Large-Scale Optimization

Sanjiv Kumar, Google Research, NY

EECS-6898, Columbia University - Fall, 2010

Learning and Optimization

In most cases, learning from data reduces to optimizing a function with respect to model parameters

Given training data $D = \{x_i, y_i\}_{i=1,\dots,n}$ we want to learn a model

prediction $y = f(x; w)$ $\qquad$ $e.g., x \in \Re^d, y \in \Re$ or $y \in \{-1, 1\}$

Learning and Optimization

In most cases, learning from data reduces to optimizing a function with respect to model parameters

Given training data $D = \{x_i, y_i\}_{i=1,\dots,n}$ we want to learn a model

prediction $y = f(x; w)$ $\qquad e.g., x \in \Re^d, y \in \Re$ or $y \in \{-1, 1\}$

$$\hat{w} = \arg\min_w J(w)$$
$$= \arg\min_w [\underbrace{L(D, w)}_{\text{loss}} + \underbrace{\lambda R(w)}_{\text{regularizer}}]$$

Learning and Optimization

In most cases, learning from data reduces to optimizing a function with respect to model parameters

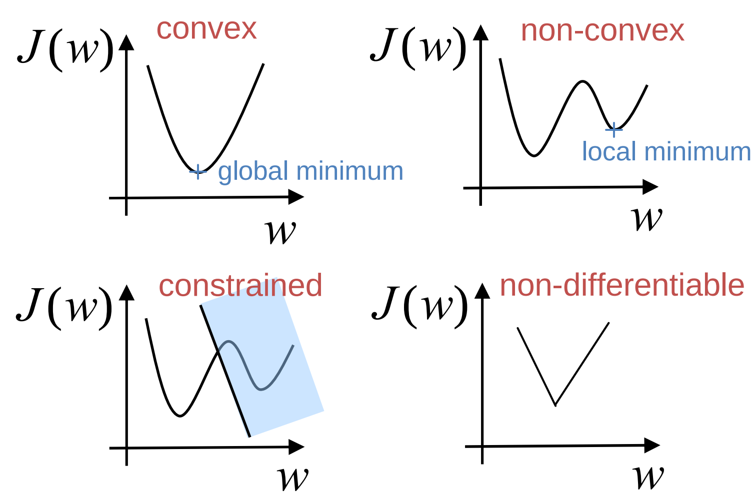
Given training data $D = \{x_i, y_i\}_{i=1,\ldots,n}$ we want to learn a model

prediction $y = f(x\,; w)$ $\quad$ $e.g., x \in \Re^d, y \in \Re$ or $y \in \{-1, 1\}$

$$\hat{w} = \arg\min_w J(w)$$
$$= \arg\min_w [\underbrace{L(D,w)}_{\text{loss}} + \underbrace{\lambda R(w)}_{\text{regularizer}}]$$

- Convex vs non-convex
- Constrained vs unconstrained
- Smooth vs non-differentiable

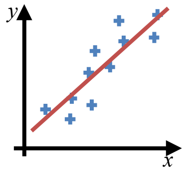
twice differentiable



Examples

Linear regression $y = w^T x + w_0$ $\quad x \in \Re^d, y \in \Re$

Absorb w_0 in w by adding
a dummy variable in x: $x_0 = 1$



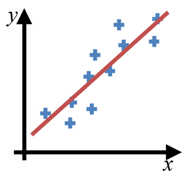
$$J(w) = \sum_i (w^T x_i - y_i)^2 + \lambda w^T w$$

convex, smooth, unconstrained

Examples

Linear regression $y = w^T x + w_0$ $x \in \Re^d, y \in \Re$

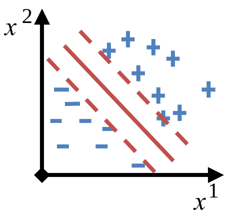
Absorb w_0 in w by adding a dummy variable in x: $x_0 = 1$



$$J(w) = \sum_i (w^T x_i - y_i)^2 + \lambda w^T w$$

convex, smooth, unconstrained

Linear SVM $y = \mathrm{sgn}(w^T x + w_0)$ $x \in \Re^d, y \in \{-1, 1\}$



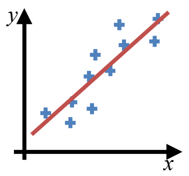
$$J(w) = \sum_i \max(0, 1 - y_i w^T x_i) + \lambda w^T w$$

convex, non-differentiable, unconstrained

Examples

Linear regression $y = w^T x + w_0$ $x \in \Re^d, y \in \Re$

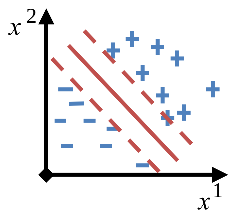
Absorb w_0 in w by adding a dummy variable in x: $x_0 = 1$



$$J(w) = \sum_i (w^T x_i - y_i)^2 + \lambda w^T w$$

convex, smooth, unconstrained

Linear SVM $y = \mathrm{sgn}(w^T x + w_0)$ $x \in \Re^d, y \in \{-1, 1\}$

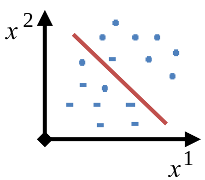


$$J(w) = \sum_i \max(0, 1 - y_i w^T x_i) + \lambda w^T w$$

convex, non-differentiable, unconstrained

Logistic Regression $p(y=1 \mid x) = \sigma(w^T x + w_0)$ $x \in \Re^d, y \in \{-1, 1\}$

$\sigma(x) = 1/(1+e^{-x})$ probabilistic



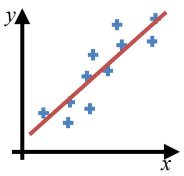
$$J(w) = -\sum_i \log(\sigma(y_i w^T x_i) + \lambda w^T w$$

convex, smooth, unconstrained

Examples

Linear regression $y = w^T x + w_0$ $\quad x \in \Re^d, y \in \Re$

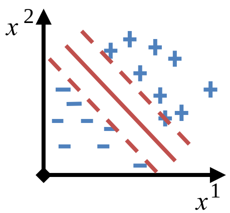
Absorb w_0 in w by adding a dummy variable in x: $x_0 = 1$



$$J(w) = \sum_i (w^T x_i - y_i)^2 + \lambda w^T w$$

convex, smooth, unconstrained

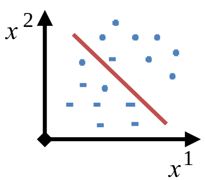
Linear SVM $y = \mathrm{sgn}(w^T x + w_0)$ $\quad x \in \Re^d, y \in \{-1, 1\}$



$$J(w) = \sum_i \max(0, 1 - y_i w^T x_i) + \lambda w^T w$$

convex, non-differentiable, unconstrained

Logistic Regression $p(y = 1 | x) = \sigma(w^T x + w_0)$ $\quad x \in \Re^d, y \in \{-1, 1\}$ $\quad \sigma(x) = 1/(1 + e^{-x})$ probabilistic



$$J(w) = -\sum_i \log(\sigma(y_i w^T x_i)) + \lambda w^T w$$

convex, smooth, unconstrained

Nonlinear (kernelized) versions: replace $w^T x$ with $\sum_i w^i k(x, x_i)$

- 2-norm regularizer changes to $w^T K w$ where K is the kernel matrix
- Same properties of the functions as their linear versions

Popular Optimization Methods

First Order Methods

- Use gradient of the function to iteratively find the local minimum
- Easy to compute and run but slow convergence
- Gradient (steepest) descent
- Coordinate descent
- Conjugate Gradient
- Stochastic Gradient Descent

Second Order Methods

- Use gradients and Hessian (curvature) iteratively
- Computationally more demanding but fast convergence
- Newton's method
- Quasi-Newton methods (BFGS and variants)
- Stochastic Quasi-Newton methods

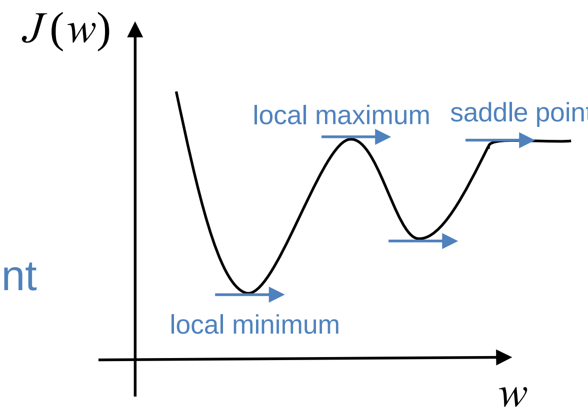
Other than line-search based methods: Trust-region

Conditions for Local Minimum

- If $\hat{w}$ is a local-minimizer then,
  - First-order necessary condition

Gradient $$\frac{\partial J(w)}{\partial w} = \nabla J(\hat{w}) = 0$$

stationary point

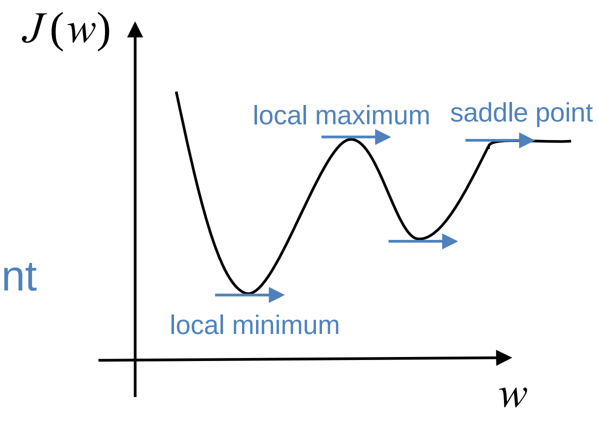


Conditions for Local Minimum

- If $\hat{w}$ is a local-minimizer then,
  - First-order necessary condition

Gradient $$\frac{\partial J(w)}{\partial w} = \nabla J(\hat{w}) = 0$$

stationary point



  - Second-order necessary condition

Hessian $$\frac{\partial^2 J(w)}{\partial w \partial w^T} = \nabla^2 J(\hat{w}) \text{ is PSD}$$

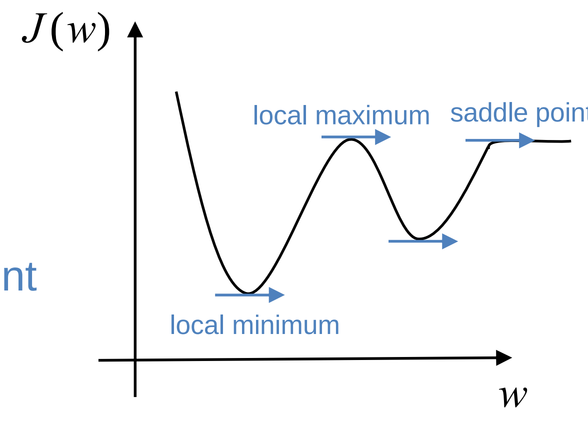
positive semi-definite

Conditions for Local Minimum

- If $\hat{w}$ is a local-minimizer then,
  - First-order necessary condition

Gradient $$\frac{\partial J(w)}{\partial w} = \nabla J(\hat{w}) = 0$$

stationary point



  - Second-order necessary condition

Hessian $$\frac{\partial^2 J(w)}{\partial w \partial w^T} = \nabla^2 J(\hat{w}) \text{ is PSD}$$

positive semi-definite

  - Sufficient conditions

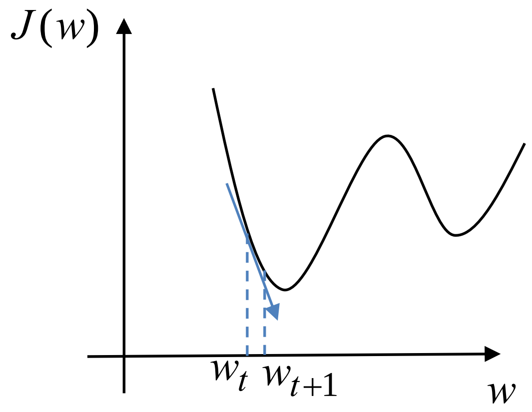
$$\nabla J(\hat{w}) = 0 \text{ and } \nabla^2 J(\hat{w}) \text{ is PD}$$ positive definite (strictly PD)

For smooth convex function → any stationary point is a global minimizer

Gradient Descent

Iteratively estimates new parameter values

$$w_{t+1} = w_t - \eta_t \nabla J(w_t)$$

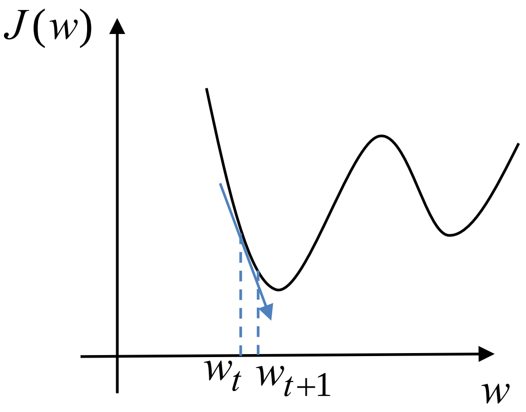


Gradient Descent

Iteratively estimates new parameter values

$$w_{t+1} = w_t - \eta_t \nabla J(w_t)$$

We want $J(w_{t+1}) \leq J(w_t)$



But this is not sufficient to reach local minima !

One has to make sure $\nabla J(w_t) \rightarrow 0$

Step-length must satisfy the Wolfe conditions of sufficient decrease

search direction

$$J(w_t + \eta_t p_t) \leq J(w_t) + c_1 \eta_t \nabla J(w_t) p_t \qquad 0 < c_1 < 1$$

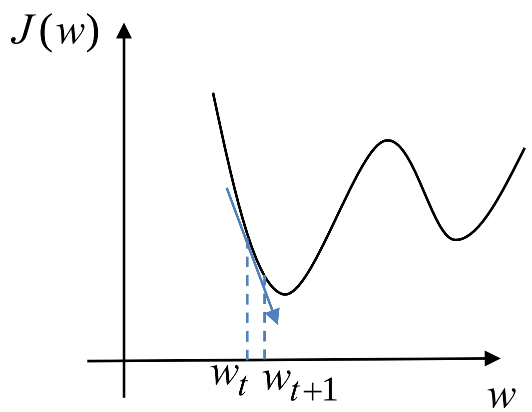
Gradient descent converges to local minimum if step size is sufficiently small !

Gradient Descent

Iteratively estimates new parameter values

$$w_{t+1} = w_t - \eta_t \nabla J(w_t)$$

We want $J(w_{t+1}) \leq J(w_t)$



But this is not sufficient to reach local minima !

One has to make sure $\nabla J(w_t) \rightarrow 0$

Step-length must satisfy the Wolfe conditions of sufficient decrease

search direction

$$J(w_t + \eta_t p_t) \leq J(w_t) + c_1 \eta_t \nabla J(w_t) p_t \qquad 0 < c_1 < 1$$

Gradient descent converges to local minimum if step size is sufficiently small !

Rate of Convergence: Linear

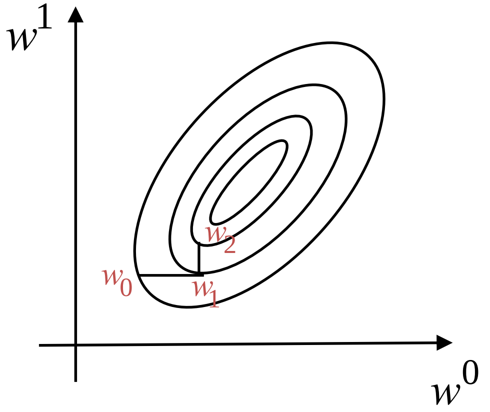
$$\frac{\|w_{t+1} - \hat{w}\|}{\|w_t - \hat{w}\|} \leq r$$

Painfully slow in practice, many heuristics are used (e.g., momentum)

Coordinate Descent

Updates one coordinate at a time keeping others fixed

$$w_{t+1}^{i} = w_{t}^{i} - \eta_t \nabla J(w_t^i)$$

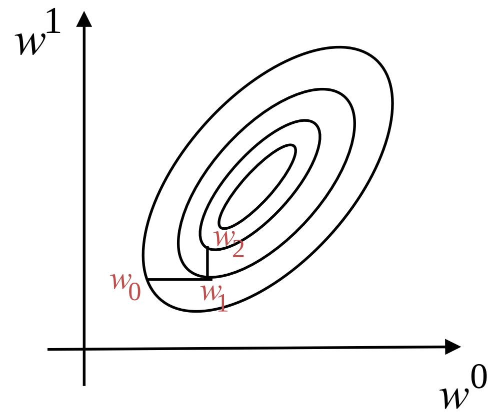


- Cycles through all the coordinates sequentially

Coordinate Descent

Updates one coordinate at a time keeping others fixed

$$w_{t+1}^{i} = w_{t}^{i} - \eta_t \nabla J(w_{t}^{i})$$



- Cycles through all the coordinates sequentially

Properties

- Very simple and easy to implement – reasonable when variables are loosely coupled
- Can be inefficient in practice (may take long time to converge)
- Convergence not guaranteed, in general

Block Coordinate Descent

- Update a small set of coordinates simultaneously
- Has been used successfully for training large SVMs

Conjugate Gradient

Parameters are searched along conjugate directions

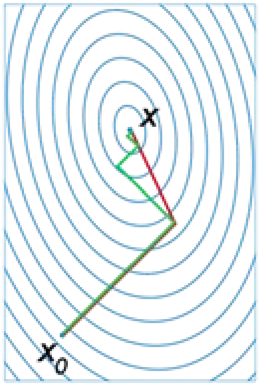
Conjugacy: Two vectors p_1 and p_2 are conjugate wrt H if

$$p_1^T H p_2 = 0$$

Starting with $p_0 = -\nabla J(w_0)$

$$w_{t+1} = w_t + \alpha_t p_t$$

green: GD
red: CG



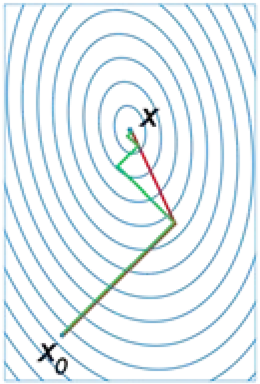
Conjugate Gradient

Parameters are searched along conjugate directions

Conjugacy: Two vectors p_1 and p_2 are conjugate wrt H if

$$p_1^T H p_2 = 0$$

green: GD
red: CG



Starting with $p_0 = -\nabla J(w_0)$

$$\boxed{w_{t+1} = w_t + \alpha_t p_t}$$

Successive conjugate directions as linear combination of gradient direction and previous conjugate direction

$$p_t = -\nabla J(w_t) + \beta_t p_{t-1} \qquad t = 1,2,\ldots.$$

Exact $\alpha_t = \dfrac{\nabla J(w_t)^T p_t}{p_t^T H_t p_t}$

But usually picked something that satisfies Wolfe's conditions !

$$\beta_t = \frac{\nabla J(w_t)^T (\nabla J(w_t) - \nabla J(w_{t-1}))}{p_{t-1}^T (\nabla J(w_t) - \nabla J(w_{t-1}))}$$

Hestenes-Stiefel form

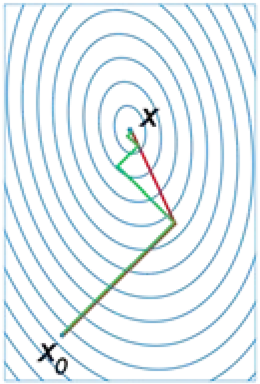
Conjugate Gradient

Parameters are searched along conjugate directions

Conjugacy: Two vectors p_1 and p_2 are conjugate wrt H if

$$p_1^T H p_2 = 0$$

green: GD
red: CG



Starting with $p_0 = -\nabla J(w_0)$

$$w_{t+1} = w_t + \alpha_t p_t$$

Successive conjugate directions as linear combination of gradient direction and previous conjugate direction

$$p_t = -\nabla J(w_t) + \beta_t p_{t-1} \qquad t = 1,2,....$$

Exact $\alpha_t = \frac{\nabla J(w_t)^T p_t}{p_t^T H_t p_t}$

But usually picked something that satisfies Wolfe's conditions !

$$\beta_t = \frac{\nabla J(w_t)^T (\nabla J(w_t) - \nabla J(w_{t-1}))}{p_{t-1}^T (\nabla J(w_t) - \nabla J(w_{t-1}))}$$

Hestenes-Stiefel form

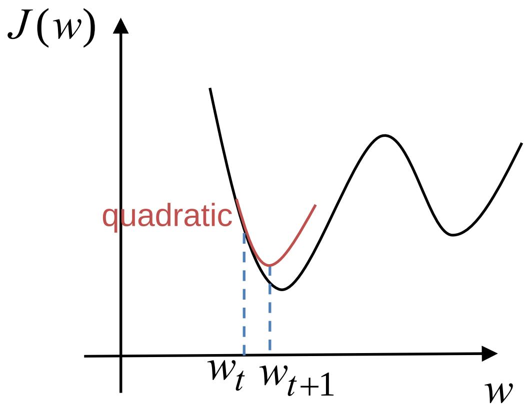
Properties

- For quadratic functions, guaranteed to return the optimal in at most d iterations
- Use preconditioning to increase the convergence rate: make condition number of Hessian as small as possible

Newton's Method

Approximates a function using second order Taylor expansion

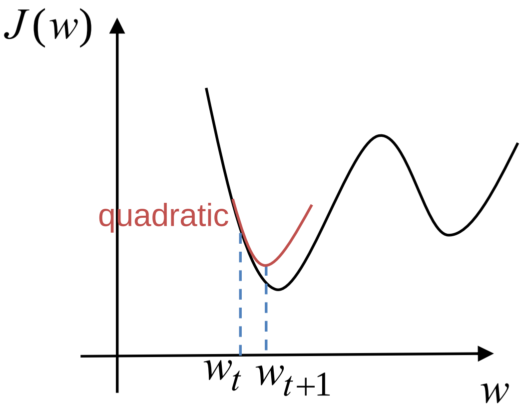
$$J(w_t + p) \approx J(w_t) + p^T \nabla J(w_t) + \frac{1}{2} p^T \nabla^2 J(w_t) p$$



Newton's Method

Approximates a function using second order Taylor expansion

$$J(w_t + p) \approx J(w_t) + p^T \nabla J(w_t) + \frac{1}{2} p^T \nabla^2 J(w_t) p$$



minimizing w.r.t. p

Hessian

$$p = \left(\nabla^2 J(w_t)\right)^{-1} \nabla J(w_t) = H_t^{-1} \nabla J(w_t)$$

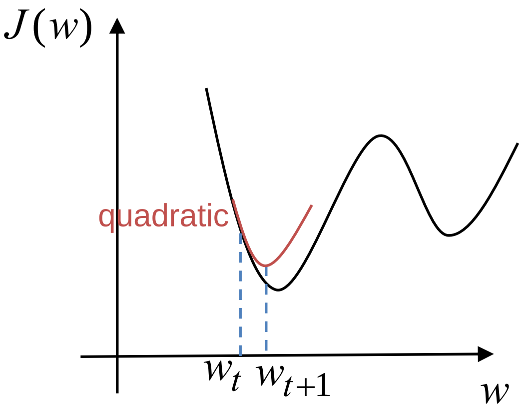
$$w_{t+1} = w_t - H_t^{-1} \nabla J(w_t)$$

- No explicit step-length parameter
- If Hessian is not positive definite, Newton step may be undefined or may even diverge
- For quadratic functions, converges in a single step !

Newton's Method

Approximates a function using second order Taylor expansion

$$J(w_t + p) \approx J(w_t) + p^T \nabla J(w_t) + \frac{1}{2} p^T \nabla^2 J(w_t) p$$



minimizing w.r.t. p

Hessian

$$p = \left(\nabla^2 J(w_t)\right)^{-1} \nabla J(w_t) = H_t^{-1} \nabla J(w_t)$$

$$w_{t+1} = w_t - H_t^{-1} \nabla J(w_t)$$

- No explicit step-length parameter
- If Hessian is not positive definite, Newton step may be undefined or may even diverge
- For quadratic functions, converges in a single step !

Rate of Convergence: Quadratic

$$\frac{\|w_{t+1} - \hat{w}\|}{\|w_t - \hat{w}\|^2} \leq k$$

Fast convergence but each iteration slow: $O(d^2)$ space and $O(d^3)$ time

Quasi-Newton Methods

Do not require computation of Hessian but still super-linear convergence

Key Idea: Approximate Hessian locally using change in gradients

Secant equation

Approx Hessian → $B_{t+1} s_t = y_t$

$$s_t = w_{t+1} - w_t, \quad y_t = \nabla J(w_{t+1}) - \nabla J(w_t)$$

Wolfe's conditions

$$s_t^T y_t > 0$$

$$B_t \succ 0$$

Quasi-Newton Methods

Do not require computation of Hessian but still super-linear convergence

Key Idea: Approximate Hessian locally using change in gradients

Secant equation

Approx Hessian → $B_{t+1} s_t = y_t$

$$s_t = w_{t+1} - w_t, \quad y_t = \nabla J(w_{t+1}) - \nabla J(w_t)$$

Wolfe's conditions

$$s_t^T y_t > 0$$

$$B_t \succ 0$$

- Additional conditions imposed on B, e.g., symmetry or diagonal
- Difference between successive B's is low-rank
- BFGS (Broyden-Fletcher-Goldfarb-Shanno): uses rank-2 (inverse) Hessian updates

if $\tilde{B}_t = B_t^{-1}$ $\quad \tilde{B}_{t+1} = \arg\min_{\tilde{B}} \left\| \tilde{B} - \tilde{B}_t \right\|_F$ subject to $\tilde{B}_{t+1} = \tilde{B}_{t+1}^T, \quad \tilde{B}_{t+1} y_t = s_t$

Quasi-Newton Methods

Do not require computation of Hessian but still super-linear convergence

Key Idea: Approximate Hessian locally using change in gradients

Secant equation

Approx Hessian → $B_{t+1} s_t = y_t$

$s_t = w_{t+1} - w_t, \quad y_t = \nabla J(w_{t+1}) - \nabla J(w_t)$

Wolfe's conditions

$s_t^T y_t > 0$

$B_t \succ 0$

- Additional conditions imposed on B, e.g., symmetry or diagonal
- Difference between successive B's is low-rank
- BFGS (Broyden-Fletcher-Goldfarb-Shanno): uses rank-2 (inverse) Hessian updates

if $\tilde{B}_t = B_t^{-1}$ $\quad \tilde{B}_{t+1} = \arg\min_{\tilde{B}} \left\| \tilde{B} - \tilde{B}_t \right\|_F$ subject to $\tilde{B}_{t+1} = \tilde{B}_{t+1}^T, \quad \tilde{B}_{t+1} y_t = s_t$

- Inverse of approx Hessian updated directly → $O(d^2)$ space and time

$$\tilde{B}_{t+1} = (I - \rho_t s_t y_t^T)\tilde{B}_t (I - \rho_t y_t s_t^T) + \rho_t s_t s_t^T \qquad \rho_t = 1/s_t^T y_t, \; \tilde{B}_0 = I$$

$O(d^2)$ cost per iteration, superlinear rate of convergence, self-correcting properties

Limited-Memory BFGS (L-BFGS)

Quasi-Newton methods produce dense Hessian approximations even when true Hessian is sparse → high storage cost for large-scale problems

Key Idea: Store approx Hessian using a few d-dim vectors from most recent iterations → Store at most m most recent pairs (s_t, y_t)

Limited-Memory BFGS (L-BFGS)

Quasi-Newton methods produce dense Hessian approximations even when true Hessian is sparse → high storage cost for large-scale problems

Key Idea: Store approx Hessian using a few d-dim vectors from most recent iterations → Store at most m most recent pairs (s_t, y_t)

$$w_{t+1} = w_t - \alpha_t \tilde{B}_t \nabla J(w_t)$$

- Iteratively estimate the product $\tilde{B}_t \nabla J(w_t)$ using most recent (s_t, y_t)
- Can be achieved efficiently in two loops in $O(md)$
- Different initialization for each inner loop possible, e.g.,

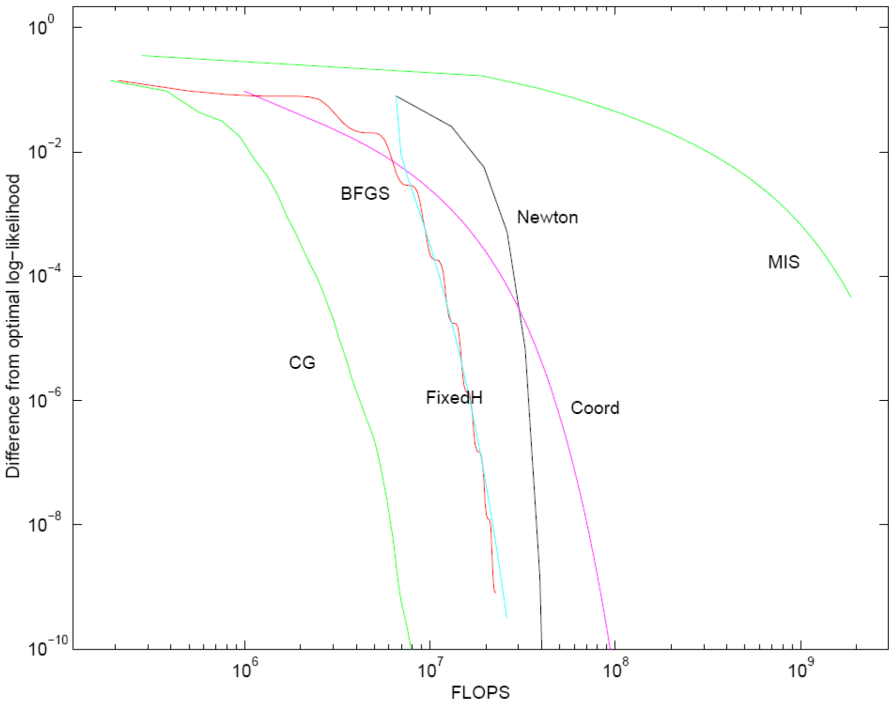
$$\tilde{B}_t^0 = \gamma_t I, \qquad \gamma_t = \frac{s_{t-1}^T y_{t-1}}{y_{t-1}^T y_{t-1}}$$

Similar to Conjugate-Gradient, rate of convergence linear instead of superlinear as in BFGS

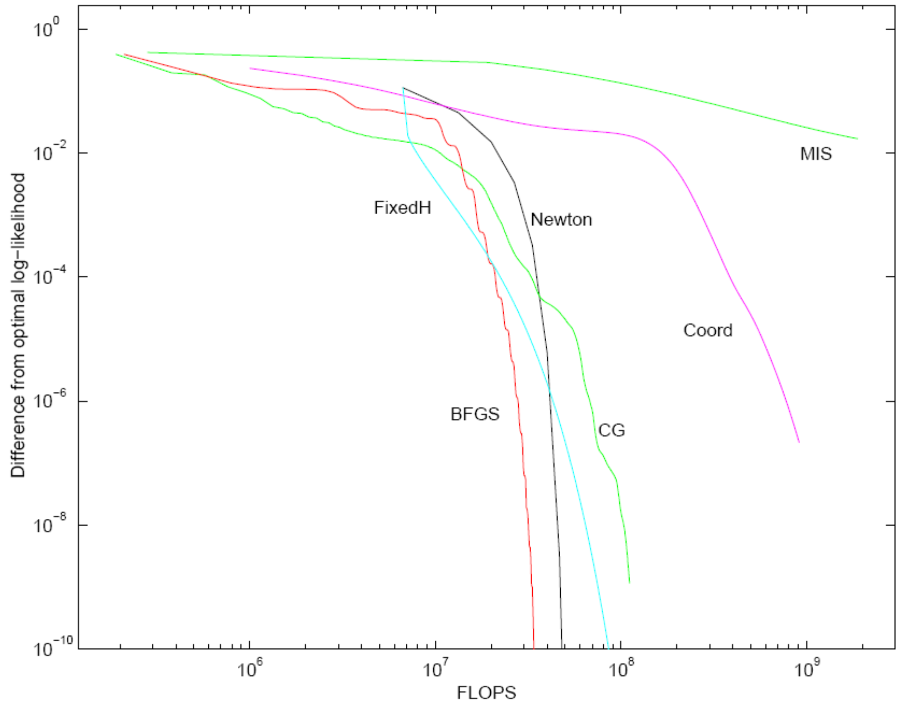
How about sparse, sampling-based or diagonal approximation of Hessian ?

Comparison: Logistic Regression

n=1500, d=100



independent features



correlated features

Hessian-based methods do better

Trade-off between number of iterations and cost per iteration !

Minka [5]

Batch vs Online

Most optimization functions use i.i.d. training data as,

Linear Regression $J(w) = \sum_i [(w^T x_i - y_i)^2 + \lambda w^T w]$

SVM $J(w) = \sum_i [\max(0, 1 - y_i w^T x_i) + \lambda w^T w]$

Logistic Regression $J(w) = -\sum_i [\log(\sigma(y_i w^T x_i) + \lambda w^T w]$

Batch methods

- compute gradients using all the training data
- each iteration needs to use full batch of data → slow for large datasets

Online (Stochastic) Methods

- use gradients from a small subset of data, usually just a single point
- do not decrease the function value monotonically
- typically have worse convergence properties than batch methods
- quite appropriate for large-scale applications where data is usually redundant
- can converge even before seeing all the data once → very fast
- popular conjecture: may also avoid bad local minima

Stochastic Gradient Descent (SGD)

At each iteration, use a small training subset to estimate the gradient

$$w_{t+1} = w_t - \eta_t \nabla J(w_t, x_t)$$

single point or a small subset of training data

Stochastic Gradient Descent (SGD)

At each iteration, use a small training subset to estimate the gradient

$$w_{t+1} = w_t - \eta_t \nabla J(w_t, x_t)$$

single point or a small subset of training data

Converges to optimal value $\hat{w}$ if,

parameters may move arbitrary distances $\sum_t \eta_t = \infty$

step size decreases fast enough $\sum_t \eta_t^2 < \infty$

Step-length $\eta_t = \frac{\tau}{\tau + t}\eta_0$ $\tau, \eta_0 > 0$ are tuning parameters

Properties

- $O(d)$ space and time per iteration instead of $O(nd)$ of gradient descent
- Outperforms batch gradient methods on large datasets
- Slow convergence on ill-conditioned problems
- Stochastic Meta-Descent: adjust a different gain for each parameter

Experiment - SGD

SVM classification task

d = 47K (~80 nonzero), n = ~800K

Model	Algorithm	Training Time	Objective	Test Error
Hinge loss, $\lambda = 10^{-4}$ *See [21,22].*	SVMLight	23,642 secs	0.2275	6.02%
	SVMPerf	66 secs	0.2278	6.03%
	SGD	**1.4 secs**	0.2275	6.02%
Logistic loss, $\lambda = 10^{-5}$ *See [23].*	LibLinear ($\rho = 10^{-2}$)	30 secs	0.18907	5.68%
	LibLinear ($\rho = 10^{-3}$)	44 secs	0.18890	5.70%
	SGD	**2.3 secs**	0.18893	5.66%

Bottou & Bousquet [8]

SGD for Sparse Data

Suppose training set vectors $\{x_i \in \Re^d\}_{i=1,\ldots,n}$ are sparse

- Only a small fraction (s) of d elements are non-zero $s << 1$
- Common for many applications: text, images, biological data,…

SGD for Sparse Data

Suppose training set vectors $\{x_i \in \Re^d\}_{i=1,\ldots,n}$ are sparse

- Only a small fraction (s) of d elements are non-zero $s << 1$
- Common for many applications: text, images, biological data,…

Objective function $J(w) = \sum_{i=1}^{n} [L(y_i w^T x_i) + \lambda R(w)]$

$$\nabla J(w_t, x_t) = \underbrace{\nabla L(y_t w_t^T x_t) y_t x_t}_{\text{sparse}} + \underbrace{\lambda \nabla R(w_t)}_{\text{dense}}$$

} dense vector update $O(d)$ per iteration

How to make sparse updates, i.e., $O(sd)$ per iteration?

SGD for Sparse Data

Suppose training set vectors $\{x_i \in \Re^d\}_{i=1,\ldots,n}$ are sparse

- Only a small fraction (s) of d elements are non-zero $s << 1$
- Common for many applications: text, images, biological data,…

Objective function $J(w) = \sum_{i=1}^{n} [L(y_i w^T x_i) + \lambda R(w)]$

$$\nabla J(w_t, x_t) = \underbrace{\nabla L(y_t w_t^T x_t) y_t x_t}_{\text{sparse}} + \underbrace{\lambda \nabla R(w_t)}_{\text{dense}}$$

} dense vector update $O(d)$ per iteration

How to make sparse updates, i.e., $O(sd)$ per iteration?

Break updates into two parts

1. Update using gradients of $L(.)$ alone ignoring the regularizer

$$w_{t+1} = w_t - \eta_t \nabla L(w_t, x_t)$$

2. Every kth iteration, adjust w using gradients of regularizer

$$w_{t+1} \leftarrow w_{t+1} - k\eta_t \nabla R(w_{t+1})$$

Perceptron

Originally proposed to learn a linear binary classifier $y = \{-1, 1\}$

$$f(x) = \mathrm{sgn}(w^T x)$$

Update Rule

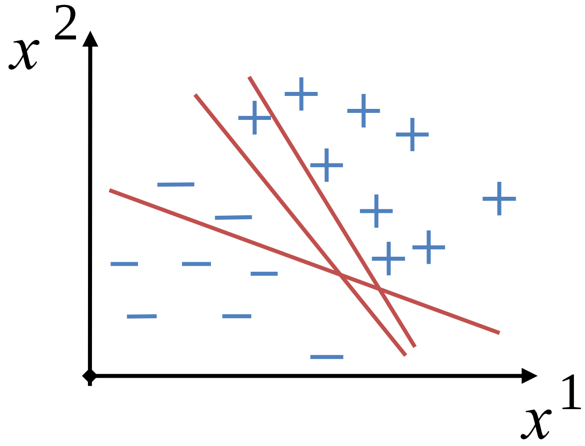
$$w_{t+1} = \begin{cases} w_t + y\,x_t & \text{if } x \text{ is misclassified, i.e., } y \neq f(x) \\ w_t & \text{otherwise} \end{cases}$$

Perceptron

Originally proposed to learn a linear binary classifier $y = \{-1, 1\}$

$$f(x) = \text{sgn}(w^T x)$$

Update Rule $$w_{t+1} = \begin{cases} w_t + y\, x_t & \text{if x is misclassified, i.e., } y \neq f(x) \\ w_t & \text{otherwise} \end{cases}$$



Properties

- Stochastic gradient descent on a non-differentiable loss function
- Guaranteed to converge if data is linearly separable
- What happens for inseparable data ?

will not converge but oscillate !

- Use of heuristics such as voting with various parameter vectors

Natural Gradient Descent

Incorporate Riemannian Metric into stochastic descent

$$G_t = E_x[\nabla J(w_t, x_t) \nabla J(w_t, x_t)^T]$$

$$w_{t+1} = w_t - \eta_t \tilde{G}_t^{-1} \nabla J(w_t, x_t)$$

$$\eta_t = \frac{\tau}{\tau + t} \eta_0$$

Natural Gradient Descent

Incorporate Riemannian Metric into stochastic descent

$$G_t = E_x[\nabla J(w_t, x_t)\nabla J(w_t, x_t)^T]$$

$$w_{t+1} = w_t - \eta_t \tilde{G}_t^{-1} \nabla J(w_t, x_t)$$

$$\eta_t = \frac{\tau}{\tau + t}\eta_0$$

Metric update $$\tilde{G}_{t+1} = (t-1)/t\ \tilde{G}_t + (1/t)\nabla J(w_t, x_t)\nabla J(w_t, x_t)^T$$

assuming Hessian to be constant

Properties

- Update matrix inverse directly → $O(d^2)$ space and time per iteration
- More stable – has good theoretical properties
- Still expensive for large-scale problems

Online-BFGS

Change of gradients estimated using a subset of training data

$\tilde{B}_{t+1} y_t = s_t$

$s_t = w_{t+1} - w_t,$

$y_t = \nabla J(w_{t+1}) - \nabla J(w_t)$

$$w_{t+1} = w_t - \eta_t \tilde{B}_t \nabla J(w_t, x_t)$$

stochastic approximation of inverse Hessian

Wolfe's conditions

$s_t^T y_t > 0$

$B_t \succ 0$

Online-BFGS

Change of gradients estimated using a subset of training data

$\tilde{B}_{t+1} y_t = s_t$
$s_t = w_{t+1} - w_t,$
$y_t = \nabla J(w_{t+1}) - \nabla J(w_t)$

$$w_{t+1} = w_t - \eta_t \tilde{B}_t \nabla J(w_t, x_t)$$

stochastic approximation of inverse Hessian

Wolfe's conditions
$s_t^T y_t > 0$
$B_t \succ 0$

1. To maintain positive curvature

Modify $y_t = \nabla J(w_{t+1}, x_t) - \nabla J(w_t, x_t) + \lambda s_t \quad \lambda \geq 0$ Compute on the same sample !

2. Initialize $\tilde{B}$ using a small coefficient to restrict initial parameter update

$$\tilde{B}_0 = \varepsilon I \qquad \varepsilon \approx 10^{-10}$$

3. Modify the update of $\tilde{B}_{t+1}$

$\rho_t = 1/s_t^T y_t$

$$\tilde{B}_{t+1} = (I - \rho_t s_t y_t^T) \tilde{B}_t (I - \rho_t s_t y_t^T) + c \rho_t s_t s_t^T \qquad 0 < c \leq 1 \quad \eta_t = \eta_t / c$$

Online-BFGS

Change of gradients estimated using a subset of training data

$\tilde{B}_{t+1} y_t = s_t$

$s_t = w_{t+1} - w_t,$

$y_t = \nabla J(w_{t+1}) - \nabla J(w_t)$

$$w_{t+1} = w_t - \eta_t \tilde{B}_t \nabla J(w_t, x_t)$$

stochastic approximation of inverse Hessian

Wolfe's conditions

$s_t^T y_t > 0$

$B_t \succ 0$

1. To maintain positive curvature

Modify $y_t = \nabla J(w_{t+1}, x_t) - \nabla J(w_t, x_t) + \lambda s_t \quad \lambda \geq 0$ Compute on the same sample !

2. Initialize $\tilde{B}$ using a small coefficient to restrict initial parameter update

$$\tilde{B}_0 = \varepsilon I \qquad \varepsilon \approx 10^{-10}$$

3. Modify the update of $\tilde{B}_{t+1}$

$\rho_t = 1 / s_t^T y_t$

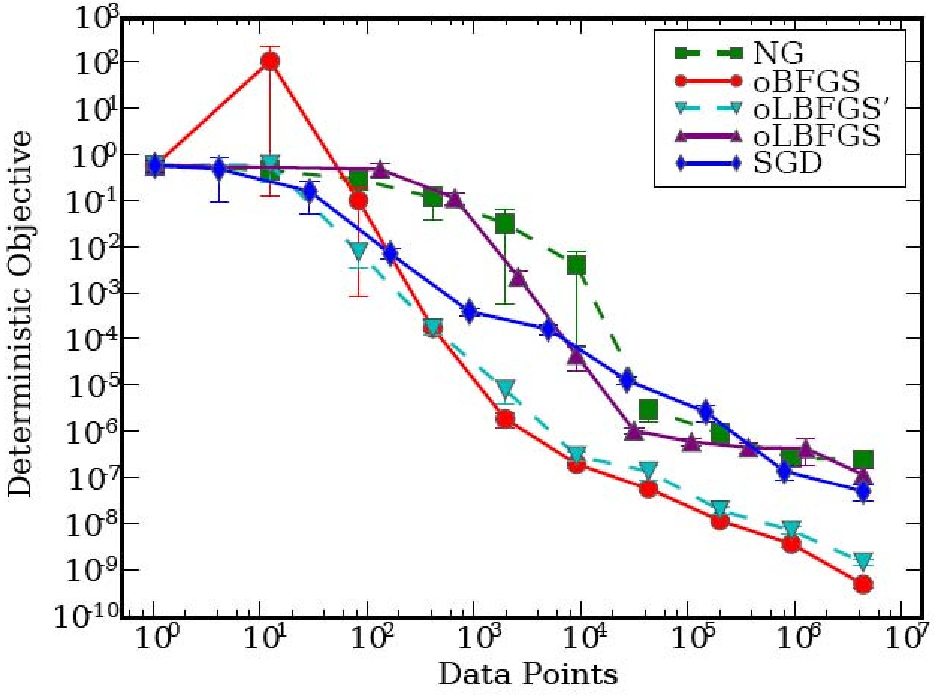
$$\tilde{B}_{t+1} = (I - \rho_t s_t y_t^T) \tilde{B}_t (I - \rho_t s_t y_t^T) + c \rho_t s_t s_t^T \qquad 0 < c \leq 1 \quad \eta_t = \eta_t / c$$

- Test of convergence: If last k stochastic gradients have been below a threshold
- Online version of L-BFGS possible → $O(md)$ cost instead of $O(d^2)$

Experiments

Synthetic quadratic function

parameters d = 5, n = ~1M

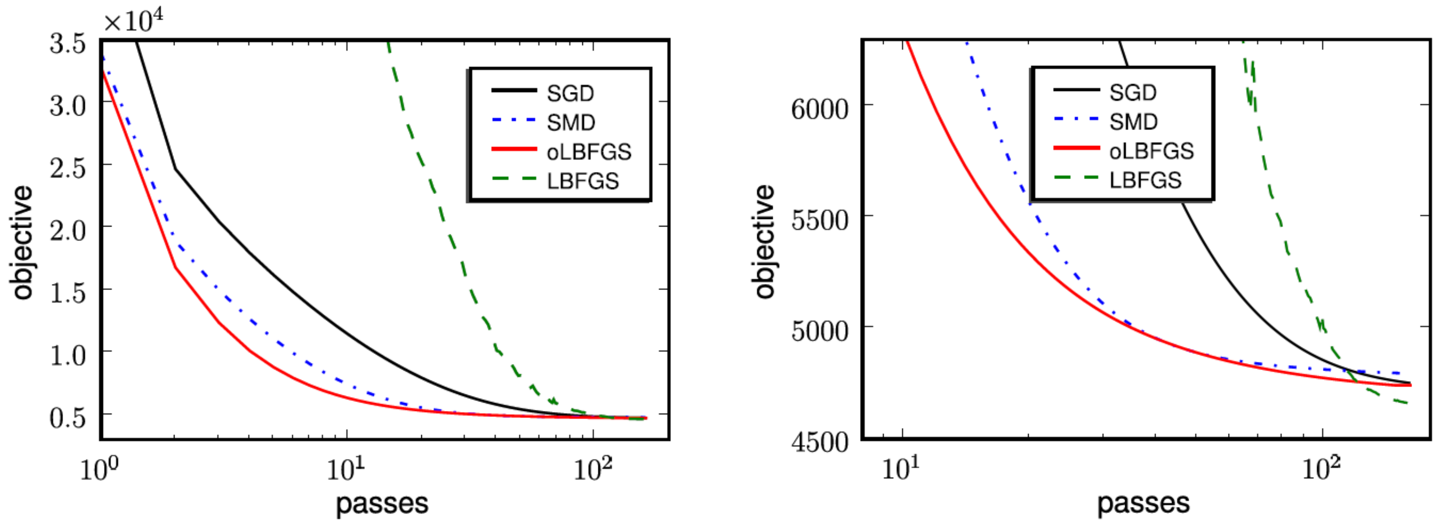


Schraudolph et al. [7]

Experiments

On Conditional Random Field (CRF) applied to text chunking

parameters d = 100K, n = ~9K



Schraudolph et al. [7]

Stochastic Gradient Descent (Quasi-Newton)

Approximate inverse Hessian by a diagonal matrix

Inspired by online-BFGS

$\tilde{B}_{t+1} y_t = s_t$

$s_t = w_{t+1} - w_t,$

$y_t = \nabla J(w_{t+1}) - \nabla J(w_t)$

$$w_{t+1} - w_t \approx \tilde{B}_{t+1}(\nabla J(w_{t+1}, x_t) - \nabla J(w_t, x_t))$$

diagonal matrix

Wolfe's conditions

$s_t^T y_t > 0$

$B_t \succ 0$

Stochastic Gradient Descent (Quasi-Newton)

Approximate inverse Hessian by a diagonal matrix

Inspired by online-BFGS

$$\tilde{B}_{t+1} y_t = s_t$$
$$s_t = w_{t+1} - w_t,$$
$$y_t = \nabla J(w_{t+1}) - \nabla J(w_t)$$

$$w_{t+1} - w_t \approx \tilde{B}_{t+1}(\nabla J(w_{t+1}, x_t) - \nabla J(w_t, x_t))$$

diagonal matrix

Wolfe's conditions

$$s_t^T y_t > 0$$
$$B_t \succ 0$$

$$[w_{t+1} - w_t]_i \approx \tilde{B}_{ii}[\nabla J(w_{t+1}, x_t) - \nabla J(w_t, x_t)]_i$$

Stochastic Gradient Descent (Quasi-Newton)

Approximate inverse Hessian by a diagonal matrix

Inspired by online-BFGS

$\tilde{B}_{t+1} y_t = s_t$
$s_t = w_{t+1} - w_t,$
$y_t = \nabla J(w_{t+1}) - \nabla J(w_t)$

$$w_{t+1} - w_t \approx \tilde{B}_{t+1}(\nabla J(w_{t+1}, x_t) - \nabla J(w_t, x_t))$$

diagonal matrix

Wolfe's conditions

$s_t^T y_t > 0$
$B_t \succ 0$

$$[w_{t+1} - w_t]_i \approx \tilde{B}_{ii}[\nabla J(w_{t+1}, x_t) - \nabla J(w_t, x_t)]_i$$

In Practice,

- Update matrix entries only every k iterations
- Do a leaky average of B to get stable updates

$$\tilde{B}_{ii} \leftarrow \tilde{B}_{ii}[1 + k\tilde{B}_{ii} r_i]^{-1}$$

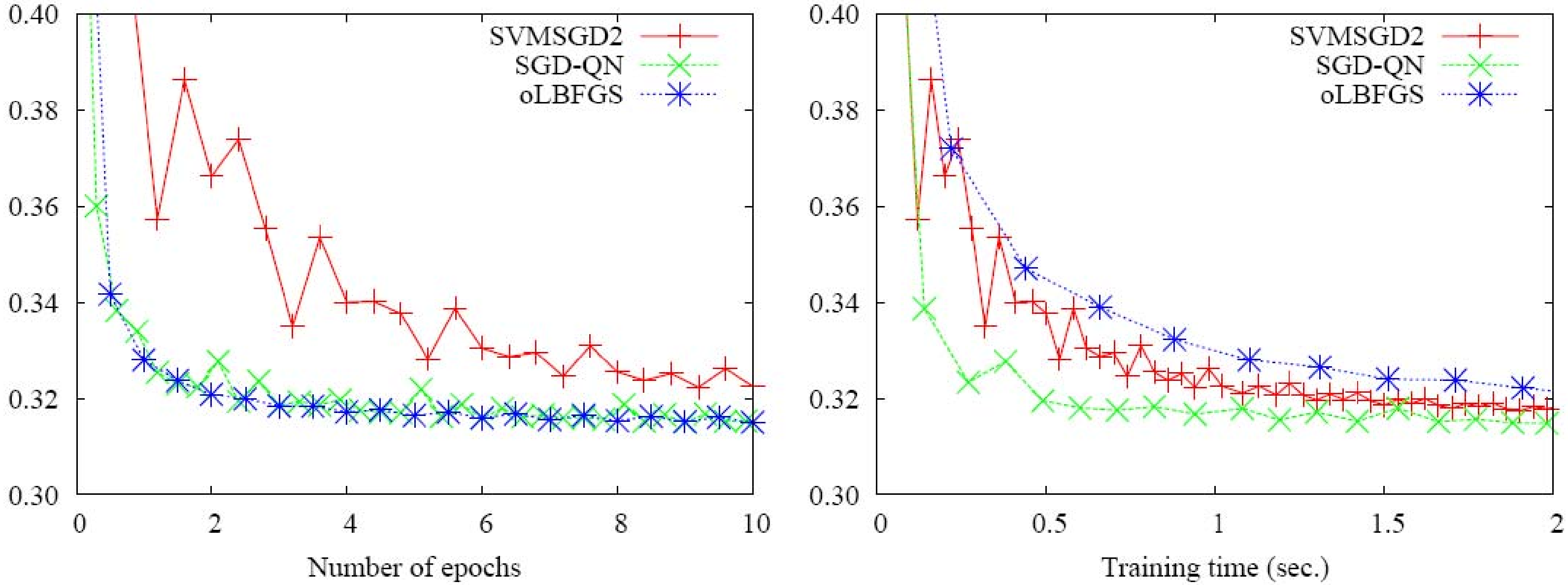
$r_i = [\nabla J(w_{t+1}, x_t) - \nabla J(w_{t-k}, x_t)]_i / [w_{t+1} - w_t]_i \quad r_i \leftarrow \max\{\lambda, \min\{100\lambda, r_i\}\}$

- Has a flavor of partially 'fixed-Hessian'

Stochastic Gradient Descent (Quasi-Newton)

Alpha dataset of Pascal Challenge (2008)

parameters d = 500 (dense), n = ~100K



	ALPHA	RCV1
SGD	0.13	36.8
SVMSGD2	0.10	0.20
SGD-QN	0.21	0.37

sparse SGD → SVMSGD2

RCV1: d = 47K, s = 0.0016

Bordes et al. [9]

References

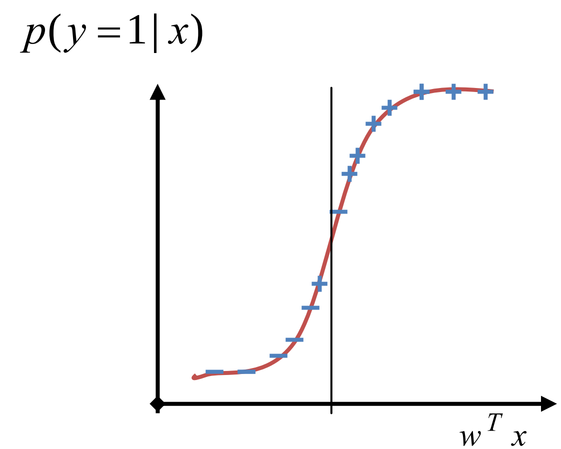
1. H. E. Robbins and S. Monro, “A stochastic approximation method,” Annals of Mathematical Statistics, 1951.
2. J. Nocedal and S. J. Wright, Numerical Optimization. Springer Series in Operations Research, 1999.
3. N. Schraudolph, “Local gain adaptation in stochastic gradient descent,” Int. conf in Artifical Neural Networks, 1999.
4. S. Amari, H. Park, K. Fukumizu, “Adaptive method of realizing natural gradient learning for multi-layer perceptrons,” Neural Computation, 2000.
5. Tom Minka, “A comparison of numerical optimizers for logistic regression”, 2003. http://research.microsoft.com/en-us/um/people/minka/papers/logreg/
6. L. Bottou and Y. Lecun, “Online learning for very large datasets,” Applied Stochastic Models in Business and Industry, 2005.
7. N. Schraudolph, J. Yu and S. Gunter, “A stochastic quasi-Newton method for online convex optimization,” AISTATS, 2007.
8. L. Bottou and O. Bousquet: Learning Using Large Datasets, *Mining Massive DataSets for Security*, NATO ASI Workshop Series, IOS Press, Amsterdam, 2008.
9. A. Bordes, L. Bottou and P. Gallinari, “SGD-QN: Careful Quasi-Newton Stochastic Gradient Descent, JMLR, 2009.
10. A. Bordes, L. Bottou, P. Gallinari, J. Chang, S. A. Smith, “Erratum: SGD-QN is less careful than expected,” 2010. http://jmlr.csail.mit.edu/papers/v11/bordes10a.html
11. J. Yu, S. Vishwanathan, S. Günter, and N. N. Schraudolph. A Quasi-Newton Approach to Nonsmooth Convex Optimization Problems in Machine Learning. *Journal of Machine Learning Research*, 11:1145–1200, 2010.

Logistic Regression

Given: A labeled training set, $\{x_i, y_i\}_{i=1 \ldots n} \quad x_i \in \Re^d, y_i \in \{-1, 1\}$

Goal: Learn a predictive function $p(y=1 | x) = \sigma(w^T x)$ Absorb w_0 in w

$$p(y_i | x_i) = \sigma(y_i w^T x_i)$$



(negative) log-posterior

$$L(w) = \underbrace{\sum_i \log(1 + \exp(-y_i w^T x_i))}_{\text{log-likelihood}} + \underbrace{\lambda w^T w}_{\text{log-prior}}$$

Convex problem, Newton method:

$$w_{t+1} = (XAX^T + \lambda I)^{-1} XAz$$

$n \sim O(100M), d \sim O(100K)$

$O(nd^2)$ multiplication

$O(d^3)$ inversion

$O(nd)$ First-order methods