Application of BFW to SVMnet:

- Given $\mathbf{x}$ is one less-augmented decoding cell for example $i$
  $\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \Delta \mathbf{x}_i^{(t)}$ (where $\Delta \mathbf{x}_i^{(t)} = \mathbf{x}_i^{(t+1)} - \mathbf{x}_i^{(t)}$)

$\mathbf{w} = \mathbf{A} \mathbf{r}$

$\mathbf{w}_i^{(t+1)} = \mathbf{w}_i^{(t)} + \Delta \mathbf{w}_i^{(t)}$ (where $\Delta \mathbf{w}_i^{(t)} = \mathbf{w}_i^{(t+1)} - \mathbf{w}_i^{(t)}$)

$\mathbf{w}_i^{(t+1)} = \mathbf{w}_i^{(t)} \quad \forall i, \forall t$

(Least square change)

Need to store more in memory

or

Store C's

(Memory/Computation tradeoff)

Note: For LSE search, also need to store $b_i^{T} \mathbf{y}_i \leq b_i^{(t)}$

Convergence criteria:

For SVMnet, can show that $G_0^2 \leq \frac{4R_i^2}{\lambda n^2}$

$\mathbf{R}_i \equiv \max_{\mathbf{y} \in \Omega} \lVert \mathbf{y} \rVert_2$

$R = \max_{i} R_i$

Hessian: $(\mathbf{ATA})_{ij,ij} = \frac{\lambda \Sigma \eta_i(y_i) \psi_i(y_j)}{\lambda n^2}$

$\langle \mathbf{H}_i \mathbf{d}_i, \mathbf{d}_i \rangle \leq \frac{\Sigma \eta_i(y_i) \psi_i(y_i)}{y_i y_j}$

$\leq \frac{\Sigma \eta_i(y_i) \psi_i(y_i)}{y_i y_j}$

$\leq R_i^2$ (where $\psi_i(y_i)$ is the $i$th element of $\mathbf{y}$)

$\leq R_i^2 \mathbf{1}_{d_i, d_i}^T \mathbf{1}_{d_i, d_i}$

$\leq \frac{4n_i^2}{\lambda}$

Compare with $G_0^2 \leq \frac{4R_i^2}{\lambda n^2}$

i.e., BFW is "in hours" faster than batch FW for SVM net

Impatience of affine invariance:

Skipped:

Importance of affine invariance:
importance of off-diagonal invariants: 
if you use $l_2$-norm, we get a bad bound $\text{diam}(M) \leq 2n$

$$||M||_{l_2} \leq \text{diam}(M)$$

Lipschitz constant $\text{in } l_2$-norm = largest e-value of Hessian

(recall Hessian $\text{ATA} = I$ $\langle \nabla^2 f(y), \nabla^2 f(y) \rangle$)

say e.g. $\langle \nabla^2 f(y), \nabla^2 f(y) \rangle \leq 1$ for lab of output

$A_n^2 (A^T _n A_n) \approx (\text{dim}) \cdot 1$

$\Rightarrow$ get largest e-value can scale with dimension of a matrix
$\Rightarrow$ really bad here because dim is exponentially

Instead, want to use $l_1$-norm of $\Delta x_i$

i.e. $l_1$-norm for Lipschitz constant

Then get $L_{\text{diam}}(\Delta x_i) \approx \sqrt{\frac{L^2}{\text{dim}}}$

**Variance Reduced SGD**

\[ \text{step } = \min_x \sum_{i=1}^{n} \frac{1}{\delta} f_i(x) \]

where $f$ is $\mu$-strongly convex

$L$-smooth

\[ f(x) - f(x^*) \leq \frac{L}{2} \left( \sum_{i=1}^{\text{diam}} \right) \]

**Batch Gradient Method**

[Cauchy 1849, continuing]

**Stochastic Gradient Method**

[aka. incremental gradient method]

[Robbins & Monro 1951]

\[ \text{log}(f(x) - f^*) \]

6D

Stochastic gradient method

\[ x_{t+1} = x_t - \eta_t \nabla f_t(x_t) \]

where $\eta_t \sim \text{unif}(\mathbb{R})$

$O(1)$ to compute

\[ Y_t = \text{const} \cdot \gamma \]

$\Rightarrow$ learn rate up to a ball of radius $\gamma$

$Y_t = \frac{1}{\sqrt{t}}$

$\Rightarrow O(\sqrt{\frac{1}{t}})$ rate

i.e. sublinear
**SAG (stochastic average gradient)**

[Le Roux, Schmidt & Bach NeurIPS 2012]

\[ \text{SAG}: \quad \text{store past gradients for each } c \quad (g_j) \]
\[ \text{at step } t \]
\[ \text{update } g_{t+1}^c = Dg_t^c - \frac{\nabla f(x_t)}{n} \quad g_{t+1} = g_{t+1}^c + \frac{g_t}{n} \quad \text{for } j \in 1 \ldots C \]

\[ x_{t+1} = x_t - \frac{1}{n} \sum_{i=1}^n g_{t+1}^i \text{ e-averaged state gradients} \]

\[ \frac{1}{n} \sum_{i=1}^n g_{t+1}^i \text{ approx of } Dg(x_t) \]

O(1) cost per iteration
Big surprise: converge linearly and fast

[but O(nd) storage cost in worst case]
"invariant aggregated gradient" (ITG)
[Blatt & al. 2007]

where you cycle deterministically through \( i_1, \ldots, i_C \)

\( \rightarrow \) linear rate for quadratic functions

but \( \gamma_{max} \approx O\left(\frac{L}{\delta}ight) \) (step size)

**SAG Convergence rate**
Thin with \( x_t = \frac{1}{L} L_{gd} \)

\[ \text{where } L = \max_c L_{gd}(c) \]

\[ \| f(x_t) - f^* \| \leq \left( 1 - \frac{\delta}{L} \right) \frac{1}{L} + \frac{1}{2} \bar{C}_0 \]

\[ D_{SAG} = \min \left\{ \frac{\delta}{L} \frac{1}{k_{gd}}, \frac{1}{8n} \right\} \text{ vs. } D_{grad} \approx \frac{1}{k_{grad}} \]

**Example**: \( l_2 \)-reg. log. regression on RCV1

\[ n = 700k \quad L = 0.25 \quad \mu = \frac{1}{n} \quad (k = \frac{n}{4}) \]

rate comparison gradient method \( \left( \frac{L+\bar{C}_0}{L+\mu} \right)^2 \approx 0.9998 \)
\[ \text{AC (n iterations)} \quad (1 - D_n) < 0.9999 \]

Practical aspects (see Schmidt et al., Math. Prog. 2016 paper):

a) Storage: \( f_i(w) = h(x_i w) \Rightarrow \nabla f_i(w) = h'(x_i w) x_i \)
   instead of storage
   \( \Rightarrow O(n) \) storage

b) Initialization of \( v_i \)'s? best: run SNO for one pass then switch SAC/SAC

c) Step size? \( \beta = \frac{1}{\sqrt{L}} \)
   - Adaptive learning rate
   - Hometic search (comes from FISTA)

\[ f_i(w) - \frac{1}{L} \nabla f_i(w) \geq f_i(w_t) \]
while \( f_i(w_t - \frac{1}{L} \nabla f_i(w_t)) \geq \frac{1}{L} \nabla f_i(w_t) \)
   set \( v_i = v_i \)
   else \( v_i = \frac{1}{a} \)

d) Non-uniform sampling?
   Sample \( i \sim \frac{1}{\sqrt{L}} \)

e) Stopping criterion?
   You can use \( \frac{1}{n} \sum_{i=1}^{n} f_i(x_t) \) to approximate \( \sum_{i=1}^{n} f_i(x_t) \)

\( \beta = \beta_{t+1} \quad x_{t+1} = x_t - \gamma \left[ \nabla f_i(x_t) - \frac{1}{L} \nabla f_i(x_t) \right] \)

g) Sparse features?
   - Sparse weighted projection on support of \( x_i \)
   \[ S_i = \frac{x_i}{u_i} : (x_i)_u > 0 \]