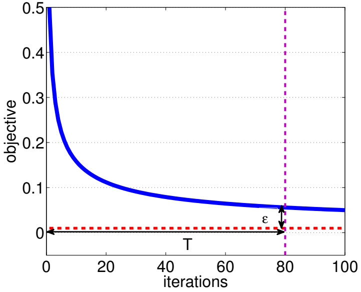
Convergence Measure

- Most optimization algorithms are iterative

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \Delta \mathbf{w}_t$$

- Iteration Complexity: the number of iterations $T(\epsilon)$ needed to have

$$F(\widehat{\mathbf{w}}_T) - \min_{\mathbf{w}} F(\mathbf{w}) \leq \epsilon \quad (\epsilon \ll 1)$$



Convergence Measure

- Most optimization algorithms are iterative

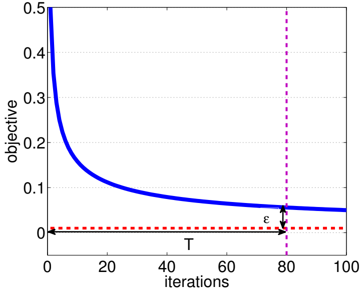
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \Delta \mathbf{w}_t$$

- Iteration Complexity: the number of iterations $T(\epsilon)$ needed to have

$$F(\widehat{\mathbf{w}}_T) - \min_{\mathbf{w}} F(\mathbf{w}) \leq \epsilon \quad (\epsilon \ll 1)$$

- Convergence Rate: after T iterations, how good is the solution

$$F(\widehat{\mathbf{w}}_T) - \min_{\mathbf{w}} F(\mathbf{w}) \leq \epsilon(T)$$



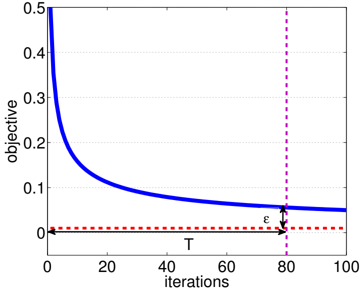
Convergence Measure

- Most optimization algorithms are iterative

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \Delta\mathbf{w}_t$$

- Iteration Complexity: the number of iterations $T(\epsilon)$ needed to have

$$F(\widehat{\mathbf{w}}_T) - \min_{\mathbf{w}} F(\mathbf{w}) \leq \epsilon \quad (\epsilon \ll 1)$$



- Convergence Rate: after T iterations, how good is the solution

$$F(\widehat{\mathbf{w}}_T) - \min_{\mathbf{w}} F(\mathbf{w}) \leq \epsilon(T)$$

- Total Runtime = Per-iteration Cost × Iteration Complexity

More on Convergence Measure

- Big $O(\cdot)$ notation: explicit dependence on T or ϵ

	Convergence Rate	Iteration Complexity
linear	$O\left(\mu^T\right) \quad (\mu < 1)$	$O\left(\log\left(\frac{1}{\epsilon}\right)\right)$
sub-linear	$O\left(\frac{1}{T^\alpha}\right) \quad \alpha > 0$	$O\left(\frac{1}{\epsilon^{1/\alpha}}\right)$

Why are we interested in Bounds?

More on Convergence Measure

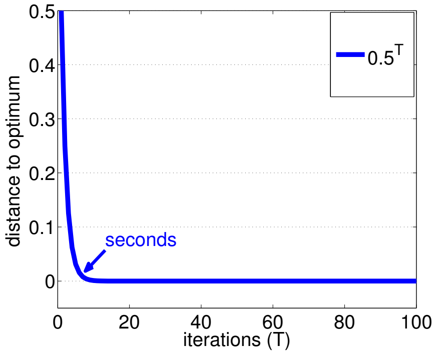
- Big $O(\cdot)$ notation: explicit dependence on T or ϵ

	Convergence Rate	Iteration Complexity
linear	$O\left(\mu^T\right) \quad (\mu < 1)$	$O\left(\log\left(\frac{1}{\epsilon}\right)\right)$
sub-linear	$O\left(\frac{1}{T^\alpha}\right) \quad \alpha > 0$	$O\left(\frac{1}{\epsilon^{1/\alpha}}\right)$

Why are we interested in Bounds?

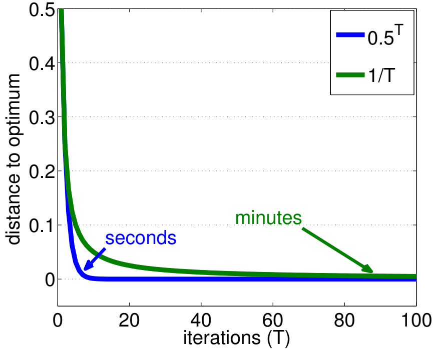
More on Convergence Measure

	Convergence Rate	Iteration Complexity
linear	$O(\mu^T) \quad (\mu < 1)$	$O\left(\log(\frac{1}{\epsilon})\right)$
sub-linear	$O(\frac{1}{T^\alpha}) \quad \alpha > 0$	$O\left(\frac{1}{\epsilon^{1/\alpha}}\right)$



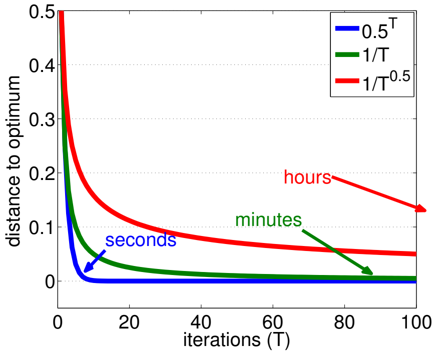
More on Convergence Measure

	Convergence Rate	Iteration Complexity
linear	$O(\mu^T) \quad (\mu < 1)$	$O\left(\log(\frac{1}{\epsilon})\right)$
sub-linear	$O(\frac{1}{T^\alpha}) \quad \alpha > 0$	$O\left(\frac{1}{\epsilon^{1/\alpha}}\right)$



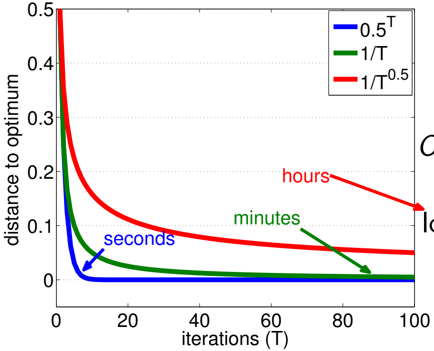
More on Convergence Measure

	Convergence Rate	Iteration Complexity
linear	$O(\mu^T) \quad (\mu < 1)$	$O\left(\log(\frac{1}{\epsilon})\right)$
sub-linear	$O(\frac{1}{T^\alpha}) \quad \alpha > 0$	$O\left(\frac{1}{\epsilon^{1/\alpha}}\right)$



More on Convergence Measure

	Convergence Rate	Iteration Complexity
linear	$O(\mu^T) \quad (\mu < 1)$	$O\left(\log(\frac{1}{\epsilon})\right)$
sub-linear	$O(\frac{1}{T^\alpha}) \quad \alpha > 0$	$O\left(\frac{1}{\epsilon^{1/\alpha}}\right)$



Theoretically, we consider

$$O(\mu^T) \prec O\left(\frac{1}{T^2}\right) \prec O\left(\frac{1}{T}\right) \prec O\left(\frac{1}{\sqrt{T}}\right)$$

$$\log\left(\frac{1}{\epsilon}\right) \prec \frac{1}{\sqrt{\epsilon}} \prec \frac{1}{\epsilon} \prec \frac{1}{\epsilon^2}$$

Non-smooth V.S. Smooth

- Non-smooth $\ell(z)$
  - hinge loss: $\ell(\mathbf{w}^\top\mathbf{x}, y) = \max(0, 1 - y\mathbf{w}^\top\mathbf{x})$
  - absolute loss: $\ell(\mathbf{w}^\top\mathbf{x}, y) = |\mathbf{w}^\top\mathbf{x} - y|$
- Smooth $\ell(z)$
  - squared hinge loss: $\ell(\mathbf{w}^\top\mathbf{x}, y) = \max(0, 1 - y\mathbf{w}^\top\mathbf{x})^2$
  - logistic loss: $\ell(\mathbf{w}^\top\mathbf{x}, y) = \log(1 + \exp(-y\mathbf{w}^\top\mathbf{x}))$
  - square loss: $\ell(\mathbf{w}^\top\mathbf{x}, y) = (\mathbf{w}^\top\mathbf{x} - y)^2$

Strong convex V.S. Non-strongly convex

- λ-strongly convex $R(\mathbf{w})$
  - ℓ_2 regularizer: $\frac{\lambda}{2}\|\mathbf{w}\|_2^2$
  - Elastic net regularizer: $\tau\|\mathbf{w}\|_1 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$
- Non-strongly convex $R(\mathbf{w})$
  - unregularized problem: $R(\mathbf{w}) \equiv 0$
  - ℓ_1 regularizer: $\tau\|\mathbf{w}\|_1$

Gradient Method in Machine Learning

$$F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- Suppose $\ell(z)$ is smooth
- Full gradient: $\nabla F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \lambda \mathbf{w}$
- Per-iteration cost: $O(nd)$

Gradient Descent

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t \nabla F(\mathbf{w}_{t-1})$$

Gradient Method in Machine Learning

$$F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- Suppose $\ell(z)$ is smooth
- Full gradient: $\nabla F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \lambda \mathbf{w}$
- Per-iteration cost: $O(nd)$

Gradient Descent

step size

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t \nabla F(\mathbf{w}_{t-1})$$

Gradient Method in Machine Learning

$$F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- If $\lambda = 0$: $R(\mathbf{w})$ is non-strongly convex
- Iteration complexity $O(\frac{1}{\epsilon})$

- If $\lambda > 0$: $R(\mathbf{w})$ is λ-strongly convex
- Iteration complexity $O(\frac{1}{\lambda}\log(\frac{1}{\epsilon}))$

Accelerated Gradient Method

Accelerated Gradient Descent

$$\mathbf{w}_t = \mathbf{v}_{t-1} - \gamma_t \nabla F(\mathbf{v}_{t-1})$$

$$\mathbf{v}_t = \mathbf{w}_t + \eta_t(\mathbf{w}_t - \mathbf{w}_{t-1})$$

- $\mathbf{w}_t$ is the output and $\mathbf{v}_t$ is an auxiliary sequence.

Accelerated Gradient Method

Accelerated Gradient Descent

$$\mathbf{w}_t = \mathbf{v}_{t-1} - \gamma_t \nabla F(\mathbf{v}_{t-1})$$

$$\mathbf{v}_t = \mathbf{w}_t + \eta_t(\mathbf{w}_t - \mathbf{w}_{t-1})$$

Momentum Step

- $\mathbf{w}_t$ is the output and $\mathbf{v}_t$ is an auxiliary sequence.

Accelerated Gradient Method

$$F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- If $\lambda = 0$: $R(\mathbf{w})$ is non-strongly convex
- Iteration complexity $O(\frac{1}{\sqrt{\epsilon}})$, better than $O(\frac{1}{\epsilon})$

- If $\lambda > 0$: $R(\mathbf{w})$ is λ-strongly convex
- Iteration complexity $O(\frac{1}{\sqrt{\lambda}}\log(\frac{1}{\epsilon}))$, better than $O(\frac{1}{\lambda}\log(\frac{1}{\epsilon}))$ for small λ

Deal with ℓ_1 regularizer

Consider a more general case

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{w}^\top\mathbf{x}_i, y_i) + \underbrace{R'(\mathbf{w}) + \tau\|\mathbf{w}\|_1}_{R(\mathbf{w})}$$

- $R(\mathbf{w}) = R'(\mathbf{w}) + \tau\|\mathbf{w}\|_1$
- $R'(\mathbf{w})$: λ-strongly convex and smooth

Deal with ℓ_1 regularizer

Consider a more general case

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \underbrace{\frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{w}^\top\mathbf{x}_i, y_i) + R'(\mathbf{w})}_{F'(\mathbf{w})} + \tau\|\mathbf{w}\|_1$$

- $R(\mathbf{w}) = R'(\mathbf{w}) + \tau\|\mathbf{w}\|_1$
- $R'(\mathbf{w})$: λ-strongly convex and smooth

Deal with ℓ_1 regularizer

Accelerated Gradient Descent

$$\mathbf{w}_t = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \nabla F'(\mathbf{v}_{t-1})^\top \mathbf{w} + \frac{1}{2\gamma_t}\|\mathbf{w}-\mathbf{v}_{t-1}\|_2^2 + \tau\|\mathbf{w}\|_1$$

$$\mathbf{v}_t = \mathbf{w}_t + \eta_t(\mathbf{w}_t - \mathbf{w}_{t-1})$$

- Proximal mapping has close-form solution: Soft-thresholding
- Iteration complexity and runtime remain unchanged.

Deal with ℓ_1 regularizer

Proximal mapping

Accelerated Gradient Descent

$$\mathbf{w}_t = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \nabla F'(\mathbf{v}_{t-1})^\top \mathbf{w} + \frac{1}{2\gamma_t}\|\mathbf{w}-\mathbf{v}_{t-1}\|_2^2 + \tau\|\mathbf{w}\|_1$$

$$\mathbf{v}_t = \mathbf{w}_t + \eta_t(\mathbf{w}_t - \mathbf{w}_{t-1})$$

- Proximal mapping has close-form solution: Soft-thresholding
- Iteration complexity and runtime remain unchanged.

Sub-Gradient Method in Machine Learning

$$F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- Suppose $\ell(z)$ is non-smooth
- Full sub-gradient: $\partial F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \partial\ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \lambda\mathbf{w}$

Sub-Gradient Descent

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t \partial F(\mathbf{w}_{t-1})$$

Sub-Gradient Method in Machine Learning

$$F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- Suppose $\ell(z)$ is non-smooth
- Full sub-gradient: $\partial F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \partial\ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \lambda\mathbf{w}$

Sub-Gradient Descent

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t \partial F(\mathbf{w}_{t-1})$$

Sub-Gradient Method

$$F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- If $\lambda = 0$: $R(\mathbf{w})$ is non-strongly convex
- Iteration complexity $O(\frac{1}{\epsilon^2})$

- If $\lambda > 0$: $R(\mathbf{w})$ is λ-strongly convex
- Iteration complexity $O(\frac{1}{\lambda\epsilon})$

- No efficient acceleration scheme in general

Problem Classes and Iteration Complexity

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + R(\mathbf{w})$$

- Iteration complexity

		$\ell(z) \equiv \ell(z, y)$	
		Non-smooth	Smooth
$R(\mathbf{w})$	Non-strongly convex	$O\left(\frac{1}{\epsilon^2}\right)$	$O\left(\frac{1}{\sqrt{\epsilon}}\right)$
	λ-strongly convex	$O\left(\frac{1}{\lambda\epsilon}\right)$	$O\left(\frac{1}{\sqrt{\lambda}}\log\left(\frac{1}{\epsilon}\right)\right)$

- Per-iteration cost: $O(nd)$, too high if n or d are large.

Outline

Stochastic First-Order Method by Data Sampling

- Stochastic Gradient Descent (SGD)
- Stochastic Variance Reduced Gradient (SVRG)
- Stochastic Average Gradient Algorithm (SAGA)
- Stochastic Dual Coordinate Ascent (SDCA)
- Accelerated Proximal Coordinate Gradient (APCG)

Assumption: $\|\mathbf{x}_i\| \leq 1$ for any i

Basic SGD (Nemirovski & Yudin (1978))

$$F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{w}^\top\mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- Full sub-gradient: $\partial F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\partial\ell(\mathbf{w}^\top\mathbf{x}_i, y_i) + \lambda\mathbf{w}$

- Randomly sample $i \in \{1, \ldots, n\}$
- Stochastic sub-gradient: $\partial\ell(\mathbf{w}^T\mathbf{x}_i, y_i) + \lambda\mathbf{w}$

$$\mathbb{E}_i[\partial\ell(\mathbf{w}^T\mathbf{x}_i, y_i) + \lambda\mathbf{w}] = \partial F(\mathbf{w})$$

Basic SGD (Nemirovski & Yudin (1978))

Applicable in all settings!

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{w}^\top\mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

sample: $i_t \in \{1, \ldots, n\}$

update: $\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t\left(\partial\ell(\mathbf{w}_{t-1}^T\mathbf{x}_{i_t}, y_{i_t}) + \lambda\mathbf{w}_{t-1}\right)$

output: $\overline{\mathbf{w}}_T = \frac{1}{T}\sum_{t=1}^{T}\mathbf{w}_t$

Basic SGD (Nemirovski & Yudin (1978))

Applicable in all settings!

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

$$\text{sample:} \quad i_t \in \{1,\ldots,n\}$$

$$\text{update:} \quad \mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t\left(\partial\ell(\mathbf{w}_{t-1}^T\mathbf{x}_{i_t}, y_{i_t}) + \lambda\mathbf{w}_{t-1}\right)$$

$$\text{output:} \quad \overline{\mathbf{w}}_T = \frac{1}{T}\sum_{t=1}^{T}\mathbf{w}_t$$

Basic SGD (Nemirovski & Yudin (1978))

$$F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{w}^\top\mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- If $\lambda = 0$: $R(\mathbf{w})$ is non-strongly convex
- Iteration complexity $O(\frac{1}{\epsilon^2})$

- If $\lambda > 0$: $R(\mathbf{w})$ is λ-strongly convex
- Iteration complexity $O(\frac{1}{\lambda\epsilon})$

- Exactly the same as sub-gradient descent!

Total Runtime

- Per-iteration cost: $O(d)$
- Much lower than full gradient method
- e.g. hinge loss (SVM)

$$\text{stochastic gradient: } \partial\ell(\mathbf{w}^\top \mathbf{x}_{i_t}, y_{i_t}) = \begin{cases} -y_{i_t}\mathbf{x}_{i_t}, & 1 - y_{i_t}\mathbf{w}^\top \mathbf{x}_{i_t} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Total Runtime

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + R(\mathbf{w})$$

- Iteration complexity

		$\ell(z) \equiv \ell(z, y)$	
		Non-smooth	Smooth
$R(\mathbf{w})$	Non-strongly convex	$O\left(\frac{1}{\epsilon^2}\right)$	$O\left(\frac{1}{\epsilon^2}\right)$
	λ-strongly convex	$O\left(\frac{1}{\lambda\epsilon}\right)$	$O\left(\frac{1}{\lambda\epsilon}\right)$

- For SGD, only strongly convexity helps but the smoothness does not make any difference!

Full Gradient V.S. Stochastic Gradient

- Full gradient method needs fewer iterations
- Stochastic gradient method has lower cost per iteration

- For small ϵ, use full gradient
- Satisfied with large ϵ, use stochastic gradient

- Full gradient can be accelerated
- Stochastic gradient cannot

- Full gradient's iterations complexity depends on smoothness and strong convexity
- Stochastic gradient's iteration complexity only depends on strong convexity

SVRG (Johnson & Zhang, 2013; Zhang et al., 2013; Xiao & Zhang, 2014)

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{w}^\top\mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- Applicable when $\ell(z)$ is smooth and $R(\mathbf{w})$ is λ-strongly convex
- Stochastic gradient:

$$g_{i_t}(\mathbf{w}) = \nabla\ell(\mathbf{w}^T\mathbf{x}_{i_t}, y_{i_t}) + \lambda\mathbf{w}$$

- $\mathrm{E}_{i_t}[g_{i_t}(\mathbf{w})] = \nabla F(\mathbf{w})$ but...
- $\mathrm{Var}[g_{i_t}(\mathbf{w})] \neq 0$ even if $\mathbf{w} = \mathbf{w}^\star$

SVRG (Johnson & Zhang, 2013; Zhang et al., 2013; Xiao & Zhang, 2014)

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- Applicable when $\ell(z)$ is smooth and $R(\mathbf{w})$ is λ-strongly convex
- Stochastic gradient:

$$g_{i_t}(\mathbf{w}) = \nabla\ell(\mathbf{w}^T \mathbf{x}_{i_t}, y_{i_t}) + \lambda\mathbf{w}$$

- $\mathrm{E}_{i_t}\left[g_{i_t}(\mathbf{w})\right] = \nabla F(\mathbf{w})$ but...
- $\mathrm{Var}\left[g_{i_t}(\mathbf{w})\right] \neq 0$ even if $\mathbf{w} = \mathbf{w}^\star$

SVRG (Johnson & Zhang, 2013; Zhang et al., 2013; Xiao & Zhang, 2014)

- Compute the full gradient at a reference point $\tilde{\mathbf{w}}$

$$\nabla F(\tilde{\mathbf{w}}) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\tilde{\mathbf{w}}^T \mathbf{x}_i, y_i) + \lambda \tilde{\mathbf{w}}$$

- Stochastic variance reduced gradient:

$$\tilde{g}_{i_t}(\mathbf{w}) = \nabla F(\tilde{\mathbf{w}}) - g_{i_t}(\tilde{\mathbf{w}}) + g_{i_t}(\mathbf{w})$$

- $\mathrm{E}_{i_t}\left[\tilde{g}_{i_t}(\mathbf{w})\right] = \nabla F(\mathbf{w})$
- $\mathrm{Var}\left[\tilde{g}_{i_t}(\mathbf{w})\right] \longrightarrow 0$ as $\tilde{\mathbf{w}}, \mathbf{w} \to \mathbf{w}^\star$

SVRG (Johnson & Zhang, 2013; Zhang et al., 2013; Xiao & Zhang, 2014)

- At optimal solution $\mathbf{w}^\star$: $\nabla F(\mathbf{w}^\star) = \mathbf{0}$
- It does not mean

$$g_{i_t}(\mathbf{w}) \longrightarrow \mathbf{0}$$

  as $\mathbf{w} \to \mathbf{w}^\star$
- However, we have

$$\tilde{g}_{i_t}(\mathbf{w}) = \nabla F(\tilde{\mathbf{w}}) - g_{i_t}(\tilde{\mathbf{w}}) + g_{i_t}(\mathbf{w}) \longrightarrow \mathbf{0}$$

  as $\tilde{\mathbf{w}}, \mathbf{w} \to \mathbf{w}^\star$

SVRG (Johnson & Zhang, 2013; Zhang et al., 2013; Xiao & Zhang, 2014)

```
Iterate $s = 1, \ldots, T-1$
    Let $\mathbf{w}_0 = \tilde{\mathbf{w}}_s$ and compute $\nabla F(\tilde{\mathbf{w}}_s)$
    Iterate $t = 1, \ldots, K$
        $\tilde{g}_{i_t}(\mathbf{w}_{t-1}) = \nabla F(\tilde{\mathbf{w}}_s) - g_{i_t}(\tilde{\mathbf{w}}_s) + g_{i_t}(\mathbf{w}_{t-1})$
        $\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t \tilde{g}_{i_t}(\mathbf{w}_{t-1})$
    $\tilde{\mathbf{w}}_{s+1} = \frac{1}{K}\sum_{t=1}^{K} \mathbf{w}_t$
output: $\tilde{\mathbf{w}}_T$
```

- $K = O\left(\frac{1}{\lambda}\right)$

SVRG (Johnson & Zhang, 2013; Zhang et al., 2013; Xiao & Zhang, 2014)

- Per-iteration cost: $O\left(d\left(n+\frac{1}{\lambda}\right)\right)$
- Iteration complexity

		$\ell(z) \equiv \ell(z, y)$	
		Non-smooth	Smooth
$R(\mathbf{w})$	Non-strongly convex	N.A.	N.A.
	λ-strongly convex	N.A.	$O\left(\log\left(\frac{1}{\epsilon}\right)\right)$

- Total Runtime: $O\left(d\left(n+\frac{1}{\lambda}\right)\log\left(\frac{1}{\epsilon}\right)\right)$
- Use proximal mapping for ℓ_1 regularizer

SVRG (Johnson & Zhang, 2013; Zhang et al., 2013; Xiao & Zhang, 2014)

- Per-iteration cost: $O\left(d\left(n+\frac{1}{\lambda}\right)\right)$
- Iteration complexity

		$\ell(z) \equiv \ell(z,y)$	
		Non-smooth	Smooth
$R(\mathbf{w})$	Non-strongly convex	N.A.	N.A.
	λ-strongly convex	N.A.	$O\left(\log\left(\frac{1}{\epsilon}\right)\right)$

- Total Runtime: $O\left(d\left(n+\frac{1}{\lambda}\right)\log\left(\frac{1}{\epsilon}\right)\right)$
- Use proximal mapping for ℓ_1 regularizer

SAGA (Defazio et al. (2014))

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{w}^\top\mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- A new version of SAG (Roux et al. (2012))
- Applicable when $\ell(z)$ is smooth
- Strong convexity is not required.

SAGA (Defazio et al. (2014))

- SAGA also reduces the variance of stochastic gradient but with a different technique
- SVRG uses gradients at the same point $\tilde{\mathbf{w}}$

$$\begin{aligned}\tilde{g}_{i_t}(\mathbf{w}) &= \nabla F(\tilde{\mathbf{w}}) - g_{i_t}(\tilde{\mathbf{w}}) + g_{i_t}(\mathbf{w})\\ \nabla F(\tilde{\mathbf{w}}) &= \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\tilde{\mathbf{w}}^T \mathbf{x}_i, y_i) + \lambda \tilde{\mathbf{w}}\end{aligned}$$

- SAGA uses gradients at different point $\{\tilde{\mathbf{w}}_1, \tilde{\mathbf{w}}_2, \cdots, \tilde{\mathbf{w}}_n\}$

$$\begin{aligned}\tilde{g}_{i_t}(\mathbf{w}) &= G - g_{i_t}(\tilde{\mathbf{w}}_{i_t}) + g_{i_t}(\mathbf{w})\\ G &= \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\tilde{\mathbf{w}}_i{}^T \mathbf{x}_i, y_i) + \lambda \tilde{\mathbf{w}}_{i_t}\end{aligned}$$

SAGA (Defazio et al. (2014))

- SAGA also reduces the variance of stochastic gradient but with a different technique
- SVRG uses gradients at the same point $\tilde{\mathbf{w}}$

$$\begin{aligned}\tilde{g}_{i_t}(\mathbf{w}) &= \nabla F(\tilde{\mathbf{w}}) - g_{i_t}(\tilde{\mathbf{w}}) + g_{i_t}(\mathbf{w}) \\ \nabla F(\tilde{\mathbf{w}}) &= \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\tilde{\mathbf{w}}^T \mathbf{x}_i, y_i) + \lambda \tilde{\mathbf{w}}\end{aligned}$$

- SAGA uses gradients at different point $\{\tilde{\mathbf{w}}_1, \tilde{\mathbf{w}}_2, \cdots, \tilde{\mathbf{w}}_n\}$

Average Gradient

$$\begin{aligned}\tilde{g}_{i_t}(\mathbf{w}) &= G - g_{i_t}(\tilde{\mathbf{w}}_{i_t}) + g_{i_t}(\mathbf{w}) \\ G &= \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\tilde{\mathbf{w}}_i{}^T \mathbf{x}_i, y_i) + \lambda \tilde{\mathbf{w}}_{i_t}\end{aligned}$$

SAGA (Defazio et al. (2014))

- Initialize average gradient G_0:

$$G_0 = \frac{1}{n}\sum_{i=1}^{n} g_i, \quad g_i = \nabla\ell(\mathbf{w}_0^\top \mathbf{x}_i, y_i) + \lambda \mathbf{w}_0$$

- stochastic variance reduced gradient:

$$\begin{aligned} \tilde{g}_{i_t}(\mathbf{w}_{t-1}) &= G_{t-1} - g_{i_t} + \left(\nabla\ell(\mathbf{w}_{t-1}^\top \mathbf{x}_{i_t}, y_{i_t}) + \lambda \mathbf{w}_{t-1}\right) \\ \mathbf{w}_t &= \mathbf{w}_{t-1} - \gamma_t \tilde{g}_{i_t}(\mathbf{w}_{t-1}) \end{aligned}$$

- Update average gradient

$$G_t = \frac{1}{n}\sum_{i=1}^{n} g_i, \quad g_{i_t} = \nabla\ell(\mathbf{w}_{t-1}^\top \mathbf{x}_{i_t}, y_{i_t}) + \lambda \mathbf{w}_{t-1}$$

SAGA (Defazio et al. (2014))

- Initialize average gradient G_0:

$$G_0 = \frac{1}{n}\sum_{i=1}^n g_i, \quad g_i = \nabla\ell(\mathbf{w}_0^\top \mathbf{x}_i, y_i) + \lambda \mathbf{w}_0$$

- stochastic variance reduced gradient:

$$\begin{aligned} \tilde{g}_{i_t}(\mathbf{w}_{t-1}) &= G_{t-1} - g_{i_t} + \left(\nabla\ell(\mathbf{w}_{t-1}^\top \mathbf{x}_{i_t}, y_{i_t}) + \lambda \mathbf{w}_{t-1}\right) \\ \mathbf{w}_t &= \mathbf{w}_{t-1} - \gamma_t \tilde{g}_{i_t}(\mathbf{w}_{t-1}) \end{aligned}$$

- Update average gradient

$$G_t = \frac{1}{n}\sum_{i=1}^n g_i, \quad g_{i_t} = \nabla\ell(\mathbf{w}_{t-1}^\top \mathbf{x}_{i_t}, y_{i_t}) + \lambda \mathbf{w}_{t-1}$$

SAGA (Defazio et al. (2014))

- Initialize average gradient G_0:

$$G_0 = \frac{1}{n}\sum_{i=1}^{n} g_i, \quad g_i = \nabla\ell(\mathbf{w}_0^\top \mathbf{x}_i, y_i) + \lambda \mathbf{w}_0$$

- stochastic variance reduced gradient:

$$\begin{aligned} \tilde{g}_{i_t}(\mathbf{w}_{t-1}) &= G_{t-1} - g_{i_t} + \left(\nabla\ell(\mathbf{w}_{t-1}^\top \mathbf{x}_{i_t}, y_{i_t}) + \lambda \mathbf{w}_{t-1}\right) \\ \mathbf{w}_t &= \mathbf{w}_{t-1} - \gamma_t \tilde{g}_{i_t}(\mathbf{w}_{t-1}) \end{aligned}$$

- Update average gradient

$$G_t = \frac{1}{n}\sum_{i=1}^{n} g_i, \quad g_{i_t} = \nabla\ell(\mathbf{w}_{t-1}^\top \mathbf{x}_{i_t}, y_{i_t}) + \lambda \mathbf{w}_{t-1}$$

SAGA: efficient update of averaged gradient

- G_t and G_{t-1} only differs in g_i for $i = i_t$
- Before we update g_i, we update

$$G_t = \frac{1}{n}\sum_{i=1}^{n} g_i = G_{t-1} - \frac{1}{n} g_{i_t} + \frac{1}{n}\left(\nabla \ell(\mathbf{w}_{t-1}^\top \mathbf{x}_{i_t}, y_{i_t}) + \lambda \mathbf{w}_{t-1}\right)$$

- computation cost: $O(d)$
- To implemente SAGA, we have to store and update all: $g_1, g_2, \ldots, g_n$
- Require extra memory space $O(nd)$

SAGA: efficient update of averaged gradient

- G_t and G_{t-1} only differs in g_i for $i = i_t$
- Before we update g_i, we update

$$G_t = \frac{1}{n}\sum_{i=1}^{n} g_i = G_{t-1} - \frac{1}{n} g_{i_t} + \frac{1}{n}\left(\nabla \ell(\mathbf{w}_{t-1}^\top \mathbf{x}_{i_t}, y_{i_t}) + \lambda \mathbf{w}_{t-1}\right)$$

- computation cost: $O(d)$
- To implemente SAGA, we have to store and update all: $g_1, g_2, \ldots, g_n$
- Require extra memory space $O(nd)$

SAGA (Defazio et al. (2014))

- Per-iteration cost: $O(d)$
- Iteration complexity

		$\ell(z) \equiv \ell(z, y)$	
		Non-smooth	Smooth
$R(\mathbf{w})$	Non-strongly convex	N.A.	$O\left(\frac{n}{\epsilon}\right)$
	λ-strongly convex	N.A.	$O\left(\left(n + \frac{1}{\lambda}\right) \log\left(\frac{1}{\epsilon}\right)\right)$

- Total Runtime (strongly convex): $O\left(d\left(n + \frac{1}{\lambda}\right) \log\left(\frac{1}{\epsilon}\right)\right)$. Same as SVRG!
- Use proximal mapping for ℓ_1 regularizer

SAGA (Defazio et al. (2014))

- Per-iteration cost: $O(d)$
- Iteration complexity

		$\ell(z) \equiv \ell(z, y)$	
		Non-smooth	Smooth
$R(\mathbf{w})$	Non-strongly convex	N.A.	$O\left(\frac{n}{\epsilon}\right)$
	λ-strongly convex	N.A.	$O\left(\left(n+\frac{1}{\lambda}\right)\log\left(\frac{1}{\epsilon}\right)\right)$

- Total Runtime (strongly convex): $O\left(d\left(n+\frac{1}{\lambda}\right)\log\left(\frac{1}{\epsilon}\right)\right)$. Same as SVRG!
- Use proximal mapping for ℓ_1 regularizer

Compare the Runtime of SGD and SVRG/SAGA

- Smooth but non-strongly convex:
  - SGD: $O\left(\frac{d}{\epsilon^2}\right)$
  - SAGA: $O\left(\frac{dn}{\epsilon}\right)$
- Smooth and strongly convex:
  - SGD: $O\left(\frac{d}{\lambda\epsilon}\right)$
  - SVRG/SAGA: $O\left(d\left(n+\frac{1}{\lambda}\right)\log\left(\frac{1}{\epsilon}\right)\right)$
- For small ϵ, use SVRG/SAGA
- Satisfied with large ϵ, use SGD

Conjugate Duality

- Define $\ell_i(z) \equiv \ell(z, y_i)$
- Conjugate function: $\ell_i^*(\alpha) \Longleftrightarrow \ell_i(z)$

$$\ell_i(z) = \max_{\alpha \in \mathbb{R}} \left[\alpha z - \ell^*(\alpha)\right], \quad \ell_i^*(\alpha) = \max_{z \in \mathbb{R}} \left[\alpha z - \ell(z)\right]$$

- E.g. hinge loss: $\ell_i(z) = \max(0, 1 - y_i z)$

$$\ell_i^*(\alpha) = \begin{cases} \alpha y_i & \text{if } -1 \leq \alpha y_i \leq 0 \\ +\infty & \text{otherwise} \end{cases}$$

- E.g. square hinge loss: $\ell_i(z) = \max(0, 1 - y_i z)^2$

$$\ell_i^*(\alpha) = \begin{cases} \frac{\alpha^2}{4} + \alpha y_i & \text{if } \alpha y_i \leq 0 \\ +\infty & \text{otherwise} \end{cases}$$

Conjugate Duality

- Define $\ell_i(z) \equiv \ell(z, y_i)$
- Conjugate function: $\ell_i^*(\alpha) \Longleftrightarrow \ell_i(z)$

$$\ell_i(z) = \max_{\alpha\in\mathbb{R}} \left[\alpha z - \ell^*(\alpha)\right], \quad \ell_i^*(\alpha) = \max_{z\in\mathbb{R}} \left[\alpha z - \ell(z)\right]$$

- E.g. hinge loss: $\ell_i(z) = \max(0, 1 - y_i z)$

$$\ell_i^*(\alpha) = \begin{cases} \alpha y_i & \text{if } -1 \leq \alpha y_i \leq 0 \\ +\infty & \text{otherwise} \end{cases}$$

- E.g. square hinge loss: $\ell_i(z) = \max(0, 1 - y_i z)^2$

$$\ell_i^*(\alpha) = \begin{cases} \frac{\alpha^2}{4} + \alpha y_i & \text{if } \alpha y_i \leq 0 \\ +\infty & \text{otherwise} \end{cases}$$

SDCA (Shalev-Shwartz & Zhang (2013))

- Stochastic Dual Coordinate Ascent (liblinear (Hsieh et al., 2008))
- Applicable when $R(\mathbf{w})$ is λ-strongly convex
- Smoothness is not required
- From Primal problem to Dual problem:

$$
\begin{aligned}
& \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\underbrace{\mathbf{w}^\top \mathbf{x}_i}_{z}, y_i) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \\
= \quad & \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \max_{\alpha_i \in \mathbb{R}} \left[\alpha_i (\mathbf{w}^\top \mathbf{x}_i) - \ell_i^*(\alpha_i) \right] + \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \\
= \quad & \max_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^{n} -\ell_i^*(\alpha_i) - \frac{\lambda}{2} \left\| \frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_i \mathbf{x}_i \right\|_2^2
\end{aligned}
$$

SDCA (Shalev-Shwartz & Zhang (2013))

- Stochastic Dual Coordinate Ascent (liblinear (Hsieh et al., 2008))
- Applicable when $R(\mathbf{w})$ is λ-strongly convex
- Smoothness is not required
- From Primal problem to Dual problem:

$$
\begin{aligned}
& \min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} \ell(\underbrace{\mathbf{w}^\top \mathbf{x}_i}_{z}, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 \\
= \quad & \min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} \max_{\alpha_i\in\mathbb{R}} \left[\alpha_i(\mathbf{w}^\top \mathbf{x}_i) - \ell_i^*(\alpha_i)\right] + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 \\
= \quad & \max_{\alpha\in\mathbb{R}^n} \frac{1}{n}\sum_{i=1}^{n} -\ell_i^*(\alpha_i) - \frac{\lambda}{2}\left\|\frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i\mathbf{x}_i\right\|_2^2
\end{aligned}
$$

SDCA (Shalev-Shwartz & Zhang (2013))

- Solve Dual Problem:

$$\max_{\alpha\in\mathbb{R}^n} \frac{1}{n}\sum_{i=1}^{n} -\ell_i^*(\alpha_i) - \frac{\lambda}{2}\left\|\frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i\mathbf{x}_i\right\|_2^2$$

- Sample $i_t \in \{1,\ldots,n\}$. Optimize α_{i_t} while fixing others

SDCA (Shalev-Shwartz & Zhang (2013))

- Maintain a primal solution: $\mathbf{w}_t = \frac{1}{\lambda n}\sum_{i=1}^n \alpha_i^t \mathbf{x}_i$
- Change variable $\alpha_i \longrightarrow \Delta\alpha_i$

$$
\begin{aligned}
& \max_{\Delta\alpha\in\mathbb{R}^n} \frac{1}{n}\sum_{i=1}^n -\ell_i^*(\alpha_i^t + \Delta\alpha_i) - \frac{\lambda}{2}\left\|\frac{1}{\lambda n}\left(\sum_{i=1}^n \alpha_i^t\mathbf{x}_i + \sum_{i=1}^n \Delta\alpha_i\mathbf{x}_i\right)\right\|_2^2 \\
\Longleftrightarrow \quad & \max_{\Delta\alpha\in\mathbb{R}^n} \frac{1}{n}\sum_{i=1}^n -\ell_i^*(\alpha_i^t + \Delta\alpha_i) - \frac{\lambda}{2}\left\|\mathbf{w}_t + \frac{1}{\lambda n}\sum_{i=1}^n \Delta\alpha_i\mathbf{x}_i\right\|_2^2
\end{aligned}
$$

SDCA (Shalev-Shwartz & Zhang (2013))

Dual Coordinate Updates

$$
\begin{aligned}
\Delta\alpha_{i_t} &= \max_{\Delta\alpha_{i_t}\in\mathbb{R}^n} -\frac{1}{n}\ell^*_{i_t}(-\alpha^t_{i_t} - \Delta\alpha_{i_t}) - \frac{\lambda}{2}\left\|\mathbf{w}_t + \frac{1}{\lambda n}\Delta\alpha_{i_t}\mathbf{x}_{i_t}\right\|_2^2 \\
\alpha^{t+1}_{i_t} &= \alpha^t_{i_t} + \Delta\alpha_{i_t} \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \frac{1}{\lambda n}\Delta\alpha_{i_t}\mathbf{x}_i
\end{aligned}
$$

SDCA updates

- Close-form solution for $\Delta\alpha_i$: hinge loss, squared hinge loss, absolute loss and square loss (Shalev-Shwartz & Zhang (2013))
- e.g. square loss

$$\Delta\alpha_i = \frac{y_i - \mathbf{w}_t^\top \mathbf{x}_i - \alpha_i^t}{1 + \|\mathbf{x}_i\|_2^2/(\lambda n)}$$

- Per-iteration cost: $O(d)$
- Approximate solution: logistic loss (Shalev-Shwartz & Zhang (2013))

SDCA

- Iteration complexity

		$\ell(z) \equiv \ell(z, y)$	
		Non-smooth	Smooth
$R(\mathbf{w})$	Non-strongly convex	N.A.	N.A.
	λ-strongly convex	$O\left(n + \frac{1}{\lambda\epsilon}\right)$	$O\left(\left(n + \frac{1}{\lambda}\right)\log\left(\frac{1}{\epsilon}\right)\right)$

- Total Runtime (smooth loss): $O\left(d\left(n + \frac{1}{\lambda}\right)\log\left(\frac{1}{\epsilon}\right)\right)$. The same as SVRG and SAGA!

SDCA

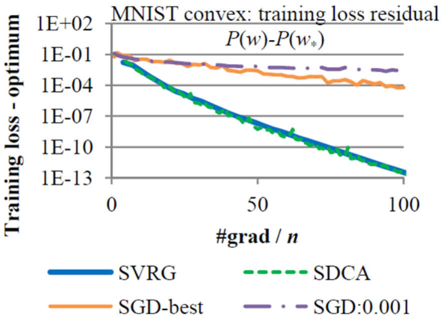
- Iteration complexity

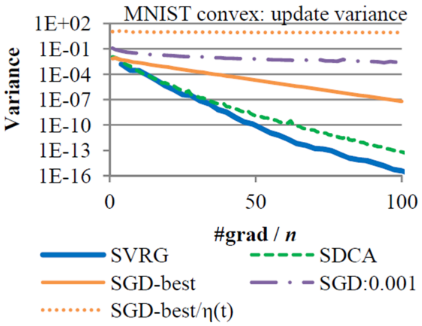
		$\ell(z) \equiv \ell(z, y)$	
		Non-smooth	Smooth
$R(\mathbf{w})$	Non-strongly convex	N.A.	N.A.
	λ-strongly convex	$O\left(n + \frac{1}{\lambda\epsilon}\right)$	$O\left(\left(n + \frac{1}{\lambda}\right) \log\left(\frac{1}{\epsilon}\right)\right)$

- Total Runtime (smooth loss): $O\left(d\left(n + \frac{1}{\lambda}\right) \log\left(\frac{1}{\epsilon}\right)\right)$. The same as SVRG and SAGA!

SVRG V.S. SDCA V.S. SGD

- ℓ_2-regularized logistic regression with $\lambda = 10^{-4}$
- MNIST data
- Johnson & Zhang (2013)





APCG (Lin et al. (2014))

- Recall the acceleration scheme for full gradient method
  - Auxiliary sequence (β^t)
  - Momentum step
- Maintain a primal solution: $\mathbf{w}_t = \frac{1}{\lambda n}\sum_{i=1}^n \beta_i^t \mathbf{x}_i$

Dual Coordinate Updates

$$\begin{aligned}
\Delta\beta_{i_t} &= \max_{\Delta\beta_{i_t}\in\mathbb{R}^n} -\frac{1}{n}\ell_{i_t}^*(-\beta_{i_t}^t - \Delta\beta_{i_t}) - \frac{\lambda}{2}\left\|\mathbf{w}_t + \frac{1}{\lambda n}\Delta\beta_{i_t}\mathbf{x}_{i_t}\right\|_2^2 \\
\alpha_{i_t}^{t+1} &= \beta_{i_t}^t + \Delta\beta_{i_t} \\
\beta^{t+1} &= \alpha^{t+1} + \eta_t(\alpha^{t+1} - \alpha^t)
\end{aligned}$$

APCG (Lin et al. (2014))

- Recall the acceleration scheme for full gradient method
  - Auxiliary sequence (β^t)
  - Momentum step
- Maintain a primal solution: $\mathbf{w}_t = \frac{1}{\lambda n}\sum_{i=1}^n \beta_i^t \mathbf{x}_i$

Dual Coordinate Updates

$$
\begin{aligned}
\Delta\beta_{i_t} &= \max_{\Delta\beta_{i_t}\in\mathbb{R}^n} -\frac{1}{n}\ell_{i_t}^*(-\beta_{i_t}^t - \Delta\beta_{i_t}) - \frac{\lambda}{2}\left\|\mathbf{w}_t + \frac{1}{\lambda n}\Delta\beta_{i_t}\mathbf{x}_{i_t}\right\|_2^2 \\
\alpha_{i_t}^{t+1} &= \beta_{i_t}^t + \Delta\beta_{i_t} \\
\beta^{t+1} &= \alpha^{t+1} + \eta_t(\alpha^{t+1} - \alpha^t)
\end{aligned}
$$

Momentum Step

APCG (Lin et al. (2014))

- Per-iteration cost: $O(d)$
- Iteration complexity

		$\ell(z) \equiv \ell(z, y)$	
		Non-smooth	Smooth
$R(\mathbf{w})$	Non-strongly convex	N.A.	N.A.
	λ-strongly convex	$O\left(n + \sqrt{\frac{n}{\lambda\epsilon}}\right)$	$O\left(\left(n + \sqrt{\frac{n}{\lambda}}\right) \log\left(\frac{1}{\epsilon}\right)\right)$

- Compared to SDCA, APCG has shorter runtime when λ is very small.

APCG V.S. SDCA

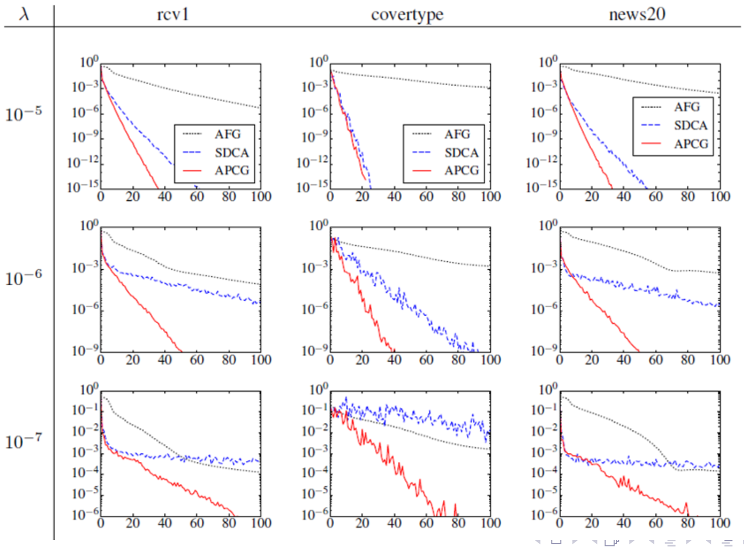
- squared hinge loss SVM
- real data

datasets	number of samples n	number of features d	sparsity
rcv1	20,242	47,236	0.16%
covtype	581,012	54	22%
news20	19,996	1,355,191	0.04%

- $F(\mathbf{w}_t) - F(\mathbf{w}^\star)$ V.S. the number of passes of data

APCG V.S. SDCA

Lin et al. (2014)



For general $R(\mathbf{w})$

- Dual Problem:

$$\max_{\alpha\in\mathbb{R}^n} \frac{1}{n}\sum_{i=1}^{n} -\ell_i^*(\alpha_i) - R^*\left(\frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i\mathbf{x}_i\right)$$

- R^* is the conjugate of R
- Sample $i_t \in \{1,\ldots,n\}$. Optimize α_{i_t} while fixing others
- Can be still updated in $O(d)$ in many cases (Shalev-Shwartz & Zhang (2013))
- Iteration complexity and runtime of SDCA and APCG remain unchanged.

APCG for primal problem

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{w}^\top\mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \tau\|\mathbf{w}\|_1$$

- Suppose $d >> n$. Per-iteration cost $O(d)$ is too high
- Apply APCG to the primal instead of dual problem
- Sample over features instead of data
- Per-iteration cost becomes $O(n)$

APCG for primal problem

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \frac{1}{2}\|X\mathbf{w}-\mathbf{y}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \tau\|\mathbf{w}\|_1$$

$$X = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]$$

- Full gradient: $\nabla F(\mathbf{w}) = X^T(X\mathbf{w}-\mathbf{y}) + \lambda\mathbf{w}$
- Partial gradient: $\nabla_i F(\mathbf{w}) = x_i^T(X\mathbf{w}-\mathbf{y}) + \lambda w_i$

Proximal Coordinate Gradient (PCG) (Nesterov (2012))

$$\mathbf{w}_i^t = \begin{cases} \arg\min_{w\in\mathbb{R}} \nabla_i F(\mathbf{w}^{t-1})w_i + \frac{1}{2\gamma_t}(w_i - \mathbf{w}_i^{t-1})^2 + \tau|w_i| & \text{if } i = t_i \\ \mathbf{w}_i^{t-1} & \text{otherwise} \end{cases}$$

- $\nabla_i F(\mathbf{w}^t)$ can be updated in $O(n)$

APCG for primal problem

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \frac{1}{2}\|X\mathbf{w}-\mathbf{y}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \tau\|\mathbf{w}\|_1$$

$$X = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]$$

- Full gradient: $\nabla F(\mathbf{w}) = X^T(X\mathbf{w}-\mathbf{y}) + \lambda\mathbf{w}$
- Partial gradient: $\nabla_i F(\mathbf{w}) = x_i^T(X\mathbf{w}-\mathbf{y}) + \lambda w_i$

Proximal Coordinate Gradient (PCG) (Nesterov (2012))

$$\mathbf{w}_i^t = \begin{cases} \arg\min_{w\in\mathbb{R}} \nabla_i F(\mathbf{w}^{t-1}) w_i + \frac{1}{2\gamma_t}(w_i - \mathbf{w}_i^{t-1})^2 + \tau|w_i| & \text{if } i = t_i \\ \mathbf{w}_i^{t-1} & \text{otherwise} \end{cases}$$

- $\nabla_i F(\mathbf{w}^t)$ can be updated in $O(n)$

APCG for primal problem

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \frac{1}{2}\|X\mathbf{w}-\mathbf{y}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \tau\|\mathbf{w}\|_1$$

$$X = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]$$

- Full gradient: $\nabla F(\mathbf{w}) = X^T(X\mathbf{w}-\mathbf{y}) + \lambda\mathbf{w}$
- Partial gradient: $\nabla_i F(\mathbf{w}) = x_i^T(X\mathbf{w}-\mathbf{y}) + \lambda w_i$

Proximal mapping

Proximal Coordinate Gradient (PCG) (Nesterov (2012))

$$\mathbf{w}_i^t = \begin{cases} \arg\min_{w\in\mathbb{R}} \nabla_i F(\mathbf{w}^{t-1}) w_i + \frac{1}{2\gamma_t}(w_i - \mathbf{w}_i^{t-1})^2 + \tau|w_i| & \text{if } i = t_i \\ \mathbf{w}_i^{t-1} & \text{otherwise} \end{cases}$$

- $\nabla_i F(\mathbf{w}^t)$ can be updated in $O(n)$

APCG for primal problem

- APCG accelerates PCG using
  - Auxiliary sequence ($\mathbf{v}^t$)
  - Momentum step

APCG

$$
\begin{aligned}
\mathbf{w}_i^t &= \begin{cases} \arg\min_{w_i \in \mathbb{R}} \nabla_i F(\mathbf{v}^{t-1}) w_i + \frac{1}{2\gamma_t}(w_i - \mathbf{v}_i^{t-1})^2 + \tau |w_i| & \text{if } i = t_i \\ \mathbf{w}_i^{t-1} & \text{otherwise} \end{cases} \\
\mathbf{v}^t &= \mathbf{w}^t + \eta_t (\mathbf{w}^t - \mathbf{w}^{t-1})
\end{aligned}
$$

APCG for primal problem

- Per-iteration cost: $O(n)$
- Iteration complexity

		$\ell(z) \equiv \ell(z, y)$	
		Non-smooth	Smooth
$R(\mathbf{w})$	Non-strongly convex	N.A.	$O\left(\frac{d}{\sqrt{\epsilon}}\right)$
	λ-strongly convex	N.A.	$O\left(\left(\frac{d}{\sqrt{\lambda}}\right)\log\left(\frac{1}{\epsilon}\right)\right)$

- $n >> d$: Apply APCG to the dual problem.
- $d >> n$: Apply APCG to the primal problem.

Which Algorithm to Use

- Satisfied with large ϵ: SGD
- For small ϵ:

		$\ell(z) \equiv \ell(z, y)$	
		Non-smooth	Smooth
$R(\mathbf{w})$	Non-strongly convex	SGD	SAGA
	λ-strongly convex	APCG	APCG

Summary

smooth	str-cvx	SGD	SAGA	SDCA	APCG
No	No	$\frac{1}{\epsilon^2}$	N.A.	N.A.	N.A.
Yes	No	$\frac{1}{\epsilon^2}$	$\frac{n}{\epsilon}$	N.A.	N.A.
No	Yes	$\frac{1}{\lambda\epsilon}$	N.A.	$n + \frac{1}{\epsilon}$	$n + \sqrt{\frac{n}{\lambda\epsilon}}$
Yes	Yes	$\frac{1}{\lambda\epsilon}$	$(n + \frac{1}{\lambda})\log(\frac{1}{\epsilon})$	$(n + \frac{1}{\lambda})\log(\frac{1}{\epsilon})$	$(n + \frac{1}{\sqrt{\lambda}})\log(\frac{1}{\epsilon})$

Table : Per-iteration cost: $O(d)$

smooth	str-cvx	SVRG
No	No	N.A.
Yes	No	N.A.
No	Yes	N.A.
Yes	Yes	$\log(\frac{1}{\epsilon})$

Table : Per-iteration cost: $O(d(n + \frac{1}{\lambda}))$

Summary

smooth	str-cvx	SGD	SAGA	SDCA	APCG
No	No	$\frac{1}{\epsilon^2}$	N.A.	N.A.	N.A.
Yes	No	$\frac{1}{\epsilon^2}$	$\frac{n}{\epsilon}$	N.A.	N.A.
No	Yes	$\frac{1}{\lambda\epsilon}$	N.A.	$n+\frac{1}{\epsilon}$	$n+\sqrt{\frac{n}{\lambda\epsilon}}$
Yes	Yes	$\frac{1}{\lambda\epsilon}$	$(n+\frac{1}{\lambda})\log(\frac{1}{\epsilon})$	$(n+\frac{1}{\lambda})\log(\frac{1}{\epsilon})$	$(n+\frac{1}{\sqrt{\lambda}})\log(\frac{1}{\epsilon})$

Table : Per-iteration cost: $O(d)$

smooth	str-cvx	SVRG
No	No	N.A.
Yes	No	N.A.
No	Yes	N.A.
Yes	Yes	$\log(\frac{1}{\epsilon})$

Table : Per-iteration cost: $O(d(n+\frac{1}{\lambda}))$

Summary

	SGD	SVRG	SAGA	SDCA	APCG
Memory	$O(d)$	$O(d)$	$O(dn)$	$O(d)$	$O(d)$
Parameters	γ_t	γ_t, K	γ_t	None	η_t
absolute					
hinge					
square					
squared hinge					
logistic					
$\lambda > 0$					
$\lambda = 0$					
Primal					
Dual					

Summary

	SGD	SVRG	SAGA	SDCA	APCG
Memory	$O(d)$	$O(d)$	$O(dn)$	$O(d)$	$O(d)$
Parameters	γ_t	γ_t, K	γ_t	None	η_t
absolute	✓	✗	✗	✓	✓
hinge	✓	✗	✗	✓	✓
square	✓	✓	✓	✓	✓
squared hinge	✓	✓	✓	✓	✓
logistic	✓	✓	✓	✓	✓
$\lambda > 0$					
$\lambda = 0$					
Primal					
Dual					

Summary

	SGD	SVRG	SAGA	SDCA	APCG
Memory	$O(d)$	$O(d)$	$O(dn)$	$O(d)$	$O(d)$
Parameters	γ_t	γ_t, K	γ_t	None	η_t
absolute	✓	✗	✗	✓	✓
hinge	✓	✗	✗	✓	✓
square	✓	✓	✓	✓	✓
squared hinge	✓	✓	✓	✓	✓
logistic	✓	✓	✓	✓	✓
$\lambda > 0$	✓	✓	✓	✓	✓
$\lambda = 0$	✓	✓	✓	✗	✗
Primal					
Dual					

Summary

	SGD	SVRG	SAGA	SDCA	APCG
Memory	$O(d)$	$O(d)$	$O(dn)$	$O(d)$	$O(d)$
Parameters	γ_t	γ_t, K	γ_t	None	η_t
absolute	✓	✗	✗	✓	✓
hinge	✓	✗	✗	✓	✓
square	✓	✓	✓	✓	✓
squared hinge	✓	✓	✓	✓	✓
logistic	✓	✓	✓	✓	✓
$\lambda > 0$	✓	✓	✓	✓	✓
$\lambda = 0$	✓	✓	✓	✗	✗
Primal	✓	✓	✓	✗	✓
Dual	✗	✗	✗	✓	✓

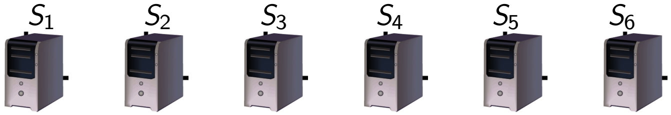
Outline

Big Data and Distributed Optimization

- data distributed over a cluster of multiple machines
- moving to single machine suffers
    - low network bandwidth
    - limited disk or memory
- communication V.S. computation
    - RAM 100 nanoseconds
    - standard network connection $250,000$ nanoseconds

Distributed Data

- N data points are partitioned and distributed to m machines
- $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N] = S_1 \cup S_2 \cup \cdots \cup S_m$
- Machine j only has access to S_j.
- W.L.O.G: $|S_j| = n = \frac{N}{m}$



A simple solution: Average Solution

- Global problem

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \left\{ F(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + R(\mathbf{w}) \right\}$$

- Machine j solves a local problem

$$\mathbf{w}_j = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \left\{ f_j(\mathbf{w}) = \frac{1}{n}\sum_{i\in S_j} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + R(\mathbf{w}) \right\}$$

S_1
S_2
S_3
S_4
S_5
S_6
$\mathbf{w}_1$
$\mathbf{w}_2$
$\mathbf{w}_3$
$\mathbf{w}_4$
$\mathbf{w}_5$
$\mathbf{w}_6$

Center computes: $\widehat{\mathbf{w}} = \frac{1}{m}\sum_{j=1}^{m} \mathbf{w}_j$, Issue: Will not converge to $\mathbf{w}^\star$

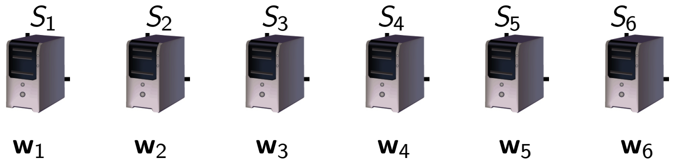
A simple solution: Average Solution

- Global problem

$$\mathbf{w}^{\star} = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \left\{ F(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell(\mathbf{w}^{\top}\mathbf{x}_i, y_i) + R(\mathbf{w}) \right\}$$

- Machine j solves a local problem

$$\mathbf{w}_j = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \left\{ f_j(\mathbf{w}) = \frac{1}{n}\sum_{i\in S_j} \ell(\mathbf{w}^{\top}\mathbf{x}_i, y_i) + R(\mathbf{w}) \right\}$$

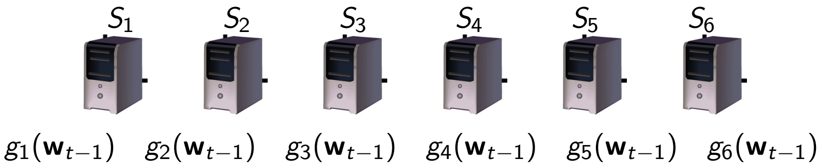


Center computes: $\widehat{\mathbf{w}} = \frac{1}{m}\sum_{j=1}^{m}\mathbf{w}_j$, Issue: Will not converge to $\mathbf{w}^{\star}$

Mini-Batch SGD: Average Stochastic Gradient

- Machine j sample $i_t \in S_j$ and construct a stochastic gradient

$$g_j(\mathbf{w}_{t-1}) = \partial\ell(\mathbf{w}_{t-1}^\top \mathbf{x}_{i_t}, y_{i_t}) + \partial R(\mathbf{w}_{t-1})$$



Center computes: $\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t \underbrace{\frac{1}{m}\sum_{j=1}^{m} g_j(\mathbf{w}_{t-1})}_{\text{Mini-batch SG}}$

Total Runtime

Single machine

- Total Runtime
  = Per-iteration Cost × Iteration Complexity

Distributed optimization

- Total Runtime
  = (Communication Time Per-round + Local Runtime Per-round)
  × Rounds of Communication

Mini-Batch SGD

Applicable in all settings!

- Communication Time Per-round: increase in m in a complicated way.
- Local Runtime Per-round: $O(1)$

Suppose $R(\mathbf{w})$ is λ-strongly convex

- Rounds of Communication: $O(\frac{1}{m\lambda\epsilon})$

Suppose $R(\mathbf{w})$ is non-strongly convex

- Rounds of Communication: $O(\frac{1}{m\epsilon^2})$

More machines reduce the rounds of communication but increase communication time per-round.

Mini-Batch SGD

Applicable in all settings!

- Communication Time Per-round: increase in m in a complicated way.
- Local Runtime Per-round: $O(1)$

Suppose $R(\mathbf{w})$ is λ-strongly convex

- Rounds of Communication: $O(\frac{1}{m\lambda\epsilon})$

Suppose $R(\mathbf{w})$ is non-strongly convex

- Rounds of Communication: $O(\frac{1}{m\epsilon^2})$

More machines reduce the rounds of communication but increase communication time per-round.

Distributed SDCA (Yang, 2013; Ma et al., 2015)

- Only works when $R(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2$ with $\lambda > 0$. (No ℓ_1)
- Global dual problem

$$\max_{\alpha\in\mathbb{R}^N} \frac{1}{N}\sum_{i=1}^N -\ell_i^*(\alpha_i) - \frac{\lambda}{2}\left\|\frac{1}{\lambda N}\sum_{i=1}^N \alpha_i\mathbf{x}_i\right\|_2^2$$

- $\alpha = [\alpha_{S_1}, \alpha_{S_2}, \cdots, \alpha_{S_m}]$
- Machine j solves a local dual problem only over α_{S_j}

$$\max_{\alpha_{S_j}\in\mathbb{R}^n} \frac{1}{N}\sum_{i\in S_j} -\ell_i^*(\alpha_i) - \frac{\lambda}{2}\left\|\frac{1}{\lambda N}\sum_{i=1}^N \alpha_i\mathbf{x}_i\right\|_2^2$$

Distributed SDCA (Yang, 2013; Ma et al., 2015)

- Only works when $R(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2$ with $\lambda > 0$. (No ℓ_1)
- Global dual problem

$$\max_{\alpha\in\mathbb{R}^N} \frac{1}{N}\sum_{i=1}^N -\ell_i^*(\alpha_i) - \frac{\lambda}{2}\left\|\frac{1}{\lambda N}\sum_{i=1}^N \alpha_i\mathbf{x}_i\right\|_2^2$$

- $\alpha = [\alpha_{S_1}, \alpha_{S_2}, \cdots, \alpha_{S_m}]$
- Machine j solves a local dual problem only over α_{S_j}

$$\max_{\alpha_{S_j}\in\mathbb{R}^n} \frac{1}{N}\sum_{i\in S_j} -\ell_i^*(\alpha_i) - \frac{\lambda}{2}\left\|\frac{1}{\lambda N}\sum_{i=1}^N \alpha_i\mathbf{x}_i\right\|_2^2$$

DisDCA (Yang, 2013),CoCoA+ (Ma et al., 2015)

- Center maintains a primal solution: $\mathbf{w}^t = \frac{1}{\lambda N}\sum_{i=1}^{N} \alpha_i^t \mathbf{x}_i$
- Change variable $\alpha_i \longrightarrow \Delta\alpha_i$

$$\max_{\Delta\alpha_{S_j}\in\mathbb{R}^n} \frac{1}{N}\sum_{i\in S_j} -\ell_i^*(\alpha_i^t + \Delta\alpha_i) - \frac{\lambda}{2}\left\|\frac{1}{\lambda N}\left(\sum_{i=1}^{N}\alpha_i^t\mathbf{x}_i + \sum_{i\in S_j}\Delta\alpha_i\mathbf{x}_i\right)\right\|_2^2$$

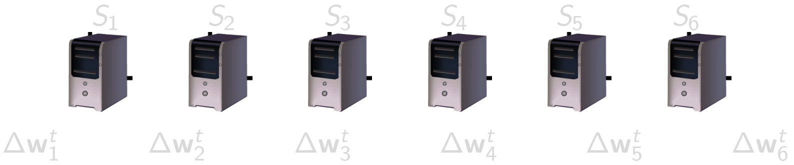
$$\Longleftrightarrow \quad \max_{\Delta\alpha_{S_j}\in\mathbb{R}^n} \frac{1}{N}\sum_{i\in S_j} -\ell_i^*(\alpha_i^t + \Delta\alpha_i) - \frac{\lambda}{2}\left\|\mathbf{w}^t + \frac{1}{\lambda N}\sum_{i\in S_j}\Delta\alpha_i\mathbf{x}_i\right\|_2^2$$

DisDCA (Yang, 2013),CoCoA+ (Ma et al., 2015)

- Machine j approximately solves

$$\Delta\alpha^t_{S_j} \approx \arg\max_{\Delta\alpha_{S_j}\in\mathbb{R}^n} \frac{1}{N}\sum_{i\in S_j} -\ell^*_i(\alpha^t_i + \Delta\alpha_i) - \frac{\lambda}{2}\left\|\mathbf{w}^t + \frac{1}{\lambda N}\sum_{i\in S_j}\Delta\alpha_i \mathbf{x}_i\right\|_2^2$$

$$\alpha^{t+1}_{S_j} = \alpha^t_{S_j} + \Delta\alpha^t_{S_j}, \quad \Delta\mathbf{w}^t_j = \frac{1}{\lambda N}\sum_{i\in S_j}\Delta\alpha^t_{S_j}\mathbf{x}_i$$



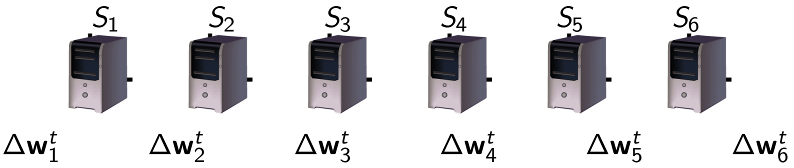
Center computes: $\mathbf{w}^{t+1} = \mathbf{w}^t + \sum_{j=1}^m \Delta\mathbf{w}^t_j$

DisDCA (Yang, 2013),CoCoA+ (Ma et al., 2015)

- Machine j approximately solves

$$\Delta\alpha^t_{S_j} \approx \arg\max_{\Delta\alpha_{S_j}\in\mathbb{R}^n} \frac{1}{N}\sum_{i\in S_j} -\ell^*_i(\alpha^t_i + \Delta\alpha_i) - \frac{\lambda}{2}\left\|\mathbf{w}^t + \frac{1}{\lambda N}\sum_{i\in S_j}\Delta\alpha_i\mathbf{x}_i\right\|_2^2$$

$$\alpha^{t+1}_{S_j} = \alpha^t_{S_j} + \Delta\alpha^t_{S_j}, \quad \Delta\mathbf{w}^t_j = \frac{1}{\lambda N}\sum_{i\in S_j}\Delta\alpha^t_{S_j}\mathbf{x}_i$$



Center computes: $\mathbf{w}^{t+1} = \mathbf{w}^t + \sum_{j=1}^m \Delta\mathbf{w}^t_j$

CoCoA+ (Ma et al., 2015)

- Local objective value

$$\mathcal{G}_j(\Delta\alpha_{S_j}, \mathbf{w}^t) = \frac{1}{N}\sum_{i\in S_j} -\ell_i^*(\alpha_i^t + \Delta\alpha_i) - \frac{\lambda}{2}\left\|\mathbf{w}^t + \frac{1}{\lambda N}\sum_{i\in S_j}\Delta\alpha_i \mathbf{x}_i\right\|_2^2$$

- Solve $\Delta\alpha_{S_j}^t$ by any local solver as long as

$$\left(\max_{\Delta\alpha_{S_j}} \mathcal{G}_j(\Delta\alpha_{S_j}, \mathbf{w}^t) - \mathcal{G}_j(\Delta\alpha_{S_j}^t, \mathbf{w}^t)\right) \leq \Theta\left(\max_{\Delta\alpha_{S_j}} \mathcal{G}_j(\Delta\alpha_{S_j}, \mathbf{w}^t) - \mathcal{G}_j(\mathbf{0}, \mathbf{w}^t)\right)$$

$$0 < \Theta < 1$$

CoCoA+ (Ma et al., 2015)

Suppose $\ell(z)$ is smooth, $R(\mathbf{w})$ is λ-strongly convex and SDCA is the local solver

- Local Runtime Per-round: $O((\frac{1}{\lambda} + \frac{N}{m}) \log \left(\frac{1}{\Theta}\right))$
- Rounds of Communication: $O(\frac{1}{1-\Theta} \frac{1}{\lambda} \log \left(\frac{1}{\epsilon}\right))$

Suppose $\ell(z)$ is non-smooth, $R(\mathbf{w})$ is λ-strongly convex and SDCA is the local solver

- Local Runtime Per-round: $O((\frac{1}{\lambda} + \frac{N}{m}) \frac{1}{\Theta})$
- Rounds of Communication: $O(\frac{1}{1-\Theta} \frac{1}{\lambda\epsilon})$

CoCoA+ (Ma et al., 2015)

Suppose $\ell(z)$ is smooth, $R(\mathbf{w})$ is λ-strongly convex and SDCA is the local solver

- Local Runtime Per-round: $O((\frac{1}{\lambda} + \frac{N}{m}) \log \left(\frac{1}{\Theta}\right))$
- Rounds of Communication: $O(\frac{1}{1-\Theta} \frac{1}{\lambda} \log \left(\frac{1}{\epsilon}\right))$

Suppose $\ell(z)$ is non-smooth, $R(\mathbf{w})$ is λ-strongly convex and SDCA is the local solver

- Local Runtime Per-round: $O((\frac{1}{\lambda} + \frac{N}{m}) \frac{1}{\Theta})$
- Rounds of Communication: $O(\frac{1}{1-\Theta} \frac{1}{\lambda\epsilon})$

Distributed SDCA in Practice

- Choice of Θ (how long we run the local solver?)
- Choice of m (how many machines to use?)

- Fast machines but slow network: Use small Θ and small m
- Fast network but slow machines: Use large Θ and large m

Distributed SDCA in Practice

- Choice of Θ (how long we run the local solver?)
- Choice of m (how many machines to use?)

- Fast machines but slow network: Use small Θ and small m
- Fast network but slow machines: Use large Θ and large m

DiSCO (Zhang & Xiao, 2015)

- The rounds of communication of Dual SDCA does not depends on m.

- DiSCO: Distributed Second-Order method (Zhang & Xiao, 2015)
- The rounds of communication of DiSCO depends on m.

DiSCO (Zhang & Xiao, 2015)

- Global problem

$$\min_{\mathbf{w}\in\mathbb{R}^d}\left\{F(\mathbf{w})=\frac{1}{N}\sum_{i=1}^{N}\ell(\mathbf{w}^\top\mathbf{x}_i, y_i)+R(\mathbf{w})\right\}$$

- Local problem

$$\min_{\mathbf{w}\in\mathbb{R}^d}\left\{f_j(\mathbf{w})=\frac{1}{n}\sum_{i\in S_j}\ell(\mathbf{w}^\top\mathbf{x}_i, y_i)+R(\mathbf{w})\right\}$$

- Global problem can be written as

$$\min_{\mathbf{w}\in\mathbb{R}^d}\left\{F(\mathbf{w})=\frac{1}{m}\sum_{j=1}^{m}f_j(\mathbf{w})\right\}$$

- Applicable when $f_j(\mathbf{w})$ is smooth, λ-strongly convex and self-concordant

Newton Direction

- At optimal solution $w^\star$:

$$\nabla F(\mathbf{w}^\star) = \mathbf{0}$$

- We hope moving $\mathbf{w}^t$ along $-\mathbf{v}^t$ leads to $\nabla F(\mathbf{w}^t - \mathbf{v}^t) = \mathbf{0}$
- Taylor expansion:

$$F(\mathbf{w}^t - \mathbf{v}^t) \approx \nabla F(\mathbf{w}^t) - \nabla^2 F(\mathbf{w}^t)\mathbf{v}^t = \mathbf{0}$$

- Such $\mathbf{v}^t$ is called a Newton's direction

Newton Method

Newton Method

Find a Netwon direction $\mathbf{v}_t$ by solving

$$\nabla^2 F(\mathbf{w}_t)\mathbf{v}_t = \nabla F(\mathbf{w}_t)$$

Then update

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma_t \mathbf{v}_t$$

Require solving a linear $d \times d$ equation system. Costly when $d > 1000$.

DiSCO (Zhang & Xiao, 2015)

Inexact Newton Method

Find an inexact Netwon direction $\mathbf{v}_t$ using Preconditioned Conjugate Gradient (PCG) (Golub & Ye, 1997)

$$\|\nabla^2 F(\mathbf{w}_t)\mathbf{v}_t - \nabla F(\mathbf{w}_t)\|_2 \leq \epsilon_t$$

Then update

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma_t \mathbf{v}_t$$

DiSCO (Zhang & Xiao, 2015)

PCG

Keep computing

$$\mathbf{v} \longleftarrow P^{-1} \times \nabla^2 F(\mathbf{w}_t) \times \mathbf{v}$$

iteratively until $\mathbf{v}$ becomes an inexact Newton direction.

- Preconditioner: $P = \nabla^2 f_1(\mathbf{w}_t) + \mu I$
- μ: a tuning parameter such that

$$\|\nabla^2 f_1(\mathbf{w}_t) - \nabla^2 F(\mathbf{w}_t)\|_2 \leq \mu$$

- f_1 is “similar” to F so that P is a good local preconditioner.
- $\mu = O(\frac{1}{\sqrt{n}}) = O(\sqrt{\frac{m}{N}})$

DiSCO (Zhang & Xiao, 2015)

PCG

Keep computing

$$\mathbf{v} \longleftarrow P^{-1} \times \nabla^2 F(\mathbf{w}_t) \times \mathbf{v}$$

iteratively until $\mathbf{v}$ becomes an inexact Newton direction.

- Preconditioner: $P = \nabla^2 f_1(\mathbf{w}_t) + \mu I$
- μ: a tuning parameter such that

$$\|\nabla^2 f_1(\mathbf{w}_t) - \nabla^2 F(\mathbf{w}_t)\|_2 \leq \mu$$

- f_1 is "similar" to F so that P is a good local preconditioner.
- $\mu = O(\frac{1}{\sqrt{n}}) = O(\sqrt{\frac{m}{N}})$

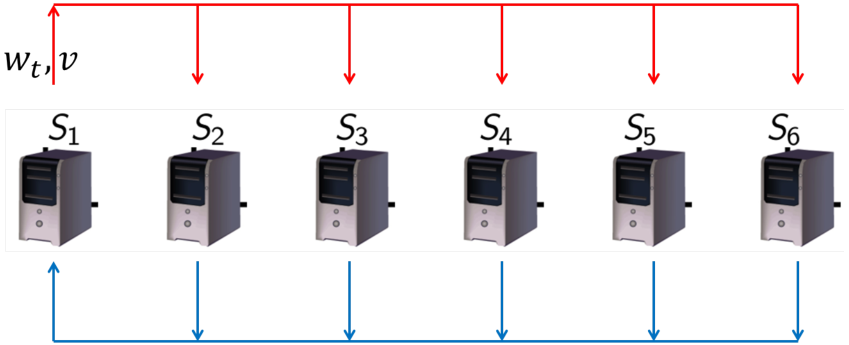
DiSCO (Zhang & Xiao, 2015)

DiSCO compute $\nabla^2 F(\mathbf{w}_t) \times \mathbf{v}$ distributedly:

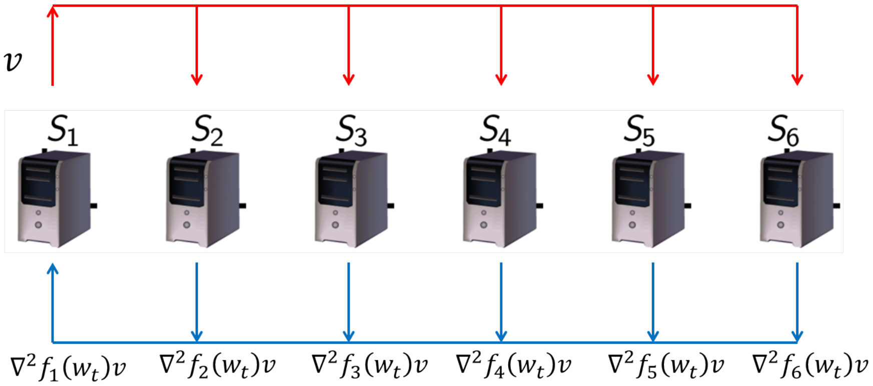
$$\begin{aligned} & \nabla^2 F(\mathbf{w}_t) \times \mathbf{v} \\ = \; & \underbrace{\nabla^2 f_1(\mathbf{w}_t) \times \mathbf{v}}_{\text{machine } 1} + \underbrace{\nabla^2 f_2(\mathbf{w}_t) \times \mathbf{v}}_{\text{machine } 2} + \cdots + \underbrace{\nabla^2 f_m(\mathbf{w}_t) \times}\end{aligned}$$

Then, compute $P^{-1} \times \nabla^2 F(\mathbf{w}_t) \times \mathbf{v}$ only in machine 1

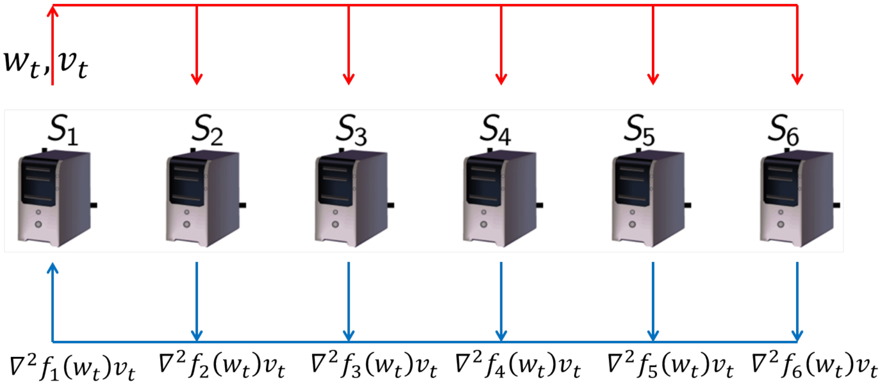
DiSCO (Zhang & Xiao, 2015)



DiSCO (Zhang & Xiao, 2015)

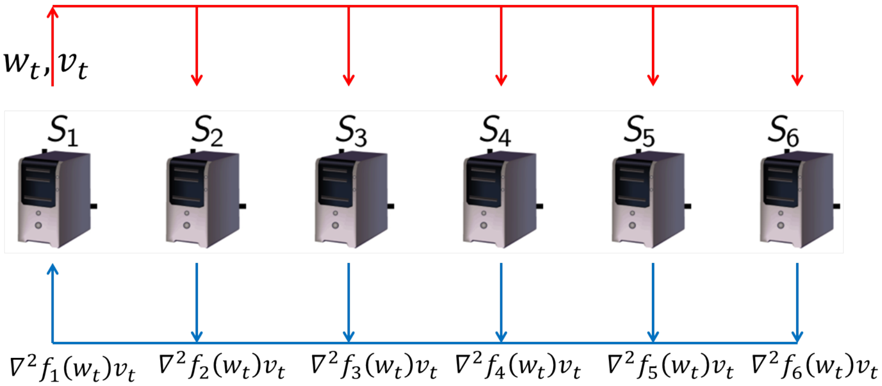


DiSCO (Zhang & Xiao, 2015)



Machine 1: $\mathrm{P}^{-1}(\nabla^2 F(w_t)v_t) = P^{-1}\left(\frac{1}{\mathrm{m}}\sum_{j=1} \nabla^2 f_j(w_t)v_t\right)$

DiSCO (Zhang & Xiao, 2015)



Machine 1: $$\mathrm{P}^{-1}(\nabla^2 F(w_t)v_t) = P^{-1}\left(\frac{1}{\mathrm{m}}\sum_{j=1} \nabla^2 f_j(w_t)v_t\right)$$

Run PCG until $||\nabla^2 F(w_t)v_t - \nabla F(w_t)||_2 \leq \epsilon_t$

DiSCO (Zhang & Xiao, 2015)

- For high-dimensional data (e.g. $d \geq 1000$), computing $P^{-1} \times \nabla^2 F(\mathbf{w}_t) \times \mathbf{v}$ is time costly.
- Instead, use SDCA in machine 1 to solve

$$P^{-1} \times \nabla^2 F(\mathbf{w}_t) \times \mathbf{v} \approx \arg\min_{\mathbf{u} \in \mathbb{R}^d} \frac{1}{2}\mathbf{u}^\top P\mathbf{u} - \mathbf{u}^\top F(\mathbf{w}_t)\mathbf{v}$$

- Local runtime: $O(\frac{N}{m} + \frac{1+\mu}{\lambda+\mu})$

DiSCO (Zhang & Xiao, 2015)

Suppose SDCA is the local solver

- Local Runtime Per-round: $O(\frac{N}{m} + \frac{1+\mu}{\lambda+\mu})$
- Rounds of Communication: $O(\sqrt{\frac{\mu}{\lambda}}\log\left(\frac{1}{\epsilon}\right))$

Choice of m (how many machines to use?). $(\mu = O(\sqrt{\frac{m}{N}}))$

- Fast machines but slow network: Use small m
- Fast network but slow machine: Use large m

DiSCO (Zhang & Xiao, 2015)

Suppose SDCA is the local solver

- Local Runtime Per-round: $O(\frac{N}{m} + \frac{1+\mu}{\lambda+\mu})$
- Rounds of Communication: $O(\sqrt{\frac{\mu}{\lambda}} \log\left(\frac{1}{\epsilon}\right))$

Choice of m (how many machines to use?). $(\mu = O(\sqrt{\frac{m}{N}}))$

- Fast machines but slow network: Use small m
- Fast network but slow machine: Use large m

DSVRG (Lee et al., 2015)

- A distributed version of SVRG using a "round-robin" scheme
- Assume the user can control the distribution of data before algorithm.
- Applicable when $f_j(\mathbf{w})$ is smooth and λ-strongly convex

DSVRG (Lee et al., 2015)

```
Iterate $s = 1, \ldots, T-1$
    Let $\mathbf{w}_0 = \tilde{\mathbf{w}}_s$ and compute $\nabla F(\tilde{\mathbf{w}}_s)$
    Iterate $t = 1, \ldots, K$
        $\tilde{g}_{i_t}(\mathbf{w}_{t-1}) = \nabla F(\tilde{\mathbf{w}}_s) - g_{i_t}(\tilde{\mathbf{w}}_s) + g_{i_t}(\mathbf{w}_{t-1})$
        $\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t \tilde{g}_{i_t}(\mathbf{w}_{t-1})$
    $\tilde{\mathbf{w}}_{s+1} = \frac{1}{K}\sum_{t=1}^{K} \mathbf{w}_t$
output: $\tilde{\mathbf{w}}_T$
```

- Each machine can only sample from its own data. However,
- $\mathbb{E}_{i_t \in S_j}[\tilde{g}_{i_t}(\mathbf{w}_{t-1})] \neq \nabla F(\mathbf{w}_{t-1})$

DSVRG (Lee et al., 2015)

Easy to distribute

```
Iterate $s = 1, \ldots, T-1$
    Let $\mathbf{w}_0 = \tilde{\mathbf{w}}_s$ and compute $\nabla F(\tilde{\mathbf{w}}_s)$
    Iterate $t = 1, \ldots, K$
        $\tilde{g}_{i_t}(\mathbf{w}_{t-1}) = \nabla F(\tilde{\mathbf{w}}_s) - g_{i_t}(\tilde{\mathbf{w}}_s) + g_{i_t}(\mathbf{w}_{t-1})$
        $\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t \tilde{g}_{i_t}(\mathbf{w}_{t-1})$
    $\tilde{\mathbf{w}}_{s+1} = \frac{1}{K}\sum_{t=1}^{K} \mathbf{w}_t$
output: $\tilde{\mathbf{w}}_T$
```

- Each machine can only sample from its own data. However,
- $\mathbb{E}_{i_t \in S_j}[\tilde{g}_{i_t}(\mathbf{w}_{t-1})] \neq \nabla F(\mathbf{w}_{t-1})$

DSVRG (Lee et al., 2015)

```
Iterate $s = 1, \ldots, T-1$
    Let $\mathbf{w}_0 = \tilde{\mathbf{w}}_s$ and compute $\nabla F(\tilde{\mathbf{w}}_s)$
    Iterate $t = 1, \ldots, K$
        $\tilde{g}_{i_t}(\mathbf{w}_{t-1}) = \nabla F(\tilde{\mathbf{w}}_s) - g_{i_t}(\tilde{\mathbf{w}}_s) + g_{i_t}(\mathbf{w}_{t-1})$
        $\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t \tilde{g}_{i_t}(\mathbf{w}_{t-1})$
    $\tilde{\mathbf{w}}_{s+1} = \frac{1}{K}\sum_{t=1}^{K} \mathbf{w}_t$
output: $\tilde{\mathbf{w}}_T$
```

Hard to distribute

- Each machine can only sample from its own data. However,
- $\mathbb{E}_{i_t \in S_j}[\tilde{g}_{i_t}(\mathbf{w}_{t-1})] \neq \nabla F(\mathbf{w}_{t-1})$

DSVRG (Lee et al., 2015)

```
Iterate $s = 1, \dots, T-1$
    Let $\mathbf{w}_0 = \tilde{\mathbf{w}}_s$ and compute $\nabla F(\tilde{\mathbf{w}}_s)$
    Iterate $t = 1, \dots, K$
        $\tilde{g}_{i_t}(\mathbf{w}_{t-1}) = \nabla F(\tilde{\mathbf{w}}_s) - g_{i_t}(\tilde{\mathbf{w}}_s) + g_{i_t}(\mathbf{w}_{t-1})$
        $\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t \tilde{g}_{i_t}(\mathbf{w}_{t-1})$
    $\tilde{\mathbf{w}}_{s+1} = \frac{1}{K}\sum_{t=1}^{K} \mathbf{w}_t$
output: $\tilde{\mathbf{w}}_T$
```

- Each machine can only sample from its own data. However,
- $\mathbb{E}_{i_t \in S_j}[\tilde{g}_{i_t}(\mathbf{w}_{t-1})] \neq \nabla F(\mathbf{w}_{t-1})$

DSVRG (Lee et al., 2015)

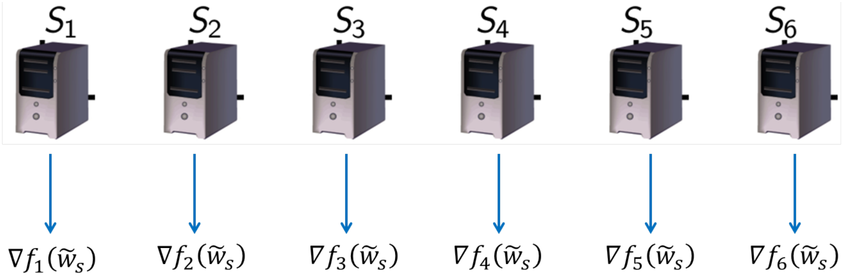
Solution:

- Store a second set of data R_j in machine j, which are sampled with replacement from $\{x_1, x_2, \ldots, x_n\}$ before the algorithm starts.
- Construct the stochastic gradient $\tilde{g}_{i_t}(\mathbf{w}_{t-1})$ by sampling $i_t \in R_j$ and removing i_t from R_j after.

$$\mathbb{E}_{i_t \in R_j}[\tilde{g}_{i_t}(\mathbf{w}_{t-1})] = \nabla F(\mathbf{w}_{t-1})$$

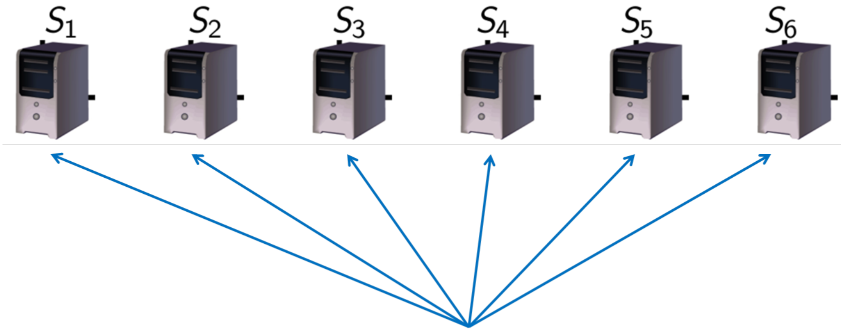
- When $R_j = \emptyset$, pass $\mathbf{w}_t$ to next machine.

Full Gradient Step


S_1
S_2
S_3
S_4
S_5
S_6
$\nabla f_1(\tilde{w}_s)$
$\nabla f_2(\tilde{w}_s)$
$\nabla f_3(\tilde{w}_s)$
$\nabla f_4(\tilde{w}_s)$
$\nabla f_5(\tilde{w}_s)$
$\nabla f_6(\tilde{w}_s)$

Center: $$\nabla F(\tilde{w}_s) = \frac{1}{\mathrm{m}} \sum_{j=1} \nabla f_j(\tilde{w}_s)$$

Full Gradient Step



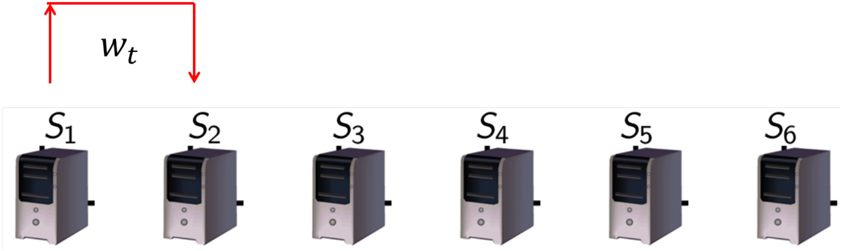
Center: $\nabla F(\widetilde{w}_s) = \frac{1}{m}\sum_{j=1} \nabla f_j(\widetilde{w}_s)$

Stochastic Gradient Step

Iterate $t = 1, \ldots, m$

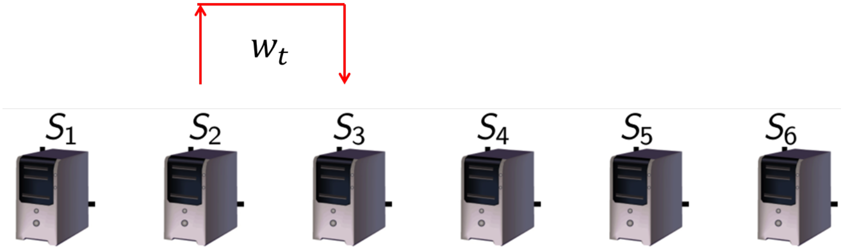
$$\tilde{g}_{i_t}(\mathbf{w}_{t-1}) = \nabla F(\tilde{\mathbf{w}}_s) - g_{i_t}(\tilde{\mathbf{w}}_s) + g_{i_t}(\mathbf{w}_{t-1})$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t \tilde{g}_{i_t}(\mathbf{w}_{t-1})$$



Stochastic Gradient Step

```
Iterate $t = 1, \dots, m$
    $\tilde{g}_{i_t}(\mathbf{w}_{t-1}) = \nabla F(\tilde{\mathbf{w}}_s) - g_{i_t}(\tilde{\mathbf{w}}_s) + g_{i_t}(\mathbf{w}_{t-1})$
    $\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t \tilde{g}_{i_t}(\mathbf{w}_{t-1})$
```

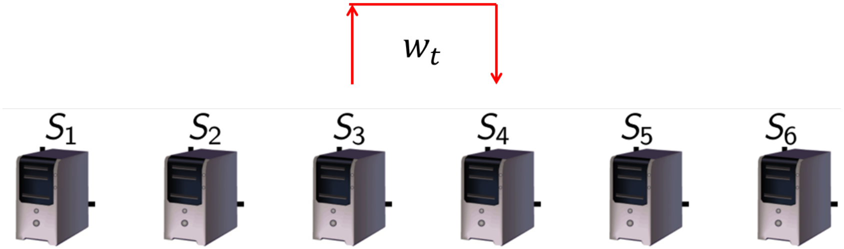


Stochastic Gradient Step

Iterate $t = 1, \ldots, m$

$$\tilde{g}_{i_t}(\mathbf{w}_{t-1}) = \nabla F(\tilde{\mathbf{w}}_s) - g_{i_t}(\tilde{\mathbf{w}}_s) + g_{i_t}(\mathbf{w}_{t-1})$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma_t \tilde{g}_{i_t}(\mathbf{w}_{t-1})$$



DSVRG (Lee et al., 2015)

Suppose $|R_j| = r$ for all j.

- Local Runtime Per-round: $O(\frac{N}{m} + r)$
- Rounds of Communication: $O(\frac{1}{r\lambda}\log\left(\frac{1}{\epsilon}\right))$

DSVRG (Lee et al., 2015)

Choice of m (how many machines to use?).

- Fast machines but slow network: Use small m
- Fast network but slow machine: Use large m

Choice of r (how many data points to pre-sample in R_j?).

- The larger, the better
- Required machine memory space: $|S_j| + |R_j| = \frac{N}{m} + r$

Other Distributed Optimization Methods

- ADMM (Boyd et al., 2011; Ozdaglar, 2015)
  - Rounds of Communication:
    $O(\text{Network Graph Dependency Term} \times \frac{1}{\sqrt{\lambda}} \log(\frac{1}{\epsilon}))$
- DANE (Shamir et al., 2014)
  - Approximate Newton direction with a difference approach from DISCO

Summary

$\ell(z)$ is smooth and $R(\mathbf{w})$ is λ-strongly convex.

alg.	Mini-SGD	Dist-SDCA	DiSCO
Runtime Per-round	$O(1)$	$O(\frac{1}{\lambda} + \frac{N}{m})$	$O(\frac{N}{m})$
Round of Comm.	$O(\frac{1}{\lambda m \epsilon})$	$O(\frac{1}{\lambda}\log(\frac{1}{\epsilon}))$	$O(\frac{m^{1/4}}{\sqrt{\lambda}N^{1/4}}\log(\frac{1}{\epsilon}))$

Table : Assume $\mu = O(\sqrt{\frac{m}{N}})$

alg.	DSVRG	
	Full Grad.	Stoch. Grad.
Runtime Per-round	$O(\frac{N}{m})$	$O(r)$
Round of Comm.	$O(\log(\frac{1}{\epsilon}))$	$O(\frac{1}{r\lambda}\log(\frac{1}{\epsilon}))$

Summary

$\ell(z)$ is non-smooth and $g(\mathbf{w})$ is λ-strongly convex.

alg.	Mini-SGD	Dist-SDCA	DiSCO	DSVRG
Runtime Per-round	$O(1)$	$O(\frac{1}{\lambda}+\frac{N}{m})$	N.A.	N.A.
Round of Comm.	$O(\frac{1}{\lambda m\epsilon})$	$O(\frac{1}{\lambda\epsilon})$	N.A.	N.A.

Summary

$g(\mathbf{w})$ is non-strongly convex.

alg.	Mini-SGD	Dist-SDCA	DiSCO	DSVRG
Runtime Per-round	$O(1)$	N.A.	N.A.	N.A.
Round of Comm.	$O(\frac{1}{m\epsilon^2})$	N.A.	N.A.	N.A.

Which Algorithm to Use

- Algorithm to use

		$\ell(z) \equiv \ell(z, y)$	
		Non-smooth	Smooth
$R(\mathbf{w})$	Non-strongly convex	Mini-SGD	DSVRG
	λ-strongly convex	Dist-SDCA	DiSCO/DSVRG

- Between DiSCO/DSVRG
  - Use DiSCO for small λ, e.g., $\lambda < 10^{-5}$
  - Use DSVRG for large λ, e.g., $\lambda > 10^{-5}$

Distributed Machine Learning Systems and Library

- Petuum: http://petuum.github.io
- Apache Spark: http://spark.apache.org/
- Parameter Server: http://parameterserver.org/
- Birds: http://cs.uiowa.edu/~tyng/software.html

Thank You! Questions?

Big Data Analytics: Optimization and Randomization

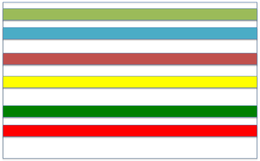
Part III: Randomization

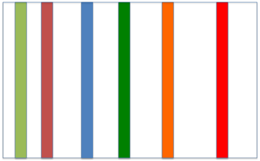
Outline

Random Sketch

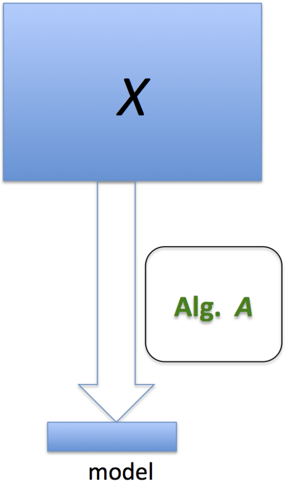
Approximate a large data matrix

by a much smaller sketch

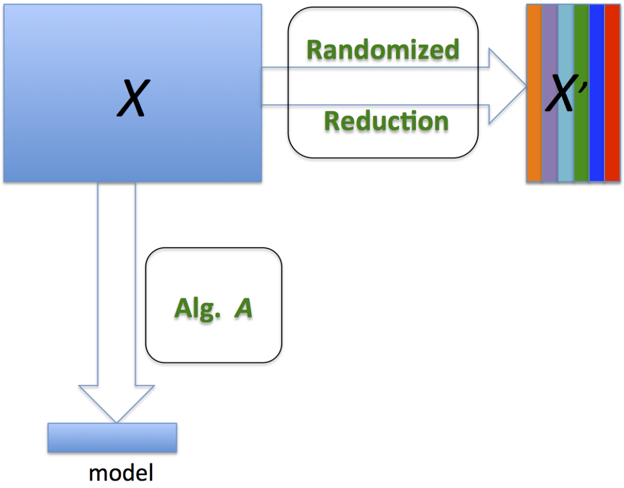




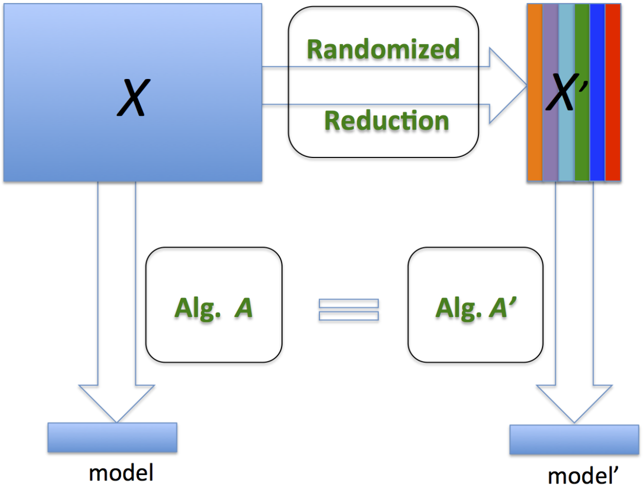
The Framework of Randomized Algorithms



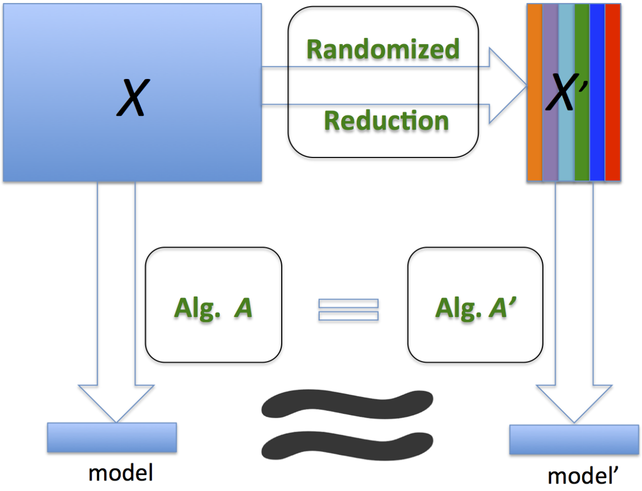
The Framework of Randomized Algorithms



The Framework of Randomized Algorithms



The Framework of Randomized Algorithms



Why randomized dimension reduction?

- Efficient
- Robust (e.g., dropout)
- Formal Guarantees
- Can explore parallel algorithms

Randomized Dimension Reduction

- Johnson-Lindenstauss (JL) transforms
- Subspace embeddings
- Column sampling

JL Lemma

JL Lemma (Johnson & Lindenstrauss, 1984)

For any $0 < \epsilon, \delta < 1/2$, there exists a probability distribution on $m \times d$ real matrices A such that there exists a small universal constant $c > 0$ and for any fixed $\mathbf{x} \in \mathbb{R}^d$ with a probability at least $1 - \delta$, we have

$$\left| \|A\mathbf{x}\|_2^2 - \|\mathbf{x}\|_2^2 \right| \leq c\sqrt{\frac{\log(1/\delta)}{m}}\|\mathbf{x}\|_2^2$$

or for $m = \Theta(\epsilon^{-2}\log(1/\delta))$, then with a probability at least $1 - \delta$

$$\left| \|A\mathbf{x}\|_2^2 - \|\mathbf{x}\|_2^2 \right| \leq \epsilon\|\mathbf{x}\|_2^2$$

Embedding a set of points into low dimensional space

Given a set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, we can embed them into a low dimensional space $A\mathbf{x}_1, \ldots, A\mathbf{x}_n \in \mathbb{R}^m$ such that
the pairwise distance between any two points are well preserved in the low dimensional space

$$\|A\mathbf{x}_i - A\mathbf{x}_j\|_2^2 = \|A(\mathbf{x}_i - \mathbf{x}_j)\|_2^2 \leq (1+\epsilon)\,\|\mathbf{x}_i - \mathbf{x}_j\|_2^2$$
$$\|A\mathbf{x}_i - A\mathbf{x}_j\|_2^2 = \|A(\mathbf{x}_i - \mathbf{x}_j)\|_2^2 \geq (1-\epsilon)\,\|\mathbf{x}_i - \mathbf{x}_j\|_2^2$$

In other words, in order to have all pairwise Euclidean distances preserved up to $1 \pm \epsilon$, only $m = \Theta(\epsilon^{-2}\log(n^2/\delta))$ dimensions are necessary

JL transforms: Gaussian Random Projection

Gaussian Random Projection (Dasgupta & Gupta, 2003): $A \in \mathbb{R}^{m\times d}$

- $A_{ij} \sim \mathcal{N}(0, 1/m)$
- $m = \Theta(\epsilon^{-2}\log(1/\delta))$
- Computational cost of AX: where $X \in \mathbb{R}^{d\times n}$
  - mnd for dense matrices
  - $\text{nnz}(X)m$ for sparse matrices

Computational Cost is very High (could be as high as solving many problems)

Accelerate JL transforms: using discrete distributions

Using Discrete Distributions (Achlioptas, 2003):

- $\Pr(A_{ij} = \pm\frac{1}{\sqrt{m}}) = 0.5$
- $\Pr(A_{ij} = \pm\sqrt{\frac{3}{m}}) = \frac{1}{6}$, $\Pr(A_{ij} = 0) = \frac{2}{3}$
- Database friendly
- Replace multiplications by additions and subtractions

Accelerate JL transforms: using Hadmard transform (I)

Fast JL transform based on randomized Hadmard transform:

Motivation: Can we simply use random sampling matrix $P \in \mathbb{R}^{m\times d}$ that randomly selects m coordinates out of d coordinates (scaled by $\sqrt{d/m}$)?

Unfortunately: by Chernoff bound

$$\left|\|P\mathbf{x}\|_2^2 - \|\mathbf{x}\|_2^2\right| \leq \frac{\sqrt{d}\|\mathbf{x}\|_\infty}{\|\mathbf{x}\|_2}\sqrt{\frac{3\log(2/\delta)}{m}}\|\mathbf{x}\|_2^2$$

Unless $\frac{\sqrt{d}\|\mathbf{x}\|_\infty}{\|\mathbf{x}\|_2} \leq c$, the random sampling doest not work

Remedy is given by randomized Hadmard transform

Accelerate JL transforms: using Hadmard transform (I)

Fast JL transform based on randomized Hadmard transform:

Motivation: Can we simply use random sampling matrix $P \in \mathbb{R}^{m\times d}$ that randomly selects m coordinates out of d coordinates (scaled by $\sqrt{d/m}$)?

Unfortunately: by Chernoff bound

$$\left| \|P\mathbf{x}\|_2^2 - \|\mathbf{x}\|_2^2 \right| \leq \frac{\sqrt{d}\|\mathbf{x}\|_\infty}{\|\mathbf{x}\|_2}\sqrt{\frac{3\log(2/\delta)}{m}}\|\mathbf{x}\|_2^2$$

Unless $\frac{\sqrt{d}\|\mathbf{x}\|_\infty}{\|\mathbf{x}\|_2} \leq c$, the random sampling doest not work

Remedy is given by randomized Hadmard transform

Accelerate JL transforms: using Hadmard transform (I)

Fast JL transform based on randomized Hadmard transform:

Motivation: Can we simply use random sampling matrix $P \in \mathbb{R}^{m\times d}$ that randomly selects m coordinates out of d coordinates (scaled by $\sqrt{d/m}$)?

Unfortunately: by Chernoff bound

$$\left|\|P\mathbf{x}\|_2^2 - \|\mathbf{x}\|_2^2\right| \leq \frac{\sqrt{d}\|\mathbf{x}\|_\infty}{\|\mathbf{x}\|_2}\sqrt{\frac{3\log(2/\delta)}{m}}\|\mathbf{x}\|_2^2$$

Unless $\frac{\sqrt{d}\|\mathbf{x}\|_\infty}{\|\mathbf{x}\|_2} \leq c$, the random sampling doest not work

Remedy is given by randomized Hadmard transform

Randomized Hadmard transform

Hadmard transform:

- $H \in \mathbb{R}^{d \times d}$: $H = \sqrt{\frac{1}{d}} H_{2^k}$

$$H_1 = [1], \quad H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad H_{2^k} = \begin{bmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{bmatrix}$$

- $\|H\mathbf{x}\|_2 = \|\mathbf{x}\|_2$ and H is orthogonal
- Computational costs of Hx: $d \log(d)$

randomized Hadmard transform: HD

- $D \in \mathbb{R}^{d \times d}$: a diagonal matrix $\Pr(D_{ii} = \pm 1) = 0.5$
- HD is orthogonal and $\|HD\mathbf{x}\|_2 = \|\mathbf{x}\|_2$

Key property: $\dfrac{\sqrt{d}\|HD\mathbf{x}\|_\infty}{\|HD\mathbf{x}\|_2} \leq \sqrt{\log(d/\delta)}$ w.h.p $1 - \delta$

Randomized Hadmard transform

Hadmard transform:

- $H \in \mathbb{R}^{d\times d}$: $H = \sqrt{\frac{1}{d}} H_{2^k}$

$$H_1 = [1], \quad H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad H_{2^k} = \begin{bmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{bmatrix}$$

- $\|H\mathbf{x}\|_2 = \|\mathbf{x}\|_2$ and H is orthogonal
- Computational costs of Hx: $d\log(d)$

randomized Hadmard transform: HD

- $D \in \mathbb{R}^{d\times d}$: a diagonal matrix $\Pr(D_{ii} = \pm 1) = 0.5$
- HD is orthogonal and $\|HD\mathbf{x}\|_2 = \|\mathbf{x}\|_2$

Key property: $\frac{\sqrt{d}\|HD\mathbf{x}\|_\infty}{\|HD\mathbf{x}\|_2} \leq \sqrt{\log(d/\delta)}$ w.h.p $1-\delta$

Randomized Hadmard transform

Hadmard transform:

- $H \in \mathbb{R}^{d\times d}$: $H = \sqrt{\frac{1}{d}} H_{2^k}$

$$H_1 = [1], \quad H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad H_{2^k} = \begin{bmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{bmatrix}$$

- $\|H\mathbf{x}\|_2 = \|\mathbf{x}\|_2$ and H is orthogonal
- Computational costs of Hx: $d\log(d)$

randomized Hadmard transform: HD

- $D \in \mathbb{R}^{d\times d}$: a diagonal matrix $\Pr(D_{ii} = \pm 1) = 0.5$
- HD is orthogonal and $\|HD\mathbf{x}\|_2 = \|\mathbf{x}\|_2$

Key property: $\frac{\sqrt{d}\|HD\mathbf{x}\|_\infty}{\|HD\mathbf{x}\|_2} \leq \sqrt{\log(d/\delta)}$ w.h.p $1-\delta$

Accelerate JL transforms: using Hadmard transform (I)

Fast JL transform based on randomized Hadmard transform (Tropp, 2011):

$$A = \sqrt{\frac{d}{m}} PHD$$

yields

$$\left| \|A\mathbf{x}\|_2^2 - \|\mathbf{x}\|_2^2 \right| \leq \sqrt{\frac{3\log(2/\delta)\log(d/\delta)}{m}} \|\mathbf{x}\|_2^2$$

- $m = \Theta(\epsilon^{-2}\log(1/\delta)\log(d/\delta))$ suffice for $1 \pm \epsilon$
- additional factor $\log(d/\delta)$ can be removed
- Computational cost of AX: $O(nd\log(m))$

Accelerate JL transforms: using a sparse matrix (I)

Random hashing (Dasgupta et al., 2010)

$$A = HD$$

where $D \in \mathbb{R}^{d\times d}$ and $H \in \mathbb{R}^{m\times d}$

- random hashing: $h(j) : \{1, \ldots, d\} \to \{1, \ldots, m\}$
- $H_{ij} = 1$ if $h(j) = i$: sparse matrix (each column has only one non-zero entry)
- $D \in \mathbb{R}^{d\times d}$: a diagonal matrix $\Pr(D_{ii} = \pm 1) = 0.5$
- $[A\mathbf{x}]_j = \sum_{i:h(i)=j} x_i D_{ii}$

Technically speaking, random hashing does not satisfy JL lemma

Accelerate JL transforms: using a sparse matrix (I)

Random hashing (Dasgupta et al., 2010)

$$A = HD$$

where $D \in \mathbb{R}^{d\times d}$ and $H \in \mathbb{R}^{m\times d}$

- random hashing: $h(j) : \{1, \ldots, d\} \rightarrow \{1, \ldots, m\}$
- $H_{ij} = 1$ if $h(j) = i$: sparse matrix (each column has only one non-zero entry)
- $D \in \mathbb{R}^{d\times d}$: a diagonal matrix $\Pr(D_{ii} = \pm 1) = 0.5$
- $[A\mathbf{x}]_j = \sum_{i:h(i)=j} x_i D_{ii}$

Technically speaking, random hashing does not satisfy JL lemma

Accelerate JL transforms: using a sparse matrix (I)

- key properties:
  - $\mathrm{E}[\langle HD\mathbf{x}_1, HD\mathbf{x}_2\rangle] = \langle \mathbf{x}_1, \mathbf{x}_2\rangle$
  - and norm perserving $|\|HD\mathbf{x}\|_2^2 - \|\mathbf{x}\|_2^2| \leq \epsilon\|\mathbf{x}\|_2^2$, only when

$$\frac{\|\mathbf{x}\|_\infty}{\|\mathbf{x}\|_2} \leq \frac{1}{\sqrt{c}}$$

Apply randomized Hadmard transform P first: $\Theta(c\log(c/\delta))$ blocks of randomized Hadmard transform

$$\frac{\|P\mathbf{x}\|_\infty}{\|P\mathbf{x}\|_2} \leq \frac{1}{\sqrt{c}}$$

Accelerate JL transforms: using a sparse matrix (II)

Sparse JL transform based on block random hashing (Kane & Nelson, 2014)

$$A = \begin{bmatrix} \frac{1}{\sqrt{s}} Q_1 \\ \dots \\ \frac{1}{\sqrt{s}} Q_s \end{bmatrix}$$

- Each $Q_s \in \mathbb{R}^{v \times d}$ is an independent random hashing (HD) matrix
- Set $v = \Theta(\epsilon^{-1})$ and $s = \Theta(\epsilon^{-1} \log(1/\delta))$
- Computational Cost of AX: $O\left(\frac{nnz(X)}{\epsilon} \log\left[\frac{1}{\delta}\right]\right)$

Randomized Dimension Reduction

- Johnson-Lindenstauss (JL) transforms
- Subspace embeddings
- Column sampling

Subspace Embeddings

Definition: a subspace embedding given some parameters $0 < \epsilon, \delta < 1, k \leq d$ is a distribution $\mathcal{D}$ over matrices $A \in \mathbb{R}^{m \times d}$ such that for any fixed linear subspace $W \in \mathbb{R}^d$ with $dim(W) = k$ it holds that

$$\Pr_{A \sim \mathcal{D}}(\forall \mathbf{x} \in W, \|A\mathbf{x}\|_2 \in (1 \pm \epsilon)\|\mathbf{x}\|_2) \geq 1 - \delta$$

It implies

- If $U \in \mathbb{R}^{d \times k}$ is orthogonal matrix (contains the orthonormal bases)
  - $AU \in \mathbb{R}^{m \times k}$ is of full column rank
  - $\|AU\|_2 \in (1 \pm \epsilon)$
  - $(1 - \epsilon)^2 \leq \|U^\top A^\top AU\|_2 \leq (1 + \epsilon)^2$
- These are key properties in the theoretical analysis of many algorithms (e.g., low-rank matrix approximation, randomized least-squares regression, randomized classification)

Subspace Embeddings

Definition: a subspace embedding given some parameters $0 < \epsilon, \delta < 1, k \leq d$ is a distribution $\mathcal{D}$ over matrices $A \in \mathbb{R}^{m \times d}$ such that for any fixed linear subspace $W \in \mathbb{R}^d$ with $dim(W) = k$ it holds that

$$\Pr_{A \sim \mathcal{D}}(\forall \mathbf{x} \in W, \|A\mathbf{x}\|_2 \in (1 \pm \epsilon)\|\mathbf{x}\|_2) \geq 1 - \delta$$

It implies

- If $U \in \mathbb{R}^{d \times k}$ is orthogonal matrix (contains the orthonormal bases)
  - $AU \in \mathbb{R}^{m \times k}$ is of full column rank
  - $\|AU\|_2 \in (1 \pm \epsilon)$
  - $(1 - \epsilon)^2 \leq \|U^\top A^\top AU\|_2 \leq (1 + \epsilon)^2$
- These are key properties in the theoretical analysis of many algorithms (e.g., low-rank matrix approximation, randomized least-squares regression, randomized classification)

Subspace Embeddings

Definition: a subspace embedding given some parameters $0 < \epsilon, \delta < 1, k \leq d$ is a distribution $\mathcal{D}$ over matrices $A \in \mathbb{R}^{m \times d}$ such that for any fixed linear subspace $W \in \mathbb{R}^d$ with $dim(W) = k$ it holds that

$$\Pr_{A \sim \mathcal{D}}(\forall \mathbf{x} \in W, \|A\mathbf{x}\|_2 \in (1 \pm \epsilon)\|\mathbf{x}\|_2) \geq 1 - \delta$$

It implies

- If $U \in \mathbb{R}^{d \times k}$ is orthogonal matrix (contains the orthonormal bases)
  - $AU \in \mathbb{R}^{m \times k}$ is of full column rank
  - $\|AU\|_2 \in (1 \pm \epsilon)$
  - $(1 - \epsilon)^2 \leq \|U^\top A^\top AU\|_2 \leq (1 + \epsilon)^2$
- These are key properties in the theoretical analysis of many algorithms (e.g., low-rank matrix approximation, randomized least-squares regression, randomized classification)

Subspace Embeddings

From a JL transform to a Subspace Embedding (Sarlós, 2006).
Let $A \in \mathbb{R}^{m \times d}$ be a JL transform. If

$$m = O\left(\frac{k \log\left[\frac{k}{\delta \epsilon}\right]}{\epsilon^2}\right)$$

Then w.h.p $1 - \delta^k$, $A \in \mathbb{R}^{m \times d}$ is a subspace embedding w.r.t a k-dimensional space in $\mathbb{R}^d$

Subspace Embeddings

Making block random hashing a Subspace Embedding (Nelson & Nguyen, 2013).

$$A = \begin{bmatrix} \frac{1}{\sqrt{s}} Q_1 \\ \dots \\ \frac{1}{\sqrt{s}} Q_s \end{bmatrix}$$

- Each $Q_s \in \mathbb{R}^{v \times d}$ is an independent random hashing (HD) matrix
- Set $v = \Theta(k\epsilon^{-1}\log^5(k/\delta))$ and $s = \Theta(\epsilon^{-1}\log^3(k/\delta))$
- w.h.p $1-\delta$, $A \in \mathbb{R}^{m \times d}$ with $m = \Theta\left(\frac{k\log^8(k/\delta)}{\epsilon^2}\right)$ is a subspace embedding w.r.t a k-dimensional space in $\mathbb{R}^d$
- Computational Cost of AX: $O\left(\frac{nnz(X)}{\epsilon}\log^3\left[\frac{k}{\delta}\right]\right)$

Sparse Subspace Embedding (SSE)

Random hashing is SSE with a Constant Probability (Nelson & Nguyen, 2013)

$$A = HD$$

where $D \in \mathbb{R}^{d\times d}$ and $H \in \mathbb{R}^{m\times d}$

- $m = \Omega(k^2/\epsilon^2)$ suffice for a subspace embedding with a probability $2/3$
- Computational Cost AX: $O(nnz(X))$

Randomized Dimensionality Reduction

- Johnson-Lindenstauss (JL) transforms
- Subspace embeddings
- Column (Row) sampling

Column sampling

- Column subset selection (feature selection)
- More interpretable
- Uniform sampling usually does not work (not a JL transform)
- Non-oblivious sampling (data-dependent sampling)
  - leverage-score sampling

Column sampling

- Column subset selection (feature selection)
- More interpretable
- Uniform sampling usually does not work (not a JL transform)
- Non-oblivious sampling (data-dependent sampling)
  - leverage-score sampling

Column sampling

- Column subset selection (feature selection)
- More interpretable
- Uniform sampling usually does not work (not a JL transform)
- Non-oblivious sampling (data-dependent sampling)
    - leverage-score sampling

Leverage-score sampling (Drineas et al., 2006)

Let $X \in \mathbb{R}^{d\times n}$ be a rank-k matrix

- $X = U\Sigma V^\top$: $U \in \mathbb{R}^{d\times k}$, $\Sigma \in \mathbb{R}^{k\times k}$
- Leverage scores $\|U_{i*}\|_2^2$, $i = 1, \ldots, d$
- Let $p_i = \frac{\|U_{i*}\|_2^2}{\sum_{i=1}^d \|U_{i*}\|_2^2}$, $i = 1, \ldots, d$
- Let $i_1, \ldots, i_m \in \{1, \ldots, d\}$ denote m indices selected by following p_i
- Let $A \in \mathbb{R}^{m\times d}$ be sampling-and-rescaling matrix:

$$A_{ij} = \begin{cases} \frac{1}{\sqrt{mp_j}} & \text{if } j = i_j \\ 0 & \text{otherwise} \end{cases}$$

- $AX \in \mathbb{R}^{m\times n}$ is a small sketch of X

Leverage-score sampling (Drineas et al., 2006)

Let $X \in \mathbb{R}^{d\times n}$ be a rank-k matrix

- $X = U\Sigma V^\top$: $U \in \mathbb{R}^{d\times k}$, $\Sigma \in \mathbb{R}^{k\times k}$
- Leverage scores $\|U_{i*}\|_2^2$, $i = 1, \ldots, d$
- Let $p_i = \frac{\|U_{i*}\|_2^2}{\sum_{i=1}^d \|U_{i*}\|_2^2}$, $i = 1, \ldots, d$
- Let $i_1, \ldots, i_m \in \{1, \ldots, d\}$ denote m indices selected by following p_i
- Let $A \in \mathbb{R}^{m\times d}$ be sampling-and-rescaling matrix:

$$A_{ij} = \begin{cases} \frac{1}{\sqrt{mp_j}} & \text{if } j = i_j \\ 0 & \text{otherwise} \end{cases}$$

- $AX \in \mathbb{R}^{m\times n}$ is a small sketch of X

Leverage-score sampling (Drineas et al., 2006)

Let $X \in \mathbb{R}^{d\times n}$ be a rank-k matrix

- $X = U\Sigma V^\top$: $U \in \mathbb{R}^{d\times k}$, $\Sigma \in \mathbb{R}^{k\times k}$
- Leverage scores $\|U_{i*}\|_2^2$, $i = 1, \ldots, d$
- Let $p_i = \frac{\|U_{i*}\|_2^2}{\sum_{i=1}^d \|U_{i*}\|_2^2}$, $i = 1, \ldots, d$
- Let $i_1, \ldots, i_m \in \{1, \ldots, d\}$ denote m indices selected by following p_i
- Let $A \in \mathbb{R}^{m\times d}$ be sampling-and-rescaling matrix:

$$A_{ij} = \begin{cases} \frac{1}{\sqrt{mp_j}} & \text{if } j = i_j \\ 0 & \text{otherwise} \end{cases}$$

- $AX \in \mathbb{R}^{m\times n}$ is a small sketch of X

Leverage-score sampling (Drineas et al., 2006)

Let $X \in \mathbb{R}^{d\times n}$ be a rank-k matrix

- $X = U\Sigma V^\top$: $U \in \mathbb{R}^{d\times k}$, $\Sigma \in \mathbb{R}^{k\times k}$
- Leverage scores $\|U_{i*}\|_2^2$, $i = 1, \ldots, d$
- Let $p_i = \frac{\|U_{i*}\|_2^2}{\sum_{i=1}^d \|U_{i*}\|_2^2}$, $i = 1, \ldots, d$
- Let $i_1, \ldots, i_m \in \{1, \ldots, d\}$ denote m indices selected by following p_i
- Let $A \in \mathbb{R}^{m\times d}$ be sampling-and-rescaling matrix:

$$A_{ij} = \begin{cases} \frac{1}{\sqrt{mp_j}} & \text{if } j = i_j \\ 0 & \text{otherwise} \end{cases}$$

- $AX \in \mathbb{R}^{m\times n}$ is a small sketch of X

Leverage-score sampling (Drineas et al., 2006)

Let $X \in \mathbb{R}^{d \times n}$ be a rank-k matrix

- $X = U\Sigma V^\top$: $U \in \mathbb{R}^{d \times k}$, $\Sigma \in \mathbb{R}^{k \times k}$
- Leverage scores $\|U_{i*}\|_2^2$, $i = 1, \ldots, d$
- Let $p_i = \frac{\|U_{i*}\|_2^2}{\sum_{i=1}^d \|U_{i*}\|_2^2}$, $i = 1, \ldots, d$
- Let $i_1, \ldots, i_m \in \{1, \ldots, d\}$ denote m indices selected by following p_i
- Let $A \in \mathbb{R}^{m \times d}$ be sampling-and-rescaling matrix:

$$A_{ij} = \begin{cases} \frac{1}{\sqrt{mp_j}} & \text{if } j = i_j \\ 0 & \text{otherwise} \end{cases}$$

- $AX \in \mathbb{R}^{m \times n}$ is a small sketch of X

Properties of Leverage-score sampling

When $m = \Theta\left(\frac{k}{\epsilon^2}\log\left[\frac{2k}{\delta}\right]\right)$, w.h.p $1-\delta$,

- $AU \in \mathbb{R}^{m\times k}$ is full column rank
- $\sigma_i^2(AU) \geq (1-\epsilon) \geq (1-\epsilon)^2$
- $\sigma_i^2(AU) \leq 1+\epsilon \leq (1+\epsilon)^2$
- Leverage-score sampling performs like a subspace embedding (only for U, the top singular vector matrix of X)
- Computational cost: compute top-k SVD of X, expensive
- Randomized algoritms to compute approximate leverage scores

Properties of Leverage-score sampling

When $m = \Theta\left(\frac{k}{\epsilon^2}\log\left[\frac{2k}{\delta}\right]\right)$, w.h.p $1-\delta$,

- $AU \in \mathbb{R}^{m\times k}$ is full column rank
- $\sigma_i^2(AU) \geq (1-\epsilon) \geq (1-\epsilon)^2$
- $\sigma_i^2(AU) \leq 1+\epsilon \leq (1+\epsilon)^2$
- Leverage-score sampling performs like a subspace embedding (only for U, the top singular vector matrix of X)
- Computational cost: compute top-k SVD of X, expensive
- Randomized algoritms to compute approximate leverage scores

When uniform sampling makes sense?

Coherence measure

$$\mu_k = \frac{d}{k} \max_{1 \le i \le d} \|U_{i*}\|_2^2$$

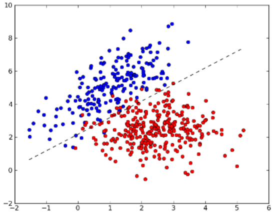
- Valid when the coherence measure is small (some real data mining datasets have small coherence measures)
- The Nyström method usually uses uniform sampling (Gittens, 2011)

Outline

Classification

Classification problems:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i\mathbf{w}^\top\mathbf{x}_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$



- $y_i \in \{+1, -1\}$: label
- Loss function $\ell(z)$: $z = y\mathbf{w}^\top\mathbf{x}$
    1. SVMs: (squared) hinge loss $\ell(z) = \max(0, 1 - z)^p$, where $p = 1, 2$
    2. Logistic Regression: $\ell(z) = \log(1 + \exp(-z))$

Randomized Classification

For large-scale high-dimensional problems, the computational cost of optimization is $O((nd + d\kappa)\log(1/\epsilon))$.

Use random reduction $A \in \mathbb{R}^{d\times m}$ $(m \ll d)$, we reduce $X \in \mathbb{R}^{n\times d}$ to $\widehat{X} = XA \in \mathbb{R}^{n\times m}$. Then solve

$$\min_{\mathbf{u}\in\mathbb{R}^m} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i \mathbf{u}^\top \widehat{\mathbf{x}}_i) + \frac{\lambda}{2}\|\mathbf{u}\|_2^2$$

- JL transforms
- Sparse subspace embeddings

Randomized Classification

Two questions:

- Is there any performance guarantee?
  - margin is preserved: if data is linearly separable (Balcan et al., 2006) as long as $m \geq \frac{12}{\epsilon^2}\log(\frac{6m}{\delta})$
  - generalization performance is preserved: if the data matrix if of low rank and $m = \Omega(\frac{kploy(\log(k/\delta\epsilon))}{\epsilon^2})$ (Paul et al., 2013)
- How to recover an accurate model in the original high-dimensional space?

  Dual Recovery (Zhang et al., 2014) and Dual Sparse Recovery (Yang et al., 2015)

Randomized Classification

Two questions:

- Is there any performance guarantee?
    - margin is preserved: if data is linearly separable (Balcan et al., 2006) as long as $m \geq \frac{12}{\epsilon^2}\log(\frac{6m}{\delta})$
    - generalization performance is preserved: if the data matrix if of low rank and $m = \Omega(\frac{kploy(\log(k/\delta\epsilon))}{\epsilon^2})$ (Paul et al., 2013)
- How to recover an accurate model in the original high-dimensional space?

  Dual Recovery (Zhang et al., 2014) and Dual Sparse Recovery (Yang et al., 2015)

Randomized Classification

Two questions:

- Is there any performance guarantee?
  - margin is preserved: if data is linearly separable (Balcan et al., 2006) as long as $m \geq \frac{12}{\epsilon^2}\log(\frac{6m}{\delta})$
  - generalization performance is preserved: if the data matrix if of low rank and $m = \Omega(\frac{kploy(\log(k/\delta\epsilon))}{\epsilon^2})$ (Paul et al., 2013)
- How to recover an accurate model in the original high-dimensional space?

  Dual Recovery (Zhang et al., 2014) and Dual Sparse Recovery (Yang et al., 2015)

Randomized Classification

Two questions:

- Is there any performance guarantee?
  - margin is preserved: if data is linearly separable (Balcan et al., 2006) as long as $m \geq \frac{12}{\epsilon^2}\log(\frac{6m}{\delta})$
  - generalization performance is preserved: if the data matrix if of low rank and $m = \Omega(\frac{kploy(\log(k/\delta\epsilon))}{\epsilon^2})$ (Paul et al., 2013)
- How to recover an accurate model in the original high-dimensional space?

  Dual Recovery (Zhang et al., 2014) and Dual Sparse Recovery (Yang et al., 2015)

Randomized Classification

Two questions:

- Is there any performance guarantee?
  - margin is preserved: if data is linearly separable (Balcan et al., 2006) as long as $m \geq \frac{12}{\epsilon^2}\log(\frac{6m}{\delta})$
  - generalization performance is preserved: if the data matrix if of low rank and $m = \Omega(\frac{kploy(\log(k/\delta\epsilon))}{\epsilon^2})$ (Paul et al., 2013)
- How to recover an accurate model in the original high-dimensional space?

  Dual Recovery (Zhang et al., 2014) and Dual Sparse Recovery (Yang et al., 2015)

The Dual probelm

Using Fenchel conjugate

$$\ell_i^*(\alpha_i) = \max_{\alpha_i} \alpha_i z - \ell(z, y_i)$$

Primal:

$$\mathbf{w}_* = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^n \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

Dual:

$$\alpha_* = \arg\max_{\alpha\in\mathbb{R}^n} -\frac{1}{n}\sum_{i=1}^n \ell_i^*(\alpha_i) - \frac{1}{2\lambda n^2}\alpha^\top X X^\top \alpha$$

From dual to primal:

$$\mathbf{w}_* = -\frac{1}{\lambda n} X^\top \alpha_*$$

Dual Recovery for Randomized Reduction

From dual formulation: $\mathbf{w}_*$ lies in the row space of the data matrix $X \in \mathbb{R}^{n\times d}$

- Dual Recovery: $\widetilde{\mathbf{w}}_* = -\frac{1}{\lambda n} X^\top \widehat{\alpha}_*$, where

$$\widehat{\alpha}_* = \arg\max_{\alpha \in \mathbb{R}^n} -\frac{1}{n}\sum_{i=1}^n \ell_i^*(\alpha_i) - \frac{1}{2\lambda n^2}\alpha^\top \widehat{X}\widehat{X}^\top \alpha$$

  and $\widehat{X} = XA \in \mathbb{R}^{n\times m}$
- Subspace Embedding A with $m = \Theta(r\log(r/\delta)\epsilon^{-2})$
- Guarantee: under low-rank assumption of the data matrix X (e.g., $rank(X) = r$), with a high probability $1-\delta$,

$$\|\widetilde{\mathbf{w}}_* - \mathbf{w}_*\|_2 \le \frac{\epsilon}{1-\epsilon}\|\mathbf{w}_*\|_2$$

Dual Recovery for Randomized Reduction

From dual formulation: $\mathbf{w}_*$ lies in the row space of the data matrix $X \in \mathbb{R}^{n\times d}$

- Dual Recovery: $\widetilde{\mathbf{w}}_* = -\frac{1}{\lambda n} X^\top \widehat{\alpha}_*$, where

$$\widehat{\alpha}_* = \arg\max_{\alpha\in\mathbb{R}^n} -\frac{1}{n}\sum_{i=1}^n \ell_i^*(\alpha_i) - \frac{1}{2\lambda n^2}\alpha^\top \widehat{X}\widehat{X}^\top \alpha$$

  and $\widehat{X} = XA \in \mathbb{R}^{n\times m}$
- Subspace Embedding A with $m = \Theta(r\log(r/\delta)\epsilon^{-2})$
- Guarantee: under low-rank assumption of the data matrix X (e.g., $rank(X) = r$), with a high probability $1-\delta$,

$$\|\widetilde{\mathbf{w}}_* - \mathbf{w}_*\|_2 \le \frac{\epsilon}{1-\epsilon}\|\mathbf{w}_*\|_2$$

Dual Recovery for Randomized Reduction

From dual formulation: $\mathbf{w}_*$ lies in the row space of the data matrix $X \in \mathbb{R}^{n\times d}$

- Dual Recovery: $\widetilde{\mathbf{w}}_* = -\frac{1}{\lambda n} X^\top \widehat{\alpha}_*$, where

$$\widehat{\alpha}_* = \arg\max_{\alpha\in\mathbb{R}^n} -\frac{1}{n}\sum_{i=1}^n \ell_i^*(\alpha_i) - \frac{1}{2\lambda n^2}\alpha^\top \widehat{X}\widehat{X}^\top \alpha$$

  and $\widehat{X} = XA \in \mathbb{R}^{n\times m}$
- Subspace Embedding A with $m = \Theta(r\log(r/\delta)\epsilon^{-2})$
- Guarantee: under low-rank assumption of the data matrix X (e.g., $rank(X) = r$), with a high probability $1-\delta$,

$$\|\widetilde{\mathbf{w}}_* - \mathbf{w}_*\|_2 \le \frac{\epsilon}{1-\epsilon}\|\mathbf{w}_*\|_2$$

Dual Sparse Recovery for Randomized Reduction

Assume the optimal dual solution α_* is sparse (i.e., the number of support vectors is small)

- Dual Sparse Recovery: $\widetilde{\mathbf{w}}_* = -\frac{1}{\lambda n} X^\top \widehat{\alpha}_*$, where

$$\widehat{\alpha}_* = \arg\max_{\alpha \in \mathbb{R}^n} -\frac{1}{n}\sum_{i=1}^n \ell_i^*(\alpha_i) - \frac{1}{2\lambda n^2}\alpha^\top \widehat{X}\widehat{X}^\top \alpha - \frac{\tau}{n}\|\alpha\|_1$$

  where $\widehat{X} = XA \in \mathbb{R}^{n\times m}$
- JL transform A with $m = \Theta(s\log(n/\delta)\epsilon^{-2})$
- Guarantee: if α_* is s-sparse, with a high probability $1-\delta$,

$$\|\widetilde{\mathbf{w}}_* - \mathbf{w}_*\|_2 \leq \epsilon\|\mathbf{w}_*\|_2$$

Dual Sparse Recovery for Randomized Reduction

Assume the optimal dual solution α_* is sparse (i.e., the number of support vectors is small)

- Dual Sparse Recovery: $\widetilde{\mathbf{w}}_* = -\frac{1}{\lambda n} X^\top \widehat{\alpha}_*$, where

$$\widehat{\alpha}_* = \arg\max_{\alpha \in \mathbb{R}^n} -\frac{1}{n}\sum_{i=1}^n \ell_i^*(\alpha_i) - \frac{1}{2\lambda n^2}\alpha^\top \widehat{X}\widehat{X}^\top \alpha - \frac{\tau}{n}\|\alpha\|_1$$

  where $\widehat{X} = XA \in \mathbb{R}^{n\times m}$

- JL transform A with $m = \Theta(s\log(n/\delta)\epsilon^{-2})$
- Guarantee: if α_* is s-sparse, with a high probability $1-\delta$,

$$\|\widetilde{\mathbf{w}}_* - \mathbf{w}_*\|_2 \leq \epsilon \|\mathbf{w}_*\|_2$$

Dual Sparse Recovery for Randomized Reduction

Assume the optimal dual solution α_* is sparse (i.e., the number of support vectors is small)

- Dual Sparse Recovery: $\widetilde{\mathbf{w}}_* = -\frac{1}{\lambda n} X^\top \widehat{\alpha}_*$, where

$$\widehat{\alpha}_* = \arg\max_{\alpha \in \mathbb{R}^n} -\frac{1}{n}\sum_{i=1}^n \ell_i^*(\alpha_i) - \frac{1}{2\lambda n^2}\alpha^\top \widehat{X}\widehat{X}^\top \alpha - \frac{\tau}{n}\|\alpha\|_1$$

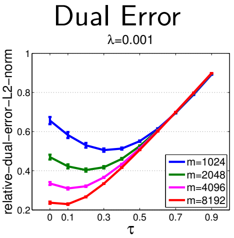
  where $\widehat{X} = XA \in \mathbb{R}^{n\times m}$

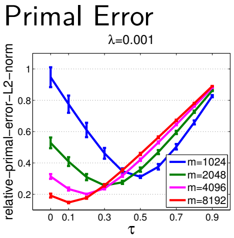
- JL transform A with $m = \Theta(s\log(n/\delta)\epsilon^{-2})$
- Guarantee: if α_* is s-sparse, with a high probability $1-\delta$,

$$\|\widetilde{\mathbf{w}}_* - \mathbf{w}_*\|_2 \leq \epsilon\|\mathbf{w}_*\|_2$$

Dual Sparse Recovery

RCV1 text data, $n = 677,399$, and $d = 47,236$





Outline

Least-squares regression

Let $X \in \mathbb{R}^{n\times d}$ with $d \ll n$ and $b \in \mathbb{R}^n$. The least-squares regression problem is to find $\mathbf{w}_*$ such that

$$\mathbf{w}_* = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \|X\mathbf{w} - b\|_2$$

- Computational Cost: $O(nd^2)$
- Goal of RA: $o(nd^2)$

Randomized Least-squares regression

Let $A \in \mathbb{R}^{m\times n}$ be a random reduction matrix. Solve

$$\widehat{\mathbf{w}}_* = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \|A(X\mathbf{w} - b)\|_2 = \|AX\mathbf{w} - Ab\|_2$$

- Computational Cost: $O(md^2)$ + reduction time

Randomized Least-squares regression

Theoretical Guarantees (Sarlós, 2006; Drineas et al., 2011; Nelson & Nguyen, 2012):

$$\|X\widehat{\mathbf{w}}_* - b\|_2 \leq (1+\epsilon)\|X\mathbf{w}_* - b\|_2$$

Total Time $O(nnz(X) + d^3 \log(d/\epsilon)\epsilon^{-2})$

Outline

K-means Clustering

Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ be a set of data points.

K-means clustering aims to solve

$$\min_{C_1,\ldots,C_k} \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in C_j} \|\mathbf{x}_i - \mu_j\|_2^2$$

Computational Cost: $O(ndkt)$, where t is number of iterations.

Randomized Algorithms for K-means Clustering

Let $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top \in \mathbb{R}^{n\times d}$ be the data matrix.
High-dimensional data: Random Sketch: $\widehat{X} = XA \in \mathbb{R}^{n\times m}$, $\ell \ll d$

Approximate K-means:

$$\min_{C_1,\ldots,C_k} \sum_{j=1}^{k} \sum_{\widehat{\mathbf{x}}_i \in C_j} \|\widehat{\mathbf{x}}_i - \widehat{\mu}_j\|_2^2$$

Randomized Algorithms for K-means Clustering

Let $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top \in \mathbb{R}^{n\times d}$ be the data matrix.
High-dimensional data: Random Sketch: $\widehat{X} = XA \in \mathbb{R}^{n\times m}$, $\ell \ll d$

Approximate K-means:

$$\min_{C_1,\ldots,C_k} \sum_{j=1}^{k} \sum_{\widehat{\mathbf{x}}_i \in C_j} \|\widehat{\mathbf{x}}_i - \widehat{\mu}_j\|_2^2$$

Randomized Algorithms for K-means Clustering

For random sketch: JL transforms, sparse subspace embedding all work

- JL transform: $m = O(\frac{k\log(k/(\epsilon\delta))}{\epsilon^2})$
- Sparse subspace embedding: $m = O(\frac{k^2}{\epsilon^2\delta})$
- ϵ relates to the approximation accuracy
- Analysis of approximation error for K-means can be formulates as Constrained Low-rank Approximation (Cohen et al., 2015)

$$\min_{Q^\top Q=I} \|X - QQ^\top X\|_F^2$$

where Q is orthonormal.

Outline

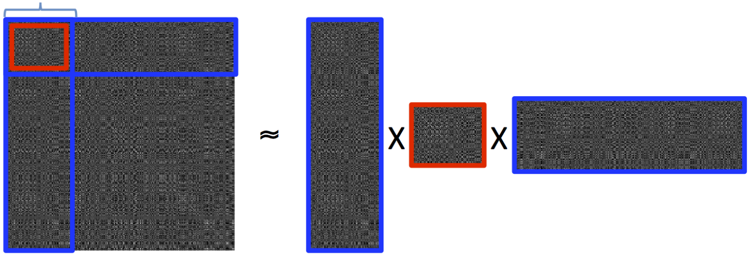
Kernel methods

- Kernel function: $\kappa(\cdot, \cdot)$
- a set of examples $\mathbf{x}_1, \ldots, \mathbf{x}_n$
- Kernel matrix: $K \in \mathbb{R}^{n \times n}$ with $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$
- K is a PSD matrix
- Computational and memory costs: $\Omega(n^2)$
- Approximation methos
    - The Nyström method
    - Random Fourier features

Kernel methods

- Kernel function: $\kappa(\cdot, \cdot)$
- a set of examples $\mathbf{x}_1, \ldots, \mathbf{x}_n$
- Kernel matrix: $K \in \mathbb{R}^{n\times n}$ with $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$
- K is a PSD matrix
- Computational and memory costs: $\Omega(n^2)$
- Approximation methos
    - The Nyström method
    - Random Fourier features

The Nyström method



Let $A \in \mathbb{R}^{n\times\ell}$ be uniform sampling matrix.

$$B = KA \in \mathbb{R}^{n\times\ell}$$

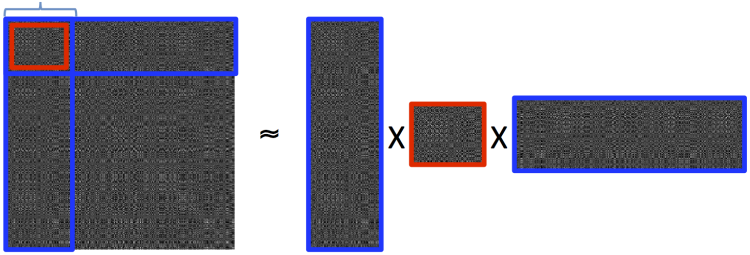
$$C = A^\top B = A^\top KA$$

The Nyström approximation (Drineas & Mahoney, 2005)

$$\widehat{K} = BC^\dagger B^\top$$

Computational Cost: $O(\ell^3 + n\ell^2)$

The Nyström method



Let $A \in \mathbb{R}^{n\times\ell}$ be uniform sampling matrix.

$$B = KA \in \mathbb{R}^{n\times\ell}$$

$$C = A^\top B = A^\top KA$$

The Nyström approximation (Drineas & Mahoney, 2005)

$$\widehat{K} = BC^\dagger B^\top$$

Computational Cost: $O(\ell^3 + n\ell^2)$

The Nyström based kernel machine

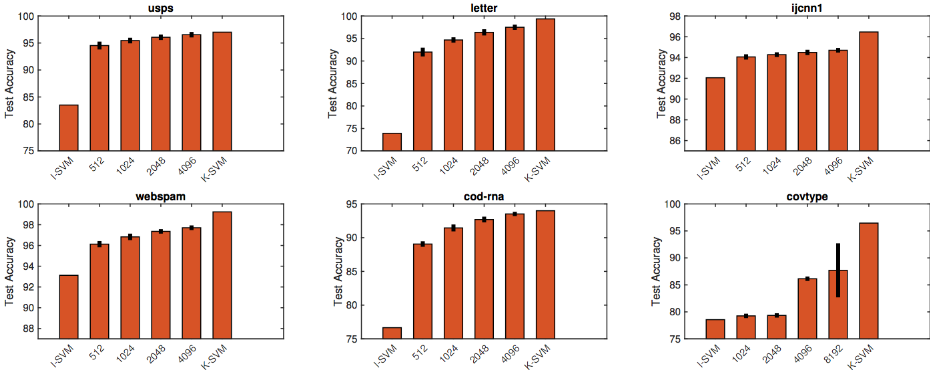
The dual problem:

$$\arg\max_{\alpha\in\mathbb{R}^n} -\frac{1}{n}\sum_{i=1}^{n}\ell_i^*(\alpha_i) - \frac{1}{2\lambda n^2}\alpha^\top BC^\dagger B^\top \alpha$$

Solve it like solving a linear method: $\widehat{X} = BC^{-1/2} \in \mathbb{R}^{n\times\ell}$

$$\arg\max_{\alpha\in\mathbb{R}^n} -\frac{1}{n}\sum_{i=1}^{n}\ell_i^*(\alpha_i) - \frac{1}{2\lambda n^2}\alpha^\top \widehat{X}\widehat{X}^\top \alpha$$

The Nyström based kernel machine



Random Fourier Features (RFF)

Bochner's theorem
A shift-invariant kernel $\kappa(\mathbf{x}, \mathbf{y}) = \kappa(\mathbf{x} - \mathbf{y})$ is a valid kernel if only if $\kappa(\delta)$ is the Fourier transform of a non-negative measure, i.e.,

$$\kappa(\mathbf{x} - \mathbf{y}) = \int p(\omega) e^{-j\omega^\top (\mathbf{x}-\mathbf{y})} d\omega$$

RFF (Rahimi & Recht, 2008): generate a set of $\omega_1, \ldots, \omega_m \in \mathbb{R}^d$ following $p(\omega)$. For an example $\mathbf{x} \in \mathbb{R}^d$, construct

$$\widehat{\mathbf{x}} = (cos(\omega_1^\top \mathbf{x}), sin(\omega_1^\top \mathbf{x}), \ldots, cos(\omega_m^\top \mathbf{x}), sin(\omega_m^\top \mathbf{x}))^\top \in \mathbb{R}^{2m}$$

RBF kernel $\exp(-\frac{\|\mathbf{x}-\mathbf{y}\|_2^2}{2\gamma^2})$: $p(\omega) = \mathcal{N}(0, \gamma^2)$

Random Fourier Features (RFF)

Bochner's theorem
A shift-invariant kernel $\kappa(\mathbf{x}, \mathbf{y}) = \kappa(\mathbf{x} - \mathbf{y})$ is a valid kernel if only if $\kappa(\delta)$ is the Fourier transform of a non-negative measure, i.e.,

$$\kappa(\mathbf{x} - \mathbf{y}) = \int p(\omega) e^{-j\omega^\top (\mathbf{x}-\mathbf{y})} d\omega$$

RFF (Rahimi & Recht, 2008): generate a set of $\omega_1, \ldots, \omega_m \in \mathbb{R}^d$ following $p(\omega)$. For an example $\mathbf{x} \in \mathbb{R}^d$, construct

$$\widehat{\mathbf{x}} = (cos(\omega_1^\top \mathbf{x}), sin(\omega_1^\top \mathbf{x}), \ldots, cos(\omega_m^\top \mathbf{x}), sin(\omega_m^\top \mathbf{x}))^\top \in \mathbb{R}^{2m}$$

RBF kernel $\exp(-\frac{\|\mathbf{x}-\mathbf{y}\|_2^2}{2\gamma^2})$: $p(\omega) = \mathcal{N}(0, \gamma^2)$

Random Fourier Features (RFF)

Bochner's theorem
A shift-invariant kernel $\kappa(\mathbf{x}, \mathbf{y}) = \kappa(\mathbf{x} - \mathbf{y})$ is a valid kernel if only if $\kappa(\delta)$ is the Fourier transform of a non-negative measure, i.e.,

$$\kappa(\mathbf{x} - \mathbf{y}) = \int p(\omega) e^{-j\omega^\top (\mathbf{x}-\mathbf{y})} d\omega$$

RFF (Rahimi & Recht, 2008): generate a set of $\omega_1, \ldots, \omega_m \in \mathbb{R}^d$ following $p(\omega)$. For an example $\mathbf{x} \in \mathbb{R}^d$, construct

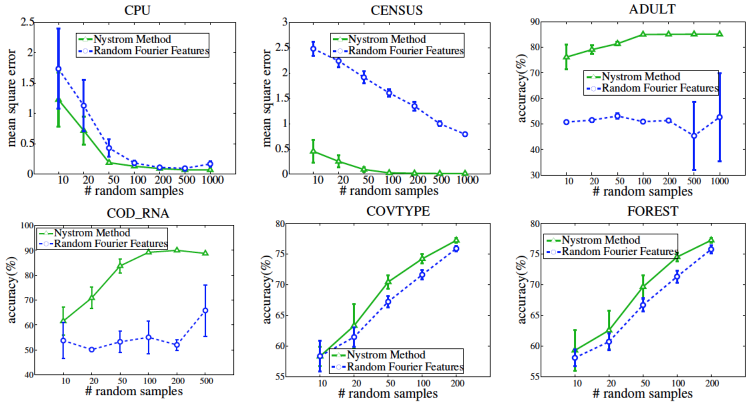
$$\widehat{\mathbf{x}} = (cos(\omega_1^\top \mathbf{x}), sin(\omega_1^\top \mathbf{x}), \ldots, cos(\omega_m^\top \mathbf{x}), sin(\omega_m^\top \mathbf{x}))^\top \in \mathbb{R}^{2m}$$

RBF kernel $\exp(-\frac{\|\mathbf{x}-\mathbf{y}\|_2^2}{2\gamma^2})$: $p(\omega) = \mathcal{N}(0, \gamma^2)$

The Nyström method vs RFF (Yang et al., 2012)

- functional approximation framework
- The Nyström method: data-dependent bases
- RFF: data independent bases
- In certain cases (e.g., large eigen-gap, skewed eigen-value distribution): the generalization performance of the Nyström method is better than RFF

The Nyström method vs RFF



Outline

Randomized low-rank matrix approximation

Let $X \in \mathbb{R}^{n\times d}$. The goal is to obtain

$$\widehat{U}\widehat{\Sigma}\widehat{V}^\top \approx X$$

where $\widehat{U} \in \mathbb{R}^{n\times k}$, $\widehat{V} \in \mathbb{R}^{d\times k}$ have orthonormal columns, $\widehat{\Sigma} \in \mathbb{R}^{k\times k}$ is a diagonal matrix with nonegative entries

- k is target rank
- The best rank-k approximation $X_k = U_k\Sigma_k V_k^\top$
- Approximation error

$$\|\widehat{U}\widehat{\Sigma}\widehat{V}^\top - X\|_\xi \leq (1+\epsilon)\|U_k\Sigma_k V_k^\top - X\|_\xi$$

where $\xi = F$ or $\xi = 2$

Why low-rank approximation?

Applications in Data mining and Machine learning

- PCA
- Spectral clustering
- $\cdots$

Why randomized algorithms?

Deterministic Algorithms

- Truncated SVD $O(nd\min(n,d))$
- Rank-Revealing QR factorization $O(ndk)$
- Krylov subspace method (e.g. Lanczos algorithm): $O(kT_{mult} + (n+d)k^2)$, where T_{mult} denotes the cost of matrix-vector product.

Randomized Algorithms

- Speed can be faster (e.g., $O(nd\log(k))$)
- Output more robust (e.g. Lanczos requires sophisticated modifications)
- Can be pass efficient
- Can exploit parallel algorithms

Why randomized algorithms?

Deterministic Algorithms

- Truncated SVD $O(nd\min(n,d))$
- Rank-Revealing QR factorization $O(ndk)$
- Krylov subspace method (e.g. Lanczos algorithm): $O(kT_{mult} + (n+d)k^2)$, where T_{mult} denotes the cost of matrix-vector product.

Randomized Algorithms

- Speed can be faster (e.g., $O(nd\log(k))$)
- Output more robust (e.g. Lanczos requires sophisticated modifications)
- Can be pass efficient
- Can exploit parallel algorithms

Randomized algorithms for low-rank matrix approximation

The Basic Randomized Algorithms for Approximating $X \in \mathbb{R}^{n\times d}$ (Halko et al., 2011)

1. Obtain a small sketch by $Y = XA \in \mathbb{R}^{n\times m}$
2. Compute $Q \in \mathbb{R}^{n\times m}$ that contains the orthonormal basis of $col(Y)$
3. Compute SVD of $Q^T X = U\Sigma V^\top$
4. Approximation $X \approx \widetilde{U}\Sigma V^\top$, where $\widetilde{U} = QU$

   Explanation: If $col(XA)$ captures the top-k column space of X well, i.e.,

$$\|X - QQ^\top X\| \leq \varepsilon$$

then

$$\|X - \widetilde{U}\Sigma V^\top\| \leq \varepsilon$$

Randomized algorithms for low-rank matrix approximation

Three questions:

1. What is the value of m?
   - $m = k + p$, p is the oversampling parameter. In practice $p = 5$ or 10 gives superb results
2. What is the computational cost?
   - Subsampled Randomized Hadmard Transform: can be as fast as $O(nd\log(k) + k^2(n + d))$
3. What is the quality?
   - Theoretical Guarantee:
   - Practically, very accurate

Randomized algorithms for low-rank matrix approximation

Three questions:

1. What is the value of m?
   - $m = k + p$, p is the oversampling parameter. In practice $p = 5$ or 10 gives superb results
2. What is the computational cost?
   - Subsampled Randomized Hadmard Transform: can be as fast as $O(nd\log(k) + k^2(n+d))$
3. What is the quality?
   - Theoretical Guarantee:
   - Practically, very accurate

Randomized algorithms for low-rank matrix approximation

Three questions:

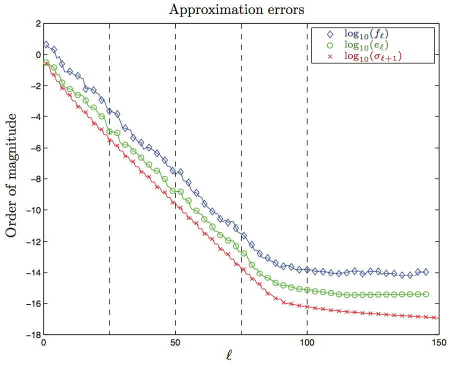
1. What is the value of m?
   - $m = k + p$, p is the oversampling parameter. In practice $p = 5$ or 10 gives superb results
2. What is the computational cost?
   - Subsampled Randomized Hadmard Transform: can be as fast as $O(nd\log(k) + k^2(n+d))$
3. What is the quality?
   - Theoretical Guarantee:
   - Practically, very accurate

Randomized algorithms for low-rank matrix approximation

Three questions:

1. What is the value of m?
   - $m = k + p$, p is the oversampling parameter. In practice $p = 5$ or 10 gives superb results
2. What is the computational cost?
   - Subsampled Randomized Hadmard Transform: can be as fast as $O(nd\log(k) + k^2(n+d))$
3. What is the quality?
   - Theoretical Guarantee:
   - Practically, very accurate

Randomized algorithms for low-rank matrix approximation



Randomized algorithms for low-rank matrix approximation

Other things

- Use power iteration to reduce the error: use $(XX^\top)^q X$

- Can use sparse JL transform/subspace embedding matrices (Frobenius norm guarantee only)

Outline

How to address big data challenge?

- Optimization perspective: improve convergence rates, exploring properties of functions
  - stochastic optimization (e.g., SDCA, SVRG, SAGA)
  - distributed optimization (e.g., DisDCA)
- Randomization perspective: reduce data size, exploring properties of data
  - randomized feature reduction (e.g., reduce the number of features)
  - randomized instance reduction (e.g., reduce the number of instances)

How can we address big data challenge?

- Optimization perspective: improve convergence rates, exploring properties of functions
    - Pro: can obtain the optimal solution
    - Con: high computational/communication costs
- Randomization perspective: reduce data size, exploring properties of data
    - Pro: fast
    - Con: still exists recovery error

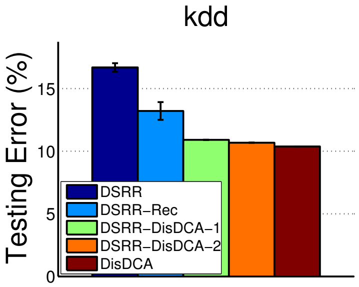
Can we combine the benefits of two techniques?

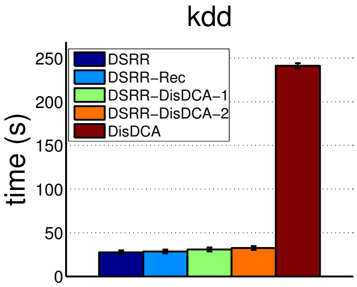
Combine Randomization and Optimization (Yang et al., 2015)

- Use randomization (Dual Spare Recovery) to obtain a good initial solution
- Initialize distributed optimization (DisDCA) to reduce cost of computation/communication
- Observe 1 or 2 epochs of computations (1 or 2 communications) suffice to obtain the same performance of pure optimization

Big Data Experiments

KDDcup Data: $n = 8,407,752$, $d = 29,890,095$, 10 machines, $m = 1024$





THANK YOU! QUESTIONS?

References I

Achlioptas, Dimitris. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671 – 687, 2003.

Balcan, Maria-Florina, Blum, Avrim, and Vempala, Santosh. Kernels as features: on kernels, margins, and low-dimensional mappings. *Machine Learning*, 65(1):79–94, 2006.

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. Distributed optimization and statistical learning via the alternating direction methods of multiplies. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

References II

Cohen, Michael B., Elder, Sam, Musco, Cameron, Musco, Christopher, and Persu, Madalina. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (STOC)*, pp. 163–172, 2015.

Dasgupta, Anirban, Kumar, Ravi, and Sarlos, Tamás. A sparse johnson: Lindenstrauss transform. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pp. 341–350, 2010.

Dasgupta, Sanjoy and Gupta, Anupam. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms*, 22(1): 60–65, 2003.

Defazio, Aaron, Bach, Francis, and Lacoste-Julien, Simon. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *NIPS*, 2014.

References III

Drineas, Petros and Mahoney, Michael W. On the nystrom method for approximating a gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2005, 2005.

Drineas, Petros, Mahoney, Michael W., and Muthukrishnan, S. Sampling algorithms for l2 regression and applications. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 1127–1136, 2006.

Drineas, Petros, Mahoney, Michael W., Muthukrishnan, S., and Sarlós, Tamàs. Faster least squares approximation. *Numerische Mathematik*, 117(2):219–249, February 2011.

Gittens, Alex. The spectral norm error of the naive nystrom extension. *CoRR*, 2011.

Golub, Gene H. and Ye, Qiang. Inexact preconditioned conjugate gradient method with inner-outer iteration. *SIAM J. Sci. Comput*, 21:1305–1320, 1997.

References IV

Halko, Nathan, Martinsson, Per Gunnar., and Tropp, Joel A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, May 2011.

Hsieh, Cho-Jui, Chang, Kai-Wei, Lin, Chih-Jen, Keerthi, S. Sathiya, and Sundararajan, S. A dual coordinate descent method for large-scale linear svm. In *ICML*, pp. 408–415, 2008.

Johnson, Rie and Zhang, Tong. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, pp. 315–323, 2013.

Johnson, William and Lindenstrauss, Joram. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*, volume 26, pp. 189–206. 1984.

Kane, Daniel M. and Nelson, Jelani. Sparser johnson-lindenstrauss transforms. *Journal of the ACM*, 61:4:1–4:23, 2014.

References V

Lee, Jason, Ma, Tengyu, and Lin, Qihang. Distributed stochastic variance reduced gradient methods. Technical report, UC Berkeley, 2015.

Lin, Qihang, Lu, Zhaosong, and Xiao, Lin. An accelerated proximal coordinate gradient method and its application to regularized empirical risk minimization. In *NIPS*, 2014.

Ma, Chenxin, Smith, Virginia, Jaggi, Martin, Jordan, Michael I., Richtárik, Peter, and Takác, Martin. Adding vs. averaging in distributed primal-dual optimization. In *ICML*, 2015.

Nelson, Jelani and Nguyen, Huy L. OSNAP: faster numerical linear algebra algorithms via sparser subspace embeddings. *CoRR*, abs/1211.1002, 2012.

References VI

Nelson, Jelani and Nguyen, Huy L. OSNAP: faster numerical linear algebra algorithms via sparser subspace embeddings. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 117–126, 2013.

Nemirovski, A. and Yudin, D. On cezari's convergence of the steepest descent method for approximating saddle point of convex-concave functons. *Soviet Math Dkl.*, 19:341–362, 1978.

Nesterov, Yurii. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22:341–362, 2012.

Ozdaglar, Asu. Distributed multiagent optimization linear convergence rate of admm. Technical report, MIT, 2015.

References VII

Paul, Saurabh, Boutsidis, Christos, Magdon-Ismail, Malik, and Drineas, Petros. Random projections for support vector machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 498–506, 2013.

Rahimi, Ali and Recht, Benjamin. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20*, pp. 1177–1184, 2008.

Recht, Benjamin. A simpler approach to matrix completion. *Journal Machine Learning Research (JMLR)*, pp. 3413–3430, 2011.

Roux, Nicolas Le, Schmidt, Mark, and Bach, Francis. A stochastic gradient method with an exponential convergence rate for strongly-convex optimization with finite training sets. *CoRR*, 2012.

References VIII

Sarlós, Tamás. Improved approximation algorithms for large matrices via random projections. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 143–152, 2006.

Shalev-Shwartz, Shai and Zhang, Tong. Stochastic dual coordinate ascent methods for regularized loss. *Journal of Machine Learning Research*, 14: 567–599, 2013.

Shamir, Ohad, Srebro, Nathan, and Zhang, Tong. Communication-efficient distributed optimiztion using an approximate newton-type method. In *ICML*, 2014.

Tropp, Joel A. Improved analysis of the subsampled randomized hadamard transform. *Advances in Adaptive Data Analysis*, 3(1-2):115–126, 2011.

Tropp, Joel A. User-friendly tail bounds for sums of random matrices. *Found. Comput. Math.*, 12(4):389–434, August 2012. ISSN 1615-3375.

References IX

Xiao, L. and Zhang, T. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4): 2057–2075, 2014.

Yang, Tianbao. Trading computation for communication: Distributed stochastic dual coordinate ascent. *NIPS'13*, pp. –, 2013.

Yang, Tianbao, Li, Yu-Feng, Mahdavi, Mehrdad, Jin, Rong, and Zhou, Zhi-Hua. Nyström method vs random fourier features: A theoretical and empirical comparison". In *Advances in Neural Information Processing Systems (NIPS)*, pp. 485–493, 2012.

Yang, Tianbao, Zhang, Lijun, Jin, Rong, and Zhu, Shenghuo. Theory of dual-sparse regularized randomized reduction. In *Proceedings of the 32nd International Conference on Machine Learning, (ICML)*, pp. 305–314, 2015.

References X

Zhang, Lijun, Mahdavi, Mehrdad, and Jin, Rong. Linear convergence with condition number independent access of full gradients. In *NIPS*, pp. 980–988. 2013.

Zhang, Lijun, Mahdavi, Mehrdad, Jin, Rong, Yang, Tianbao, and Zhu, Shenghuo. Random projections for classification: A recovery approach. *IEEE Transactions on Information Theory (IEEE TIT)*, 60(11): 7300–7316, 2014.

Zhang, Yuchen and Xiao, Lin. Communication-efficient distributed optimization of self-concordant empirical loss. In *ICML*, 2015.

Examples of Convex functions

- $ax + b$, $A\mathbf{x} + b$
- x^2, $\|\mathbf{x}\|_2^2$
- $\exp(ax)$, $\exp(\mathbf{w}^\top \mathbf{x})$
- $\log(1 + \exp(ax))$, $\log(1 + \exp(\mathbf{w}^\top \mathbf{x}))$
- $x \log(x)$, $\sum_i x_i \log(x_i)$
- $\|\mathbf{x}\|_p, p \geq 1$, $\|\mathbf{x}\|_p^2$
- $\max_i(x_i)$

Operations that preserve convexity

- Nonnegative scale: $a \cdot f(\mathbf{x})$ where $a \geq 0$
- Sum: $f(\mathbf{x}) + g(\mathbf{x})$
- Composition with affine function $f(A\mathbf{x} + b)$
- Point-wise maximum: $\max_i f_i(\mathbf{x})$

Examples:

- Least-squares regression: $\|A\mathbf{x} - b\|_2$
- SVM: $\frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i \mathbf{w}^\top \mathbf{x}_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$

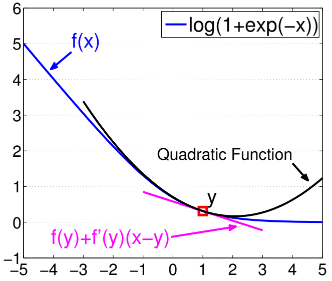
Smooth Convex function

- smooth: e.g. logistic loss $f(x) = \log(1 + \exp(-x))$

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2$$

where $L > 0$

Second Order Derivative is upper bounded $\|\nabla^2 f(x)\|_2 \leq L$



Smooth Convex function

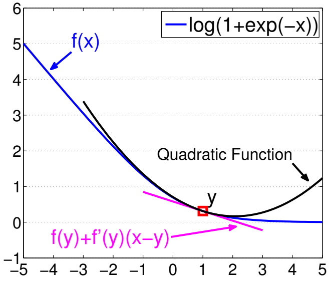
- smooth: e.g. logistic loss $f(x) = \log(1 + \exp(-x))$

smoothness constant

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2$$

where $L > 0$

Second Order Derivative is upper bounded $\|\nabla^2 f(x)\|_2 \leq L$

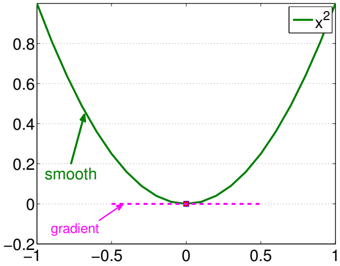


Strongly Convex function

- strongly convex: e.g. Euclidean norm $f(x) = \frac{1}{2}\|x\|_2^2$

$\|\nabla f(x) - \nabla f(y)\|_2 \geq \lambda \|x - y\|_2$
where $\lambda > 0$

Second Order Derivative is lower bounded $\|\nabla^2 f(x)\|_2 \geq \lambda$



Strongly Convex function

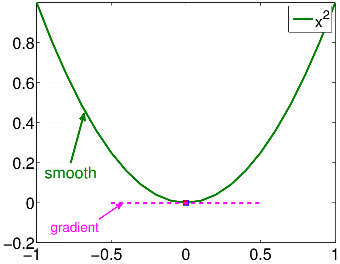
- strongly convex: e.g. Euclidean norm $f(x) = \frac{1}{2}\|x\|_2^2$

strong convexity constant

$$\|\nabla f(x) - \nabla f(y)\|_2 \geq \lambda\|x - y\|_2$$

where $\lambda > 0$

Second Order Derivative is lower bounded $\|\nabla^2 f(x)\|_2 \geq \lambda$



Smooth and Strongly Convex function

- smooth and strongly convex: e.g. quadratic function: $f(z) = \frac{1}{2}(z-1)^2$

$$\lambda\|x-y\|_2 \leq \|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x-y\|_2, \quad L \geq \lambda > 0$$

Chernoff bound

Let $X_1, \ldots, X_n$ be independent random variables. Assume $0 \leq X_i \leq 1$. Let $X = X_1 + \ldots + X_n$. $\mu = \mathrm{E}[X]$. Then

$$\Pr(X \geq (1+\epsilon)\mu) \leq \exp\left(-\frac{\epsilon^2}{2+\epsilon}\mu\right)$$

$$\Pr(X \leq (1-\epsilon)\mu) \leq \exp\left(-\frac{\epsilon^2}{2}\mu\right)$$

or

$$\Pr(|X-\mu| \geq \epsilon\mu) \leq 2\exp\left(-\frac{\epsilon^2}{2+\epsilon}\mu\right) \leq 2\exp\left(-\frac{\epsilon^2}{3}\mu\right)$$

the last inequality holds when $0 < \epsilon \leq 1$

Theoretical Guarantee of RA for low-rank approximation

$$X = U \begin{bmatrix} \Sigma_1 & \\ & \Sigma_2 \end{bmatrix} \begin{bmatrix} V_1^\top \\ V_2^\top \end{bmatrix}$$

- $X \in \mathbb{R}^{m\times n}$: the target matrix
- $\Sigma_1 \in \mathbb{R}^{k\times k}$, $V_1 \in \mathbb{R}^{n\times k}$
- $A \in \mathbb{R}^{n\times \ell}$: random reduction matrix
- $Y = XA \in \mathbb{R}^{m\times \ell}$: the small sketch

Key inequality:

$$\|(I - P_Y)X\|^2 \leq \|\Sigma_2\|^2 + \|\Sigma_2\Omega_2\Omega_1^\dagger\|^2$$

Gaussian Matrices

- G is a standard Gaussian matrix
- U and V are orthonormal matrices
- $U^T GV$ follows the standard Gaussian distribution
- $\mathrm{E}[\|SGT\|_F^2] = \|S\|_F^2 \|T\|_F^2$
- $\mathrm{E}[\|SGT\|] \leq \|S\| \|T\|_F + \|S\|_F \|T\|$
- Concentration for function of a Gaussian matrix. Suppose h is a Lipschitz function on matrices

  $$h(X) - h(Y) \leq L\|X - Y\|_F$$

  Then

  $$\Pr(h(G) \geq \mathrm{E}[h(G)] + Lt) \leq e^{-t^2/2}$$

Analysis for Randomized Least-square regression

Let $X = U\Sigma V^\top$

$$\mathbf{w}_* = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \|X\mathbf{w} - b\|_2$$

Let $Z = \|X\mathbf{w}_* - b\|_2$, $\omega = b - X\mathbf{w}_*$, and $X\mathbf{w}_* = U\alpha$

$$\widehat{\mathbf{w}}_* = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \|A(X\mathbf{w} - b)\|_2$$

Since $b - X\mathbf{w}_* = b - X(X^\top X)^\dagger X^\top b = (I - UU^\top)b$, $X\widehat{\mathbf{w}}_* - X\mathbf{w}_* = U\beta$. Then

$$\|X\widehat{\mathbf{w}}_* - b\|_2 = \|X\mathbf{w}_* - b\|_2 + \|X\widehat{\mathbf{w}}_* - X\mathbf{w}\|_2 = Z + \|\beta\|_2$$

Analysis for Randomized Least-square regression

$$AU(\alpha + \beta) = AX\widehat{\mathbf{w}}_* = AX(AX)^\dagger Ab = P_{AX}(Ab) = P_{AU}(Ab)$$

$$P_{AU}(Ab) = P_{AU}(A(\omega + U\alpha)) = AU\alpha + P_{AU}(A\omega)$$

Hence

$$U^\top A^\top AU\beta = (AU)^\top (AU)(AU)^\dagger A\omega = (AU)^\top (AU)((AU)^\top AU)^{-1}(AU)^\top A\omega$$

where we use *AU* is full column matrix. Then

$$U^\top A^\top AU\beta = U^\top A^\top A\omega$$

$$\|\beta\|_2^2/2 \leq \|U^\top A^\top AU\beta\|_2^2 = \|U^\top A^\top A\omega\|_2^2 \leq \epsilon'^2 \|U\|_F^2 \|\omega\|_2^2$$

where the last inequality uses the matrix products approximation shown in next slide. Since $\|U\|_F^2 \leq d$, setting $\epsilon' = \sqrt{\frac{\epsilon}{d}}$ suffices.

Approximate Matrix Products

Given $X \in \mathbb{R}^{n\times d}$ and $Y \in \mathbb{R}^{d\times p}$, let $A \in \mathbb{R}^{m\times d}$ one of the following matrices

- a JL transform matrix with $m = \Theta(\epsilon^{-2}\log((n+p)/\delta))$
- the sparse subspace embedding with $m = \Theta(\epsilon^{-2})$
- leverage-score sampling matrix based on $p_i \geq \frac{\|X_{i*}\|_2^2}{2\|X\|_F^2}$ and $m = \Theta(\epsilon^{-2})$

Then w.h.p $1-\delta$

$$\|XA^\top AY - XY\|_F \leq \epsilon\|X\|_F\|Y\|_F$$

Analysis for Randomized Least-square regression

$A \in \mathbb{R}^{m\times n}$

1. Subspace embedding: AU full column rank
2. Matrix product approximation: $\sqrt{\epsilon/d}$

Order of m

- JL transforms: 1. $O(d\log(d))$, 2. $O(d\log(d)\epsilon^{-1}) \Rightarrow O(d\log(d)\epsilon^{-1})$
- Sparse subspace embedding: 1. $O(d^2)$, 2. $O(d\epsilon^{-1}) \Rightarrow O(d^2\epsilon^{-1})$

If we use SSE ($A_1 \in \mathbb{R}^{m_1\times n}$) and JL transform $A_2 \in \mathbb{R}^{m_2\times m_1}$

$$\begin{aligned}\|A_2A_1(X\mathbf{w}_*^2 - b)\|_2 &\leq (1+\epsilon)\|A_1(X\mathbf{w}_*^1 - b)\|_2\\ &\leq (1+\epsilon)\|A_1(X\mathbf{w}_* - b)\|_2 \leq (1+\epsilon)^2\|X\mathbf{w}_* - b\|\end{aligned}$$

with $m_1 = O(d^2\epsilon^{-2})$ and $m_2 = d\log(d)\epsilon^{-1}$, $\mathbf{w}_*^2$ is the optimal solution using A_2A_1 and $\mathbf{w}_*^1$ is the optimal using A_1 and $\mathbf{w}_*$ is the original optimal solution.

Randomized Least-squares regression

Theoretical Guarantees (Sarlós, 2006; Drineas et al., 2011; Nelson & Nguyen, 2012):

$$\|X\widehat{\mathbf{w}}_* - b\|_2 \leq (1+\epsilon)\|X\mathbf{w}_* - b\|_2$$

- If A is a fast JL transform with $m = \Theta(\epsilon^{-1} d\log(d))$: Total Time $O(nd\log(m) + d^3\log(d)\epsilon^{-1})$
- If A is a Sparse Subspace Embedding with $m = \Theta(d^2\epsilon^{-1})$: Total Time $O(nnz(X) + d^4\epsilon^{-1})$
- If $A = A_1A_2$ combine fast JL ($m_1 = \Theta(\epsilon^{-1}d\log(d))$) and SSE ($m_2 = \Theta(d^2\epsilon^{-2})$): Total Time $O(nnz(X) + d^3\log(d/\epsilon)\epsilon^{-2})$

Matrix Chernoff bound

Lemma (Matrix Chernoff (Tropp, 2012))

Let $\mathcal{X}$ be a finite set of PSD matrices with dimension k, and suppose that $\max_{X\in\mathcal{X}} \lambda_{\max}(X) \le B$. *Sample* $\{X_1, \ldots, X_\ell\}$ *independently from* $\mathcal{X}$. *Compute*

$$\mu_{\max} = \ell\lambda_{\max}(\mathrm{E}[X_1]), \quad \mu_{\min} = \ell\lambda_{\min}(\mathrm{E}[X_1])$$

Then

$$\Pr\left\{\lambda_{\max}\left(\sum_{i=1}^{\ell} X_i\right) \ge (1+\delta)\mu_{\max}\right\} \le k\left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]^{\frac{\mu_{\max}}{B}}$$

$$\Pr\left\{\lambda_{\min}\left(\sum_{i=1}^{\ell} X_i\right) \le (1-\delta)\mu_{\min}\right\} \le k\left[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right]^{\frac{\mu_{\min}}{B}}$$

To simplify the usage of Matrix Chernoff bound, we note that

$$\left[\frac{e^{-\delta}}{[1-\delta]^{1-\delta}}\right]^{\mu} \leq \exp\left(-\frac{\delta^2}{2}\right)$$

$$\left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]^{\mu} \leq \exp\left(-\mu\delta^2/3\right), \delta \leq 1$$

$$\left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]^{\mu} \leq \exp\left(-\mu\delta\log(\delta)/2\right), \delta > 1$$

Noncommutative Bernstein Inequality

Lemma (Noncommutative Bernstein Inequality (Recht, 2011))

Let $Z_1, \ldots, Z_L$ be independent zero-mean random matrices of dimension $d_1 \times d_2$. Suppose $\tau_j^2 = \max\left\{\|\mathrm{E}[Z_j Z_j^\top]\|_2, \|\mathrm{E}[Z_j^\top Z_j\|_2\right\}$ and $\|Z_j\|_2 \leq M$ almost surely for all k. Then, for any $\epsilon > 0$,

$$\Pr\left[\left\|\sum_{j=1}^{L} Z_j\right\|_2 > \epsilon\right] \leq (d_1 + d_2)\exp\left[\frac{-\epsilon^2/2}{\sum_{j=1}^{L} \tau_j^2 + M\epsilon/3}\right]$$

Randomized Algorithms for K-means Clustering

K-means:

$$\sum_{j=1}^{k}\sum_{\mathbf{x}_i\in C_j}\|\mathbf{x}_i-\mu_j\|_2^2=\|X-CC^\top X\|_F^2$$

where $C\in\mathbb{R}^{n\times k}$ is the scaled cluster indicator matrix such that $C^\top C=I$.

Constrained Low-rank Approximation (Cohen et al., 2015)

$$\min_{P\in\mathcal{S}}\|X-PX\|_F^2$$

where $\mathcal{S}=QQ^\top$ is any set of rank k orthogonal projection matrix with orthonormal $Q\in\mathbb{R}^{n\times k}$

Low-rank Approximation: $\mathcal{S}$ is the set of all rank k orthogonal projection matrix. $P^*=U_kU_k^\top$

Randomized Algorithms for K-means Clustering

K-means:

$$\sum_{j=1}^{k} \sum_{\mathbf{x}_i \in C_j} \|\mathbf{x}_i - \mu_j\|_2^2 = \|X - CC^\top X\|_F^2$$

where $C \in \mathbb{R}^{n \times k}$ is the scaled cluster indicator matrix such that $C^\top C = I$.

Constrained Low-rank Approximation (Cohen et al., 2015)

$$\min_{P \in \mathcal{S}} \|X - PX\|_F^2$$

where $\mathcal{S} = QQ^\top$ is any set of rank k orthogonal projection matrix with orthonormal $Q \in \mathbb{R}^{n \times k}$

Low-rank Approximation: $\mathcal{S}$ is the set of all rank k orthogonal projection matrix. $P^* = U_k U_k^\top$

Randomized Algorithms for K-means Clustering

K-means:

$$\sum_{j=1}^{k} \sum_{\mathbf{x}_i \in C_j} \|\mathbf{x}_i - \mu_j\|_2^2 = \|X - CC^\top X\|_F^2$$

where $C \in \mathbb{R}^{n \times k}$ is the scaled cluster indicator matrix such that $C^\top C = I$.

Constrained Low-rank Approximation (Cohen et al., 2015)

$$\min_{P \in \mathcal{S}} \|X - PX\|_F^2$$

where $\mathcal{S} = QQ^\top$ is any set of rank k orthogonal projection matrix with orthonormal $Q \in \mathbb{R}^{n \times k}$

Low-rank Approximation: $\mathcal{S}$ is the set of all rank k orthogonal projection matrix. $P^* = U_k U_k^\top$

Randomized Algorithms for K-means Clustering

Define

$$\widehat{P}^* = \min_{P \in \mathcal{S}} \|\widehat{X} - P\widehat{X}\|_F^2$$

$$P^* = \min_{P \in \mathcal{S}} \|X - PX\|_F^2$$

Guarantees on Approximation

$$\|X - \widehat{P}^* X\|_F^2 \leq \frac{1+\epsilon}{1-\epsilon}\|X - P^* X\|_F^2$$

Properties of Leverage-score sampling

We prove the properties using Matrix Chernoff bound. Let $\Omega = AU$.

$$\Omega^\top \Omega = (AU)^\top (AU) = \sum_{j=1}^{m} \frac{1}{mp_{i_j}} \mathbf{u}_{i_j} \mathbf{u}_{i_j}^\top$$

Let $X_i = \frac{1}{mp_i}\mathbf{u}_i\mathbf{u}_i^\top$. $\mathrm{E}[X_i] = \frac{1}{m} I_k$. Therefore $\lambda_{\max}(X_i) = \lambda_{\min}(X_i) = \frac{1}{m}$. And $\lambda_{\max}(X_i) \leq \max_i \frac{\|\mathbf{u}_i\|_2^2}{mp_i} = \frac{k}{m}$. Applying the Matrix Chernoff bound for the minimum and maximum eigen-value, we have

$$\Pr(\lambda_{\min}(\Omega^\top\Omega) \leq (1-\epsilon)) \leq k \exp\left(-\frac{m\epsilon^2}{2k}\right) \leq k \exp\left(-\frac{m\epsilon^2}{3k}\right)$$

$$\Pr(\lambda_{\max}(\Omega^\top\Omega) \geq (1+\epsilon)) \leq k \exp\left(-\frac{m\epsilon^2}{3k}\right)$$

When uniform sampling makes sense?

Coherence measure

$$\mu_k = \frac{d}{k} \max_{1 \leq i \leq d} \|U_{i*}\|_2^2$$

When $\mu_k \leq \tau$ and $m = \Theta\left(\frac{k\tau}{\epsilon^2} \log\left[\frac{2k}{\delta}\right]\right)$ w.h.p $1 - \delta$,

- A formed by uniform sampling (and scaling)
- $AU \in \mathbb{R}^{m \times k}$ is full column rank
- $\sigma_i^2(AU) \geq (1 - \epsilon) \geq (1 - \epsilon)^2$
- $\sigma_i^2(AU) \leq (1 + \epsilon) \leq (1 + \epsilon)^2$
- Valid when the coherence measure is small (some real data mining datasets have small coherence measures)
- The Nyström method usually uses uniform sampling (Gittens, 2011)