

Optimization in the “Big Data” Regime

Sham M. Kakade

Machine Learning for Big Data
CSE547/STAT548

University of Washington

Announcements...

- HW2 due Mon.
- Work on your project milestones
 - read/related work summary
 - some empirical work

Today:

- Review: discuss classical optimization
- New: How do we optimize in the “big data” regime, with large sample sizes and large dimension?
Bridge classical to modern optimization.

Machine Learning and the Big Data Regime...

goal: find a d -dim parameter vector which minimizes the loss on n training examples.

- have n training examples $(x_1, y_1), \dots, (x_n, y_n)$
- have parametric a classifier $h_\theta(x, w)$, where w is a d dimensional vector.

$$\min_w L(w) \text{ where } L(w) = \sum_i \text{loss}(h(x_i, w), y_i)$$

- “Big Data Regime”: How do you optimize this when n and d are large? memory? parallelization?

Can we obtain linear time algorithms to find an ϵ -accurate solution?

i.e. find \hat{w} so that

$$L(\hat{w}) - \min_w L(w) \leq \epsilon$$

Plan:

- Goal: algorithms to get fixed target accuracy ϵ .
- Review: classical optimization viewpoints
- A modern view: can we bridge classical optimization to modern problems?
 - Dual Coordinate Descent Methods
 - Stochastic Variance Reduced Gradient method (SVRG)

Abstraction: Least Squares

$$\min_w L(w) \text{ where } L(w) = \sum_{i=1}^n (w \cdot x_i - y_i)^2 + \lambda \|w\|^2$$

How much computation time is required to get ϵ accuracy?

- n points, d dimensions.
- “Big Data Regime”: How do you optimize this when n and d are large?
- More general case: Optimize sums of convex (or non-convex) functions?
 - some guarantees will still hold

Aside: think of x as a large feature representation.

Review: Direct Solution

$$\min_w L(w) \text{ where } L(w) = \sum_{i=1}^n (w \cdot x_i - y_i)^2 + \lambda \|w\|^2$$

- solution:

$$w = (X^T X + \lambda I)^{-1} X^T Y$$

where X be the $n \times d$ matrix whose rows are x_i , and Y is an n -dim vector.

- numerical solution: the “backslash” implementation.
- time complexity: $O(nd^2)$ and memory $O(d^2)$

Not feasible due to both time and memory.

Review: Gradient Descent (and Conjugate GD)

$$\min_w L(w) \text{ where } L(w) = \sum_{i=1}^n (w \cdot x_i - y_i)^2 + \lambda \|w\|^2$$

- n points, d dimensions,
- $\lambda_{\max}, \lambda_{\min}$ are max and min eigs. of “design matrix” $\frac{1}{n} \sum_i x_i x_i^T$
- # iterations and computation time to get ϵ accuracy:
 - Gradient Descent (GD):

$$\frac{\lambda_{\max}}{\lambda_{\min}} \log 1/\epsilon, \quad \frac{\lambda_{\max}}{\lambda_{\min}} nd \log 1/\epsilon$$

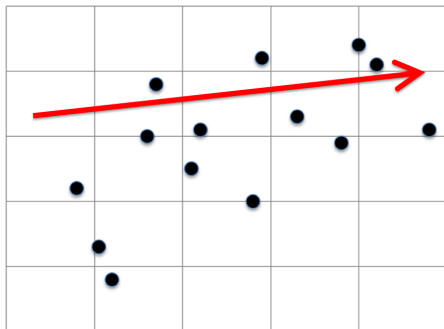
- Conjugate Gradient Descent:

$$\sqrt{\frac{\lambda_{\max}}{\lambda_{\min}}} \log 1/\epsilon, \quad \sqrt{\frac{\lambda_{\max}}{\lambda_{\min}}} nd \log 1/\epsilon$$

- memory: $O(d)$

Better runtime and memory, but still costly.

Review: Stochastic Gradient Descent (SGD)

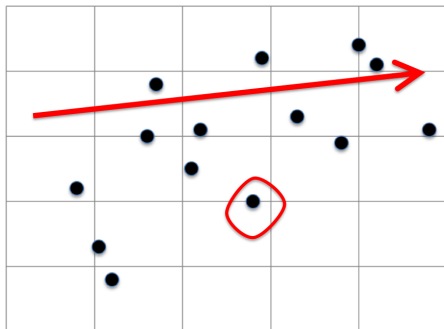


- SGD update rule: at each time t ,

sample a point (x_i, y_i)

$$w \leftarrow w - \eta(w \cdot x_i - y_i)x_i$$

Review: Stochastic Gradient Descent (SGD)

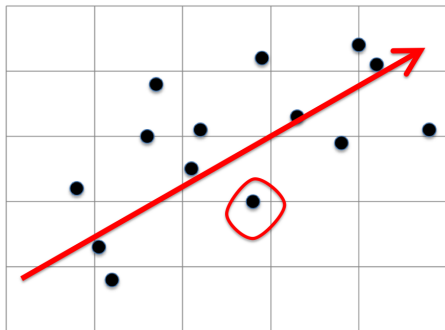


- SGD update rule: at each time t ,

sample a point (x_i, y_i)

$$w \leftarrow w - \eta(w \cdot x_i - y_i)x_i$$

Review: Stochastic Gradient Descent (SGD)



- **SGD update rule:** at each time t ,

sample a point (x_i, y_i)

$$w \leftarrow w - \eta(w \cdot x_i - y_i)x_i$$

- **Problem:** even if $w = w_*$, the update changes w .

Rate: convergence rate is $O(1/\epsilon)$, with decaying η

simple algorithm, light on memory, but poor convergence rate

Review: Stochastic Gradient Descent

- λ_{\min} is the min eig. of $\frac{1}{n} \sum_i x_i x_i^\top$
- Suppose gradients are bounded by B .
- To get ϵ accuracy:
 - # iterations to get ϵ -accuracy:

$$\frac{B^2}{\lambda_{\min} \epsilon}$$

- Computation time to get ϵ -accuracy:

$$\frac{dB^2}{\lambda_{\min} \epsilon}$$

Regression in the big data regime?

$$\min_w L(w)$$

How much computation time is required to get ϵ accuracy?

- “Big Data Regime”: How do you optimize this when n and d are large?
 - Can we ‘fix’ the instabilities of SGD?
- Let’s look at (regularized) linear regression.
 - **Convex optimization**: All results can be generalized to smooth+strongly convex loss functions.
 -
 - **Non-convex optimization**: some ideas generalize.

Duality (without Duality)

$$\begin{aligned}w &= (X^T X + \lambda I)^{-1} X^T Y \\ &= X^T (X X^T + \lambda I)^{-1} Y \\ &:= \frac{1}{\lambda} X^T \alpha\end{aligned}$$

where $\alpha = (I + X X^T / \lambda)^{-1} Y$.

- **idea:** let's compute the n-dim vector α .
- let's do this with coordinate ascent

SDCA: stochastic dual coordinate ascent

$$G(\alpha_1, \alpha_2, \dots, \alpha_n) = \frac{1}{2} \alpha^\top (I + \mathbf{X}\mathbf{X}^\top / \lambda) \alpha - \mathbf{Y}^\top \alpha$$

- the minimizer of $G(\alpha)$ is

$$\alpha = (I + \mathbf{X}\mathbf{X}^\top / \lambda)^{-1} \mathbf{Y}$$

- SDCA:

- start with $\alpha = 0$.
- choose coordinate i randomly, and update:

$$\alpha_j = \operatorname{argmin}_z G(\alpha_1, \dots, \alpha_{i-1}, z, \dots, \alpha_n)$$

- easy to do as we touch just **one** datapoint.
- return $w = \frac{1}{\lambda} \mathbf{X}^\top \alpha$.

SDCA: the algorithm

$$G(\alpha_1, \alpha_2, \dots, \alpha_n) = \frac{1}{2} \alpha^\top (I + \mathbf{X}\mathbf{X}^\top / \lambda) \alpha - \mathbf{Y}^\top \alpha$$

- start with $\alpha = 0$, $\mathbf{w} = \frac{1}{\lambda} \mathbf{X}^\top \alpha$.

- 1 choose coordinate i randomly, and compute difference:

$$\Delta \alpha_i = \frac{(y_i - \mathbf{w} \cdot \mathbf{x}_i) - \alpha_i}{1 + \|\mathbf{x}_i\|^2 / \lambda}$$

- 2 update:

$$\alpha_i \leftarrow \alpha_i + \Delta \alpha_i, \quad \mathbf{w} \leftarrow \mathbf{w} + \frac{1}{\lambda} \mathbf{x}_i \cdot \Delta \alpha_i$$

- return $\mathbf{w} = \frac{1}{\lambda} \mathbf{X}^\top \alpha$.

Guarantees: speedups for the big data regime

- n points, d dimensions, λ_{av} average eigenvalue
- Computation time to get ϵ accuracy gradient descent:
(Shalev-Shwartz & Zhang '12)

- GD vs SDCA:

$$\frac{\lambda_{\max}}{\lambda_{\min}} n d \log 1/\epsilon \rightarrow \left(n + d \frac{\lambda_{av}}{\lambda_{\min}} \right) d \log 1/\epsilon$$

- conjugate GD vs acceleration+SDCA.
One can accelerate SDCA as well. (Frosting, Ge, K., Sidford, 2015))

Comparisons to GD

- both algorithms touch one data point at a time, with same computational cost per iteration.
- SDCA has “learning rate” which adaptive to the data point.
- GD has convergence rate of $1/\epsilon$ and SDCA has $\log 1/\epsilon$ convergence rate.
- **memory:** SDCA: $O(n + d)$, SGD: $O(d)$
- SDCA: can touch points in *any* order.

SDCA advantages/disadvantages

- What about more general convex problems? e.g.

$$\min_w L(w) \text{ where } L(w) = \sum_i \text{loss}(h(x_i, w), y_i)$$

- the basic idea (formalized with duality) is pretty general for convex $\text{loss}(\cdot)$.
- works **very** well in practice.
- **memory**: SDCA needs $O(n + d)$ memory, while SGD is only $O(d)$.
- What about an algorithm for non-convex problems?
 - SDCA seems heavily tied to the convex case.
 - would an algo that is highly accurate in the convex case and sensible in the non-convex case.

(another idea) Stochastic Variance Reduced Gradient (SVRG)

- 1 **exact gradient computation:** at stage s , using \tilde{w}_s , compute:

$$\nabla L(\tilde{w}_s) = \frac{1}{n} \sum_{i=1}^n \nabla \text{loss}(\tilde{w}_s, (x_i, y_i))$$

- 2 **corrected SGD:** initialize $w \leftarrow \tilde{w}_s$. for m steps,

sample a point (x, y)

$$w \leftarrow w - \eta (\nabla \text{loss}(w, (x, y)) - \nabla \text{loss}(\tilde{w}_s, (x, y)) + \nabla L(\tilde{w}_s))$$

- 3 **update and repeat:** $\tilde{w}_{s+1} \leftarrow w$.

(another idea) Stochastic Variance Reduced Gradient (SVRG)

- 1 **exact gradient computation:** at stage s , using \tilde{w}_s , compute:

$$\nabla L(\tilde{w}_s) = \frac{1}{n} \sum_{i=1}^n \nabla \text{loss}(\tilde{w}_s, (x_i, y_i))$$

- 2 **corrected SGD:** initialize $w \leftarrow \tilde{w}_s$. for m steps,

sample a point (x, y)

$$w \leftarrow w - \eta \left(\nabla \text{loss}(w, (x, y)) - \nabla \text{loss}(\tilde{w}_s, (x, y)) + \nabla L(\tilde{w}_s) \right)$$

- 3 **update and repeat:** $\tilde{w}_{s+1} \leftarrow w$.

Two ideas:

- If $\tilde{w} = w_*$, then no update.
- unbiased updates: **blue term** is mean 0.

Guarantees of SVRG

- n points, d dimensions, λ_{av} average eigenvalue
- Computation time to get ϵ accuracy gradient descent:
(Johnson & Zhang '13)
 - GD vs SDCA:

$$\frac{\lambda_{\max}}{\lambda_{\min}} n d \log 1/\epsilon \rightarrow \left(n + d \frac{\lambda_{\text{av}}}{\lambda_{\min}} \right) d \log 1/\epsilon$$

- conjugate GD vs ??

$$\sqrt{\frac{\lambda_{\max}}{\lambda_{\min}}} n d \log 1/\epsilon \rightarrow ??$$

- memory: $O(d)$