Optimization in the “Big Data” Regime

Sham M. Kakade

Machine Learning for Big Data
CSE547/STAT548

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

- have $n$ training examples $(x_1, y_1), \ldots (x_n, y_n)$
- 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 $\epsilon$-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 $\epsilon$.
- Review: classical optimization viewpoints
- A modern view: can be 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 \( \epsilon \) 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.
min \( L(w) \) where \( L(w) = \sum_{i=1}^{n} (w \cdot x_i - y_i)^2 + \lambda ||w||^2 \)

solution:

\[ w = (X^\top X + \lambda I)^{-1} X^\top Y \]

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,
- \(\lambda_{\text{max}}, \lambda_{\text{min}}\) are max and min eigs. of “design matrix” \(\frac{1}{n} \sum_i x_i x_i^\top\)
- \# iterations and computation time to get \(\epsilon\) accuracy:
  - Gradient Descent (GD):
    \[
    \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} \log \frac{1}{\epsilon}, \quad \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} n d \log \frac{1}{\epsilon}
    \]
  - Conjugate Gradient Descent:
    \[
    \sqrt{\frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}} \log \frac{1}{\epsilon}, \quad \sqrt{\frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}} n d \log \frac{1}{\epsilon}
    \]
- 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 $(x_i, y_i)$

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

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

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

Simple algorithm, light on memory, but poor convergence rate.
**SGD update rule:** at each time $t$,

sample a point $(x_i, y_i)$

$$w \leftarrow w - \eta (w \cdot x_i - y_i)x_i$$
SGD update rule: at each time $t$,

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

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

Rate: convergence rate is $O(1/\epsilon)$, with decaying $\eta$ simple algorithm, light on memory, but poor convergence rate
\( \lambda_{\text{min}} \) is the min eig. of \( \frac{1}{n} \sum_i x_i x_i^\top \)

Suppose gradients are bounded by \( B \).

To get \( \epsilon \) accuracy:

- # iterations to get \( \epsilon \)-accuracy:
  \[
  \frac{B^2}{\lambda_{\text{min}} \epsilon}
  \]

- Computation time to get \( \epsilon \)-accuracy:
  \[
  \frac{dB^2}{\lambda_{\text{min}} \epsilon}
  \]
Regression in the big data regime?

$$\min_w L(w)$$

How much computation time is required to get $\epsilon$ 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)

\[ w = (X^\top X + \lambda I)^{-1} X^\top Y \]
\[ = X^\top (XX^\top + \lambda I)^{-1} Y \]
\[ := \frac{1}{\lambda} X^\top \alpha \]

where \( \alpha = (I + XX^\top / \lambda)^{-1} Y \).

- **idea:** let’s compute the n-dim vector \( \alpha \).
- let’s do this with coordinate ascent
SDCA: stochastic dual coordinate ascent

\[ G(\alpha_1, \alpha_2, \ldots \alpha_n) = \frac{1}{2} \alpha^\top (I + XX^\top / \lambda) \alpha - Y^\top \alpha \]

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

\[ \alpha = (I + XX^\top / \lambda)^{-1} Y \]

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

\[ \alpha_i = \text{argmin}_z G(\alpha_1, \ldots \alpha_{i-1}, z, \ldots, \alpha_n) \]

- easy to do as we touch just one datapoint.
- return \( w = \frac{1}{\lambda} X^\top \alpha \).
SDCA: the algorithm

\[ G(\alpha_1, \alpha_2, \ldots \alpha_n) = \frac{1}{2} \alpha^\top (I + XX^\top / \lambda) \alpha - Y^\top \alpha \]

1. start with \( \alpha = 0, \ w = \frac{1}{\lambda} X^\top \alpha \).
2. choose coordinate \( i \) randomly, and compute difference:
   \[ \Delta \alpha_i = \frac{(y_i - w \cdot x_i) - \alpha_i}{1 + \|x_i\|^2 / \lambda} \]
3. update:
   \[ \alpha_i \leftarrow \alpha_i + \Delta \alpha_i, \quad w \leftarrow w + \frac{1}{\lambda} x_i \cdot \Delta \alpha_i \]
4. return \( w = \frac{1}{\lambda} X^\top \alpha \).
Guarantees: speedups for the big data regime

- \( n \) points, \( d \) dimensions, \( \lambda_{av} \) average eigenvalue

Computation time to get \( \epsilon \) accuracy gradient descent: (Shalev-Shwartz & Zhang ’12)

- GD vs SDCA:

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

- 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.
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 loss(·). 
- 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 $(x, y)$

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

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

Two ideas:

- If $\tilde{w}_s = w^*$, then no update.
- unbiased updates: blue term is mean 0.
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, $\lambda_{av}$ average eigenvalue

- Computation time to get $\epsilon$ accuracy gradient descent: (Johnson & Zhang '13)
  - GD vs SDCA:
    \[
    \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} n d \log \frac{1}{\epsilon} \rightarrow \left(n + d \frac{\lambda_{av}}{\lambda_{\text{min}}} \right) d \log \frac{1}{\epsilon}
    \]
  - conjugate GD vs ??
    \[
    \sqrt{\frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}} n d \log \frac{1}{\epsilon} \rightarrow ??
    \]
  - memory: $O(d)$