Block-coordinate optimization:

"huge scale" optimization → Divide & Conquer 2010-2013

Proposed: block-coordinate proximal gradient method

Optimization: \( \min_x f(x) \)
Setup: \( \text{s.t. } x \in \bigcup_i M_i \)

i.e., \( x = (x_1, \ldots, x_n) \) blocks

\( \triangleright \) pick \( i \) at random

Then let \( x^{(t+1)} = \text{Prox}_{M_i} \left( x_i^t - \frac{1}{\sigma_i} \nabla f(x_i^t) \right) \)

\( x_i^{(t+1)} = x_i^t + \sigma_i \nabla f_i(x_i^t) \)

Lipschitz constant for \( \partial f_i(x) \)

(only update update block \( i \) at iteration \( t \))

Nestrov showed get \( \mathbb{E} f(x^{(T)}) - f(x^*) \leq \frac{L^2}{2\sigma^2} \mathbb{E} \sum_{t=1}^{T} \| x_0 - x^* \|^2 \) (convex \( f \))

**Block-coordinate FW:** idea: do a FW step on block \( i \)

Algorithm:

\( \forall t = 0, 1, \ldots \)

pick \( i \) at random

Let \( S_i^t = \text{argmin}_{x_i \in M_i} f_i(x_i^t, s_i^t) \) (FW comes for block \( i \)) \( s_{t+1} = \left( \begin{array}{c} s_i^t \\ 0 \end{array} \right) \)

\( x_i^{(t+1)} = x_i^t + (1-z_i) s_i^t \)

\( x_{i+1}^{(t+1)} = x_i^{(t+1)} \) (or, i blocks)

\( z_i = \text{line search: } \text{argmin}_{z \in [0, 1]} f(x + \sigma (x_{i+1}^{(t+1)} - x_i^{(t+1)})) \)
an important property: 
\[ \text{FW gap} = \max \langle -\nabla f(x), s - x \rangle = \max_{s \in M} \langle -\nabla \bar{f}(x), s - x \rangle \]

\[ G^\text{FW}(x) = \sum_{i=1}^{\text{blocks}} G^i_f(x) \]

as before, (can show that)
\[ g^i(x) > f(x) - \min_{y \in \mathbb{R}} f(y) \]

\[ C_f^{(i)} \leq \text{Lip}_f(m_i) \]

convergence result: define
\[ C_f^{(t)} \leq \text{Lip}_f(m_i) + \frac{1}{2} \left( \frac{1}{t-1} \right) \]

\[ \text{during FW, if block } i \text{ is updated, then} \]
\[ f(x_8) = f(x) + x \langle \nabla f(x), s_i - x \rangle + \frac{x^2}{2} \frac{C_f^{(i)}}{3} \]

\[ -g^i(x) \]

\[ \text{consider } \alpha \text{ to result in} \]
\[ f(x^\text{new}(\alpha)) \leq f(x) - \alpha \left( g^i(x) + \frac{C_f^{(i)}}{2} \right) + \frac{1}{2} \epsilon^2 \text{Lip}_f^{(i)} \]
\[
\delta_{t+1} = (1 - \frac{\gamma}{\delta_0}) \delta_t + \frac{\gamma^2}{\delta_0} \]

\(\delta_{t+1} \leq \delta_t \) for \(\frac{\gamma}{\delta_0} \leq 1\)

\[
\mathbb{E}[\mathcal{L}(\theta(t)) - \mathcal{L}(\theta^*)] \leq \frac{2\mathbb{E}[\mathcal{L}(\theta) + \mathcal{L}(\theta'^*)]}{t+2n}
\]

for \(t \geq 0\) where \(t_0 \geq n \log \frac{\delta_0}{\delta^*} \), twice to ensure that \(\delta_0 \leq \delta^*\)

If you use line search:

\[
\mathbb{E}[\mathcal{L}(\theta(t)) - \mathcal{L}(\theta^*)] \leq \frac{2\mathbb{E}[\mathcal{L}(\theta) + \mathcal{L}(\theta'^*)]}{t-6+2n}
\]

one can show that \(\delta_0^{\frac{3}{2}} \leq \delta^*\) for quadratic functions

**BCFW update is \(n\) times cheaper than batch FW**

\(\Rightarrow\) BCFW is "nested" slower than batch FW

\(\Rightarrow\) BCFW is "nested" slower than batch FW

\(\Rightarrow\) BCFW is "nested" slower than batch FW

\(\Rightarrow\) BCFW is "nested" slower than batch FW

\(\Rightarrow\) BCFW is "nested" slower than batch FW

\(\Rightarrow\) BCFW is "nested" slower than batch FW

**Extensions**:

- non-uniform sampling e.g. using \(\mathcal{L}(\theta)\)

- using away steps etc. to "get" linear convergence

**Applications to SVM dual**:

- getting \(\delta^*\) is one loss-augmented decoding call on example?

- \(w = A^T x = \mathcal{L}^T A x\)
when you do a \(\text{FW} \) step, you update \( w_{t+1} = w_t + \gamma (w_t - w_t^{(t)}) \)

For SVM/nacht \( C_s^{(2)} = \frac{4B^2}{\lambda W} \Rightarrow C_s^{(2)} = \frac{4B^2}{\lambda W} \leq C_s \)

\textbf{Theory basis:}

\text{decision theory setup:}

\text{estimates } h_0 : x \rightarrow y

\text{generalization error} = \mathbb{E}_{(x,y) \sim P} \left[ \ell(y, h_0(x)) \right]

\text{ultimate goal is find} \; w^* = \arg \min \; L_P(w)

\text{problem is do not know } P \; (\text{distribution on } (x,y))

\text{suppose} \; (x^{(i)}, y^{(i)})_{i=1}^n \; i.i.d \; \text{P training data} \; \Rightarrow \; \text{we can look at} \; L_n(w) = \frac{1}{n} \sum_{i=1}^n \ell(y^{(i)}, h_0(x^{(i)}))

\text{algorithm}

\text{learning algorithm} \; \hat{w}_n = A(D_n)

\text{note that minimizing training error gives no much guarantees in general}

\text{from statistics/pred}

\text{\( L_n(w) \rightarrow \ell(w) \; \forall w \) \; note that this is much weaker than } \text{\( \sup \; \ell_n(w) - \ell(w) \downarrow \rightarrow 0 \)
For n points, cannot get zero training error with polynomial of degree $\geq n/2$.

Suppose you have no noise.

Real job is graduate.

Learning theory, we want to understand properties of learning algorithms.

In particular, what can I say about $L_p(A(D_n))$?

"Frequentist risk" $\mathbb{E}_{D_n \text{ random}}[L_p(A(D_n))] = \mathbb{E}_{D_n \text{ random}} \mathbb{P}(A)$

"PAC framework" $\mathbb{P} \geq 1 - \delta$ $L_p(A(D_n))$ is small bound $\delta \leq 8$

[which means $L_p(A(D_n)) \leq \text{bound with prob. larger than } 1 - \delta$ on random $D_n$]

"Bayesian posterior risk" $R_{\text{Bayes}}(w | D_n) = \mathbb{E}_{\Theta \text{ posterior}}[L_p(w)]$

Bayesian estimate $\hat{w} = \arg \min_w R_{\text{Bayes}}(w | D_n)$

Could analyze the property of $R_{\text{Bayes}}(A(D_n))$ i.e. $R_{\text{Bayes}}(\hat{w} | D_n)$.

Next time:

Occam's razor:

$\forall w \quad L_p(w) \leq L_{\text{complexity}}(w) + \text{"complexity"}(w)$ with prob $\geq 1 - \delta$.
Pointers:

- Nesterov coordinate method:

- BCFW in all its gory details:
    code: [https://github.com/ppletscher/BCFWstruct](https://github.com/ppletscher/BCFWstruct)

- Improvements of BCFW for SVMstruct (non-uniform sampling, away steps, etc.):