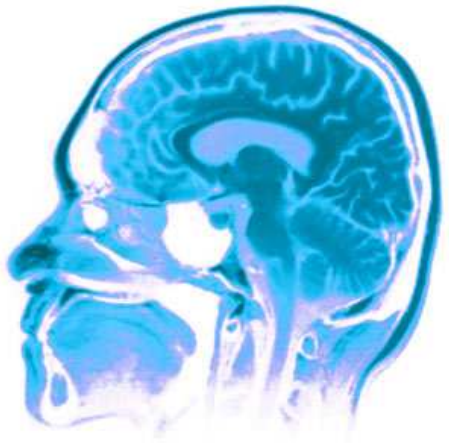
CPSC540



Second Order Optimization Methods

Nando de Freitas

2011

KPM Book Sections: *3.2.1, 3.2.2, 3.4, 3.5, 3.6*

Second-order optimization

- First-order methods do not use the Hessian, and do not model the curvature of the space. Hence they can be slow to converge.
- **Second-order** optimization methods make use of the Hessian, in one form or another, and converge much faster.
- However, storing the full Hessian takes $O(D^2)$ space, and inverting it can take $O(D^3)$ time, so the overall computation time for second-order methods may be higher than for first-order methods, depending on the cost of evaluating the objective function and its gradient.

Newton's algorithm

- The most basic second-order optimization algorithm is **Newton's algorithm**, which consists of updates of the form

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mathbf{H}_K^{-1}\mathbf{g}_k$$

- This algorithm can be derived as follows. Consider making a second-order Taylor series approximation of $f(\boldsymbol{\theta})$ around $\boldsymbol{\theta}_k$:

$$f_{quad}(\boldsymbol{\theta}) = f_k + \mathbf{g}_k^T(\boldsymbol{\theta} - \boldsymbol{\theta}_k) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_k)^T\mathbf{H}_k(\boldsymbol{\theta} - \boldsymbol{\theta}_k)$$

Let us rewrite this as

$$f_{quad}(\boldsymbol{\theta}) = \boldsymbol{\theta}^T\mathbf{A}\boldsymbol{\theta} + \mathbf{b}^T\boldsymbol{\theta} + c$$

where

$$\mathbf{A} = \frac{1}{2}\mathbf{H}_k, \quad \mathbf{b} = \mathbf{g}_k - \mathbf{H}_k\boldsymbol{\theta}_k, \quad c = f_k - \mathbf{g}_k^T\boldsymbol{\theta}_k + \frac{1}{2}\boldsymbol{\theta}_k^T\mathbf{H}_k\boldsymbol{\theta}_k$$

$$f_{quad}(\boldsymbol{\theta}) = f_k + \mathbf{g}_k^T(\boldsymbol{\theta} - \boldsymbol{\theta}_k) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_k)^T \mathbf{H}_k(\boldsymbol{\theta} - \boldsymbol{\theta}_k)$$

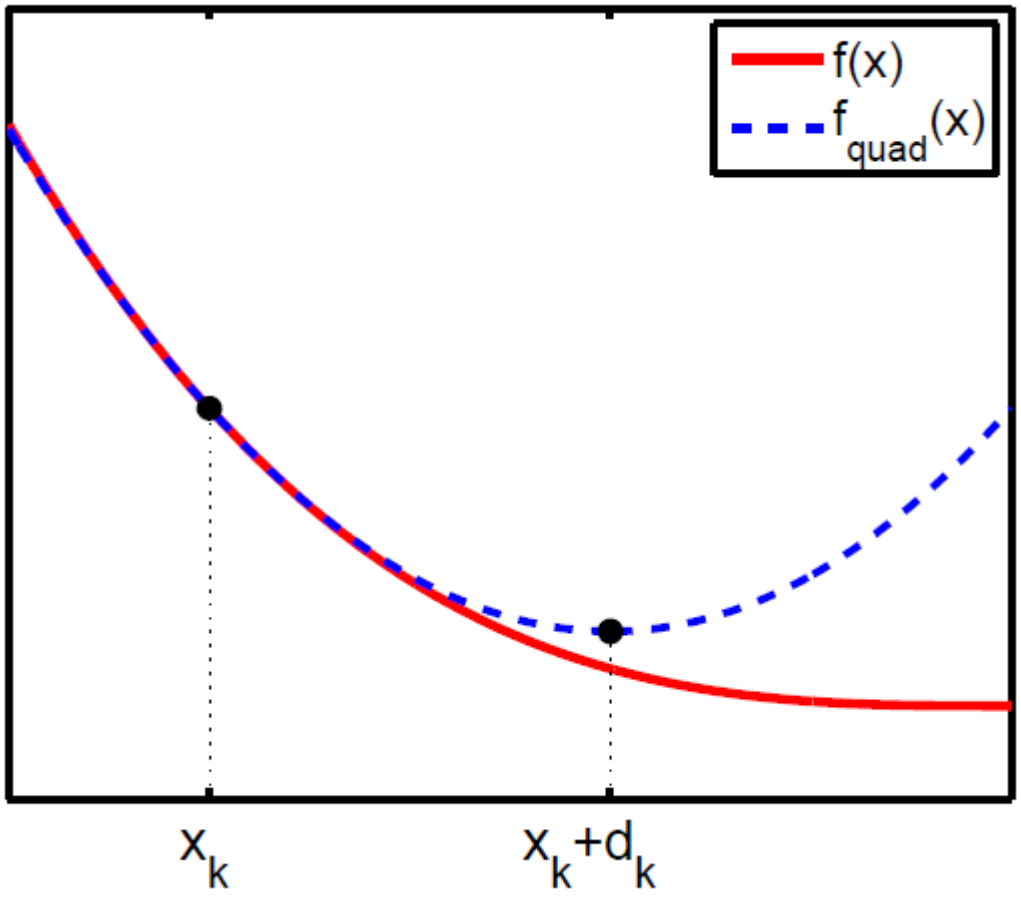
$$f_{quad}(\boldsymbol{\theta}) = \boldsymbol{\theta}^T \mathbf{A}\boldsymbol{\theta} + \mathbf{b}^T\boldsymbol{\theta} + c$$

where

$$\mathbf{A} = \frac{1}{2}\mathbf{H}_k, \quad \mathbf{b} = \mathbf{g}_k - \mathbf{H}_k\boldsymbol{\theta}_k, \quad c = f_k - \mathbf{g}_k^T\boldsymbol{\theta}_k + \frac{1}{2}\boldsymbol{\theta}_k^T\mathbf{H}_k\boldsymbol{\theta}_k$$

The minimum of f_{quad} is at?

Newton's method



Newton's algorithm

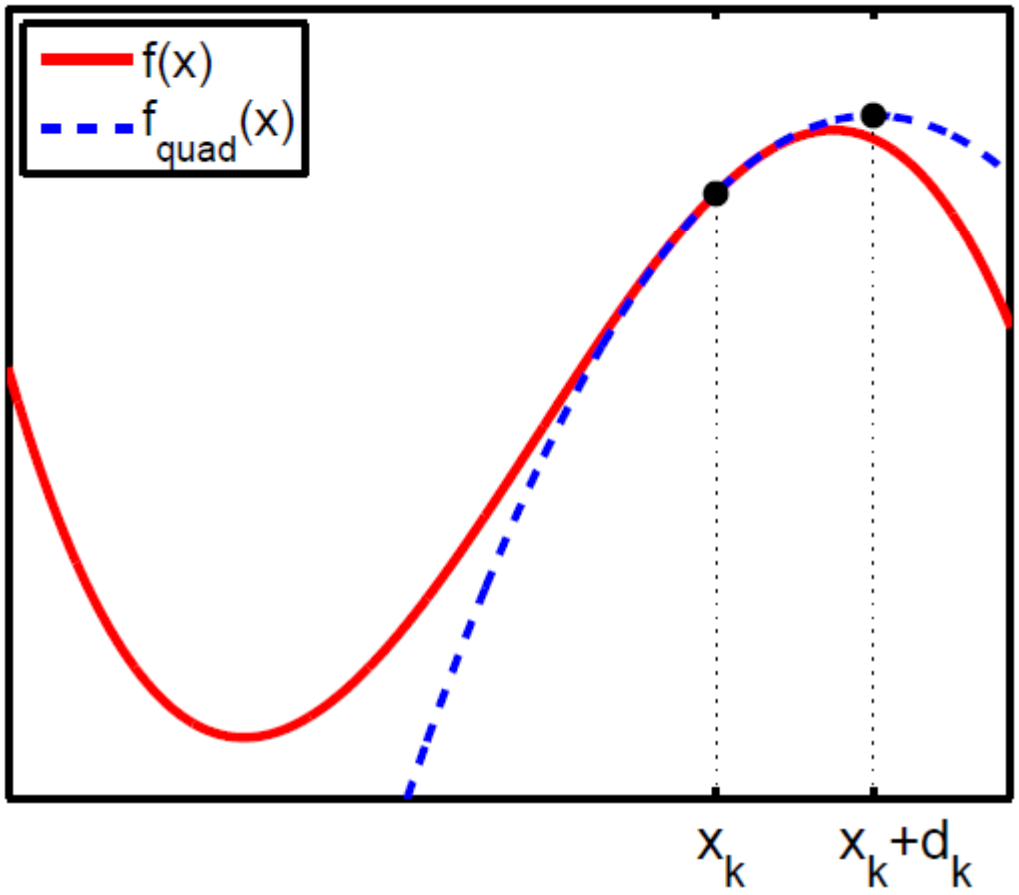
- Rather than computing $\mathbf{d}_k = -\mathbf{H}_k^{-1}\mathbf{g}_k$ directly, we can solve the linear system of equations $\mathbf{H}_k\mathbf{d}_k = -\mathbf{g}_k$ for $\mathbf{d}_k$.
- One efficient way to do this, especially if $\mathbf{H}$ is sparse, is to use a conjugate gradient method. This combination is called **Newton-CG**, and is widely used.

Newton's method

1 Initialize $\boldsymbol{\theta}_0$
2 **for** $k = 1, 2, \ldots$ *until convergence* **do**
3 Evaluate $\mathbf{g}_k = \nabla f(\boldsymbol{\theta}_k)$
4 Evaluate $\mathbf{H}_k = \nabla^2 f(\boldsymbol{\theta}_k)$
5 Solve $\mathbf{H}_k \mathbf{d}_k = -\mathbf{g}_k$ for $\mathbf{d}_k$
6 Use line search to find stepsize η_k along $\mathbf{d}_k$
7 $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \eta_k \mathbf{d}_k$

Newton's algorithm

- If the objective function is not convex, then $\mathbf{H}_k$ may not be positive definite, so $\mathbf{d}_k = -\mathbf{H}_k^{-1}\mathbf{g}_k$ may not be a descent direction.



Newton's algorithm

- One simple solution is to revert to steepest descent, $\mathbf{d}_k = -\mathbf{g}_k$. The **Levenberg Marquardt** algorithm is an adaptive way to blend between Newton steps and steepest descent steps.
- This method is widely used when solving nonlinear least squares problems.
- An alternative approach is to add $\delta_k \mathbf{I}$ to $\mathbf{H}_k$ for some $\delta_k > 0$ to make it positive definite.
- If we are using CG, we can simply truncate the CG iterations as soon as negative curvature is detected; this is called **truncated Newton**.

Iteratively reweighted least squares (IRLS)

- For the MLE for binary logistic regression, recall that the gradient and Hessian of the NLL are given by

$$\begin{aligned}
\mathbf{g}_k &= \mathbf{X}^T(\boldsymbol{\mu}_k - \mathbf{y}) \\
\mathbf{H}_k &= \mathbf{X}^T\mathbf{S}_k\mathbf{X} \\
\mathbf{S}_k &:= \text{diag}(\mu_{1k}(1-\mu_{1k}), \ldots, \mu_{Nk}(1-\mu_{Nk})) \\
\mu_{ik} &= \text{sigm}(\mathbf{w}_k^T\mathbf{x}_i)
\end{aligned}$$

- The Newton update at iteration $k+1$ for this model is as follows (using $\eta_k = 1$, since the Hessian is exact):

$$\begin{aligned}
\mathbf{w}_{k+1} &= \mathbf{w}_k - \mathbf{H}^{-1}\mathbf{g}_k \\
&= \mathbf{w}_k + (\mathbf{X}^T\mathbf{S}_k\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{y} - \boldsymbol{\mu}_k) \\
&= (\mathbf{X}^T\mathbf{S}_k\mathbf{X})^{-1}\left[(\mathbf{X}^T\mathbf{S}_k\mathbf{X})\mathbf{w}_k + \mathbf{X}^T(\mathbf{y} - \boldsymbol{\mu}_k)\right] \\
&= (\mathbf{X}^T\mathbf{S}_k\mathbf{X})^{-1}\mathbf{X}^T\left[\mathbf{S}_k\mathbf{X}\mathbf{w}_k + \mathbf{y} - \boldsymbol{\mu}_k\right]
\end{aligned}$$

Iteratively reweighted least squares (IRLS)

- We can rewrite this as a weighted least squares problem

$$\mathbf{w}_{k+1} \quad = \quad (\mathbf{X}^T \mathbf{S}_k \mathbf{X})^{-1} \mathbf{X}^T \mathbf{S}_k \mathbf{z}_k$$

  where we have defined the targets (outputs) as

$$\mathbf{z}_k := \mathbf{X}\mathbf{w}_k + \mathbf{S}_k^{-1}(\mathbf{y} - \boldsymbol{\mu}_k)$$

- Since $\mathbf{S}_k$ is a diagonal matrix, we can rewrite the targets in component form (for each case $i = 1 : N$) as

$$z_{ki} = \mathbf{w}_k^T \mathbf{x}_i + \frac{y_i - \mu_{ki}}{\mu_{ki}(1 - \mu_{ki})}$$

IRLS Algorithm

$\mathbf{w} = \mathbf{0}_D$

$w_0 = \log(\overline{y}/(1-\overline{y}))$

repeat

  $\eta_i = w_0 + \mathbf{w}^T \mathbf{x}_i$

  $\mu_i = \mathrm{sigm}(\eta_i)$

  $z_i = \eta_i + \frac{y_i - \mu_i}{\mu_i(1-\mu_i)}$

  $s_i = \mu_i(1-\mu_i)$

  $\mathbf{S} = \mathrm{diag}(s_{1:N})$

  $\mathbf{w} = (\mathbf{X}^T \mathbf{S} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{S} \mathbf{z}$

until *converged*

Quasi-Newton (variable metric) methods

- The Newton direction $\mathbf{d}_k = -\mathbf{H}_k^{-1}\mathbf{g}_k$ has two drawbacks. The first is that it is not necessarily a descent direction unless $\mathbf{H}_k$ is positive definite.
- The second is that it may be too expensive to compute $\mathbf{H}$ explicitly.
- **Quasi-Newton** methods iteratively build up an approximation to the Hessian using information gleaned from the gradient vector at each step.

BFGS

- **BFGS** (named after its inventors, Broyden, Fletcher, Goldfarb and Shanno), updates the approximation to the Hessian $\mathbf{B}_k \approx \mathbf{H}_k$ as follows:

$$\begin{aligned} \mathbf{B}_{k+1} &= \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\mathbf{B}_k^T \mathbf{s}_k^T \mathbf{s}_k \mathbf{B}_k}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k} \\ \mathbf{s}_k &= \boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1} \\ \mathbf{y}_k &= \mathbf{g}_k - \mathbf{g}_{k-1} \end{aligned}$$

  This is a rank-two update to the matrix, and ensures that the matrix remains positive definite. Why?

- Alternatively, BFGS can iteratively update an approximation to the inverse Hessian, $\mathbf{C}_k \approx \mathbf{H}_k^{-1}$, as follows:

$$\mathbf{C}_{k+1} = (\mathbf{I} - \frac{\mathbf{s}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}) \mathbf{C}_k (\mathbf{I} - \frac{\mathbf{y}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}) + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}$$

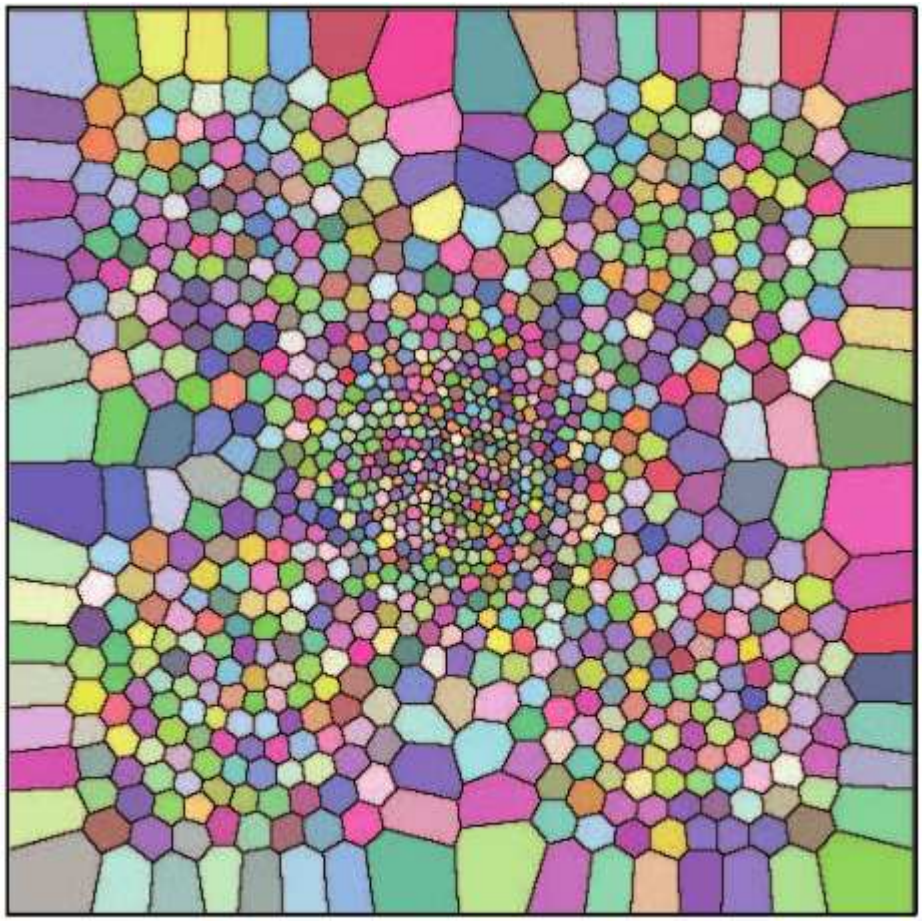
- Another similar method is called **DFP** (named after Davidon, Fletcher, and Powell).

L-BFGS

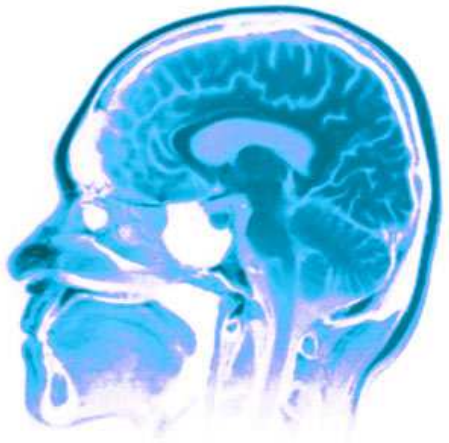
- Since storing the Hessian takes $O(D^2)$ space, for very large problems, one can use **limited memory BFGS**, or **LBFGS**, where a low rank approximation to $\mathbf{H}_k$ or $\mathbf{H}_k^{-1}$ is stored implicitly. In particular, the product $\mathbf{H}_k^{-1}\mathbf{g}_k$ can be obtained by performing a sequence of inner products with $\mathbf{s}_k$ and $\mathbf{y}_k$, using only the m most recent $(\mathbf{s}_k, \mathbf{y}_k)$ pairs, and ignoring older information. Typically $m \sim 20$ suffices for good performance.
- LBFGS is probably the method of choice for most unconstrained optimization problems that arise in machine learning (e.g., fitting logistic regression, CRFs, neural nets, etc.).

Centroidal Voronoi Tessellation

G. Rong, Y. Liu, W. Wang, X. Yin, X. Gu and X. Guo



Next class



Constrained Optimization and Duality

Nando de Freitas

2011

KPM Book Sections: *11.2, 11.3 and 30.4*