

CS6301: Optimization in Machine Learning

Lectures 11 and 12: Newton, Quasi Newton and Second Order Methods

Rishabh Iyer

Department of Computer Science
University of Texas, Dallas

<https://sites.google.com/view/cs-6301-optml/home>

February 24th, 2020



Miscellaneous

- Assignment 2 will be out tomorrow. Please check eLearning.
- The Assignment will be due on March 7th.
- Borrowed some of the Slides in this Lecture from Aarti Singh and Pradeep Ravikumar's courses.



Summary of Algorithms so Far

First Order Methods

- Gradient Descent
- Projected Gradient Descent (whenever the Projection step is easy)
- Proximal Gradient Descent (if non-smooth function is simple – i.e. proximal operator of that function can be computed efficiently.)
- Accelerated Gradient versions in the unconstrained case, projected case and proximal case.



Outline of this Lecture

- Newton's Algorithm
- Newton vs Quasi Newton Methods
- Barzilai-Borwein GD
- BFGS
- Conjugate Gradient Family
- LBFGS
- Understanding how these Algorithms perform in practice!



Newton Methods

- Recall unconstrained convex minimization:

$$\min_{x \in \text{dom}(f)} f(x)$$



Newton Methods

- Recall unconstrained convex minimization:

$$\min_{x \in \text{dom}(f)} f(x)$$

- Newtons Method:

$$x_{t+1} = x_t - \nabla^2 f(x_t)^{-1} \nabla f(x_t)$$



Newton Methods

- Recall unconstrained convex minimization:

$$\min_{x \in \text{dom}(f)} f(x)$$

- Newtons Method:

$$x_{t+1} = x_t - \nabla^2 f(x_t)^{-1} \nabla f(x_t)$$

- Requires the convex function to be twice differentiable.



Newton Methods

- Recall unconstrained convex minimization:

$$\min_{x \in \text{dom}(f)} f(x)$$

- Newton's Method:

$$x_{t+1} = x_t - \nabla^2 f(x_t)^{-1} \nabla f(x_t)$$

- Requires the convex function to be twice differentiable.
- Recall GD was:

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t)$$



Newton vs Gradient Descent

- Recall Gradient Descent step could be seen as solving the optimization problem:

$$x_{t+1} = \operatorname{argmin}_x f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2\alpha_t} \|x - x_t\|^2$$



Newton vs Gradient Descent

- Recall Gradient Descent step could be seen as solving the optimization problem:

$$x_{t+1} = \operatorname{argmin}_x f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2\alpha_t} \|x - x_t\|^2$$

- Newton step can be seen as solving:

$$x_{t+1} = \operatorname{argmin}_x f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2} (x - x_t)^T \nabla^2 f(x_t) (x - x_t)$$



Newton vs Gradient Descent

- Recall Gradient Descent step could be seen as solving the optimization problem:

$$x_{t+1} = \operatorname{argmin}_x f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2\alpha_t} \|x - x_t\|^2$$

- Newton step can be seen as solving:

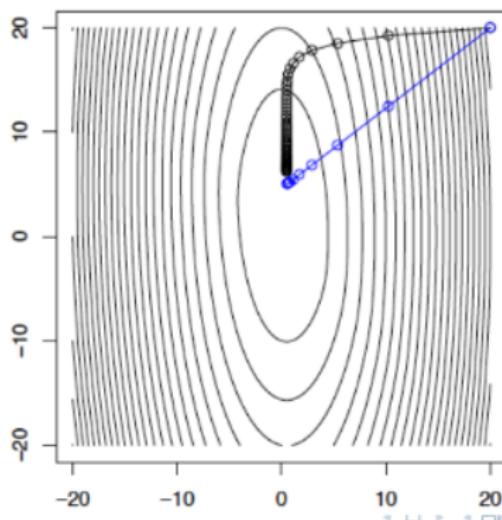
$$x_{t+1} = \operatorname{argmin}_x f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2} (x - x_t)^T \nabla^2 f(x_t) (x - x_t)$$

- No need to step size in Newton.



Newton vs Gradient Descent

For $f(x) = (10x_1^2 + x_2^2)/2 + 5 \log(1 + e^{-x_1 - x_2})$, compare gradient descent (black) to Newton's method (blue), where both take steps of roughly same length



DALLAS

Newton's method for finding the roots - I

Goal: $\phi : \mathbb{R} \rightarrow \mathbb{R}$

$$\phi(x^*) = 0$$

$$x^* = ?$$

Linear Approximation (1st order Taylor approx):

$$\underbrace{\phi(x + \Delta x)}_{\Phi(x^*) = 0} = \phi(x) + \phi'(x)\Delta x + o(|\Delta x|)$$

Therefore,

$$0 \approx \phi(x) + \phi'(x)\Delta x$$

$$x^* - x = \Delta x = -\frac{\phi(x)}{\phi'(x)}$$

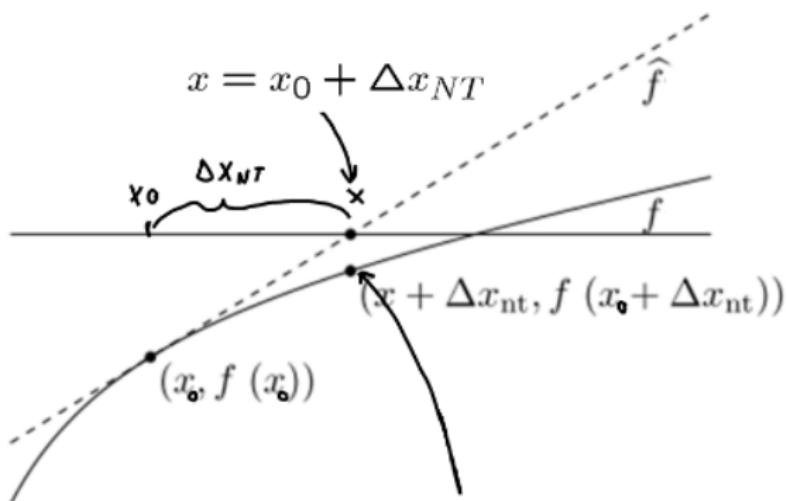
$$x_{k+1} = x_k - \frac{\phi(x)}{\phi'(x)}$$



Newton's method for finding the roots - II

Goal: finding a root

$$\hat{f}(x) = f(x_0) + f'(x_0)(x - x_0)$$



In the next step we will linearize here in x



Newton's method for finding the roots - Multivariate Functions

This can be generalized to multivariate functions

$$F : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$0_m = F(x^*) = F(x + \Delta x) = F(x) + \underbrace{\nabla F(x)}_{\mathbb{R}^{m \times n}} \underbrace{\Delta x}_{\mathbb{R}^n} + o(|\Delta x|)$$

NEGLECT

Therefore,

$$0_m = F(x) + \nabla F(x) \Delta x$$

$$\Delta x = -[\nabla F(x)]^{-1} F(x)$$

[Pseudo inverse if there is no inverse]

$$\Delta x = x_{k+1} - x_k, \text{ and thus}$$

$$x_{k+1} = x_k - [\nabla F(x_k)]^{-1} F(x_k)$$

$\mathbb{R}^n \quad \mathbb{R}^n \quad \mathbb{R}^{n \times m} \quad \mathbb{R}^m$



Newton's method for finding the roots \Rightarrow Optimization Algorithm

Newton's method for solving the optimization problem:

$$\min_x f(x)$$

is the same as Newton's method for finding the root of

$$\nabla f(x) = 0$$

History: The work of Newton (1685) and Raphson (1690) originally focused on finding roots for Polynomials. Simpson (1740) applied this idea to general nonlinear equations and optimization.



Local Convergence

- Under suitable conditions and starting close enough to the global minimum, Newton's method will reach ϵ close to the minimum in $\log \log(1/\epsilon)$ iterations!



- Under suitable conditions and starting close enough to the global minimum, Newton's method will reach ϵ close to the minimum in $\log \log(1/\epsilon)$ iterations!
- This is faster than anything we have seen so far (even strongly convex and smooth functions!)



Local Convergence

- Under suitable conditions and starting close enough to the global minimum, Newton's method will reach ϵ close to the minimum in $\log \log(1/\epsilon)$ iterations!
- This is faster than anything we have seen so far (even strongly convex and smooth functions!)
- However this holds only if we are close to the minima (hence local convergence). Global convergence from any starting point is unknown for Newton's method.



Local Convergence

- Under suitable conditions and starting close enough to the global minimum, Newton's method will reach ϵ close to the minimum in $\log \log(1/\epsilon)$ iterations!
- This is faster than anything we have seen so far (even strongly convex and smooth functions!)
- However this holds only if we are close to the minima (hence local convergence). Global convergence from any starting point is unknown for Newton's method.
- Local convergence is characterized by bounded inverse Hessians: There exists μ s.t. $||\nabla^2 f(x)^{-1}|| \leq \frac{1}{\mu}$ and Lipschitz continuous Hessians.



Local Convergence Theorem

Theorem

Let f be convex with a unique global minimum x^* . Suppose there is a ball X with center x^* such that:

- ① There exists a real number μ s.t $||\nabla^2 f(x)^{-1}|| \leq \frac{1}{\mu}$
- ② There exists a real number B s.t.

$$||\nabla^2 f(x) - \nabla^2 f(y)|| \leq B ||x - y||$$

Then for $x_t \in X$ and x_{t+1} resulting from the Newton step, we have

$$||x_{t+1} - x^*|| \leq \frac{B}{2\mu} ||x_t - x^*||^2$$

Local Convergence Theorem

- If we assume that

$$\|x_0 - x_*\| \leq \frac{\mu}{B}$$

and given the assumptions of the local convergence:



Local Convergence Theorem

- If we assume that

$$\|x_0 - x_*\| \leq \frac{\mu}{B}$$

and given the assumptions of the local convergence:

- The Newton method yields:

$$\|x_T - x^*\| \leq \frac{\mu}{B} \left(\frac{1}{2}\right)^{2T-1}$$



Local Convergence Theorem

- If we assume that

$$\|x_0 - x_*\| \leq \frac{\mu}{B}$$

and given the assumptions of the local convergence:

- The Newton method yields:

$$\|x_T - x^*\| \leq \frac{\mu}{B} \left(\frac{1}{2}\right)^{2^T-1}$$

- This is called superlinear convergence (convergen rate of $\log \log(1/\epsilon)$ iterations)!



Local Convergence Theorem

- If we assume that

$$\|x_0 - x_*\| \leq \frac{\mu}{B}$$

and given the assumptions of the local convergence:

- The Newton method yields:

$$\|x_T - x^*\| \leq \frac{\mu}{B} \left(\frac{1}{2}\right)^{2^T-1}$$

- This is called superlinear convergence (convergen rate of $\log \log(1/\epsilon)$ iterations)!
- Intuitive reason: Once the Hessian is bounded, the function is almost quadratic, which ensures Newton's algorithm converges very fast!



Local vs Global Convergence

- What about global convergence, i.e. if we start from any x_0 ?



Local vs Global Convergence

- What about global convergence, i.e. if we start from any x_0 ?
- Global convergence of Newton was not known until recently (Sai Praneeth Karimireddy, Sebastian U. Stich, Martin Jaggi, Arxiv 2018).



Local vs Global Convergence

- What about global convergence, i.e. if we start from any x_0 ?
- Global convergence of Newton was not known until recently (Sai Praneeth Karimireddy, Sebastian U. Stich, Martin Jaggi, Arxiv 2018).
- They only show linear global convergence (recall we had superlinear local convergence above).



Local vs Global Convergence

- What about global convergence, i.e. if we start from any x_0 ?
- Global convergence of Newton was not known until recently (Sai Praneeth Karimireddy, Sebastian U. Stich, Martin Jaggi, Arxiv 2018).
- They only show linear global convergence (recall we had superlinear local convergence above).
- They also show it under the assumption that the Hessians are *stable*.



Pros and Cons of Newton's Method

- Pros: Much faster than anything we have seen so far!



Pros and Cons of Newton's Method

- Pros: Much faster than anything we have seen so far!
- Pros: Even in practice, converges in very few iterations



Pros and Cons of Newton's Method

- Pros: Much faster than anything we have seen so far!
- Pros: Even in practice, converges in very few iterations
- Pros: Affine Invariance and Robustness



Pros and Cons of Newton's Method

- Pros: Much faster than anything we have seen so far!
- Pros: Even in practice, converges in very few iterations
- Pros: Affine Invariance and Robustness
- Cons: Requires computing the Hessian and inverting or solving a linear system (since we can solve $H_t(\Delta x) = -\nabla f(x_t)$ for the next step). Both are roughly $O(d^3)$!



Pros and Cons of Newton's Method

- Pros: Much faster than anything we have seen so far!
- Pros: Even in practice, converges in very few iterations
- Pros: Affine Invariance and Robustness
- Cons: Requires computing the Hessian and inverting or solving a linear system (since we can solve $H_t(\Delta x) = -\nabla f(x_t)$ for the next step). Both are roughly $O(d^3)$!
- In many cases, d (number of features) is large and each step can be prohibitive!



Affine Invariance of Newtons method

Important property Newton's method: **affine invariance**.

Assume $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is twice differentiable and $A \in \mathbb{R}^{n \times n}$ is nonsingular. Let $g(y) := f(Ay)$.

Newton step for g starting from y is

$$y^+ = y - (\nabla^2 g(y))^{-1} \nabla g(y).$$

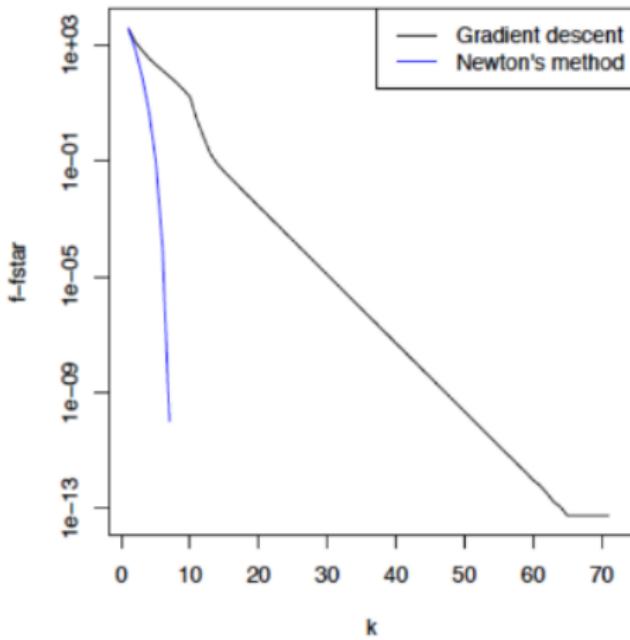
It turns out that the Newton step for f starting from $x = Ay$ is $x^+ = Ay^+$.

Therefore progress is independent of problem scaling. By contrast, this is **not true** of gradient descent.



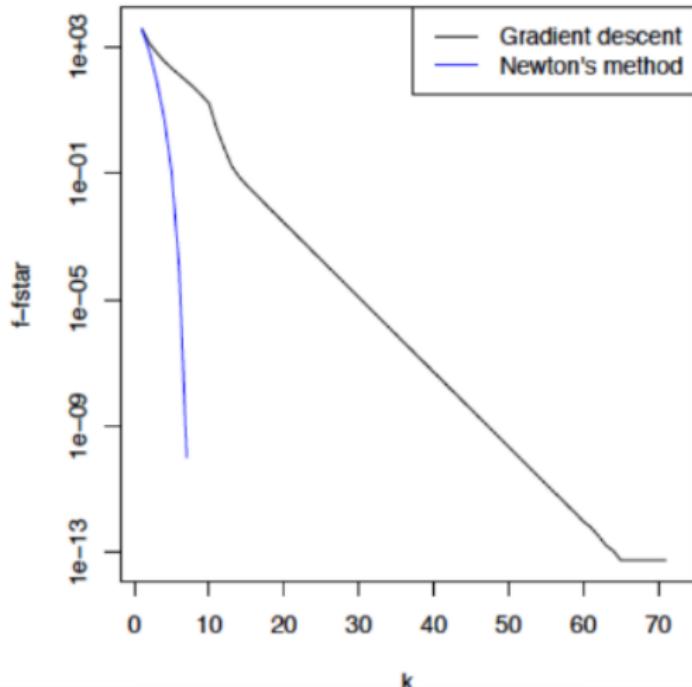
Newton vs Gradient: Empirically

Logistic Regression example with $n = 500, p = 100$. Compare GD with Newton's method.



Newton vs Gradient: Empirically

Same as earlier, just x-axis being total time instead of number of iterations
(since iteration complexity is $O(d^3)$ for Newton and $O(d)$ for GD!)



Secant Method

- Lets start with 1 dimension.



Secant Method

- Lets start with 1 dimension.
- Approximate the double derivative

$$f''(x_t) \approx H_t = \frac{f'(x_t) - f'(x_{t-1})}{x_t - x_{t-1}}$$



Secant Method

- Lets start with 1 dimension.
- Approximate the double derivative

$$f''(x_t) \approx H_t = \frac{f'(x_t) - f'(x_{t-1})}{x_t - x_{t-1}}$$

- This implies $f'(x_t) - f'(x_{t-1}) = H_t(x_t - x_{t-1})$



Secant Method

- Lets start with 1 dimension.
- Approximate the double derivative

$$f''(x_t) \approx H_t = \frac{f'(x_t) - f'(x_{t-1})}{x_t - x_{t-1}}$$

- This implies $f'(x_t) - f'(x_{t-1}) = H_t(x_t - x_{t-1})$
- Secant method: $x_{t+1} = x_t - H_t^{-1}f'(x_t)$



Secant Method

- Lets start with 1 dimension.
- Approximate the double derivative

$$f''(x_t) \approx H_t = \frac{f'(x_t) - f'(x_{t-1})}{x_t - x_{t-1}}$$

- This implies $f'(x_t) - f'(x_{t-1}) = H_t(x_t - x_{t-1})$
- Secant method: $x_{t+1} = x_t - H_t^{-1}f'(x_t)$
- In higher dimensions, the Matrix H_t must satisfy
 $\nabla f(x_t) - \nabla f(x_{t-1}) = H_t(x_t - x_{t-1})$



Secant Method

- Lets start with 1 dimension.
- Approximate the double derivative

$$f''(x_t) \approx H_t = \frac{f'(x_t) - f'(x_{t-1})}{x_t - x_{t-1}}$$

- This implies $f'(x_t) - f'(x_{t-1}) = H_t(x_t - x_{t-1})$
- Secant method: $x_{t+1} = x_t - H_t^{-1}f'(x_t)$
- In higher dimensions, the Matrix H_t must satisfy
 $\nabla f(x_t) - \nabla f(x_{t-1}) = H_t(x_t - x_{t-1})$
- This is called the secant conditions. When $d > 1$, this is an under-determined linear system and can infinitely many symmetric solutions.



History of Quasi Newton Methods

- In the mid 1950s, William Davidon was a mathematician/physicist at Argonne National Lab



History of Quasi Newton Methods

- In the mid 1950s, William Davidon was a mathematician/physicist at Argonne National Lab
- He was using coordinate descent on an optimization problem and his computer kept crashing before finishing



History of Quasi Newton Methods

- In the mid 1950s, William Davidon was a mathematician/physicist at Argonne National Lab
- He was using coordinate descent on an optimization problem and his computer kept crashing before finishing
- He figured out a way to accelerate the computation, leading to the first quasi-Newton method (soon Fletcher and Powell followed up on his work)



History of Quasi Newton Methods

- In the mid 1950s, William Davidon was a mathematician/physicist at Argonne National Lab
- He was using coordinate descent on an optimization problem and his computer kept crashing before finishing
- He figured out a way to accelerate the computation, leading to the first quasi-Newton method (soon Fletcher and Powell followed up on his work)
- Although Davidon's contribution was a major breakthrough in optimization, his original paper was rejected (for dubious notation and no convergence analysis)



History of Quasi Newton Methods

- In the mid 1950s, William Davidon was a mathematician/physicist at Argonne National Lab
- He was using coordinate descent on an optimization problem and his computer kept crashing before finishing
- He figured out a way to accelerate the computation, leading to the first quasi-Newton method (soon Fletcher and Powell followed up on his work)
- Although Davidon's contribution was a major breakthrough in optimization, his original paper was rejected (for dubious notation and no convergence analysis)
- In 1991, after more than 30 years, his paper was published in the first issue of the SIAM Journal on Optimization



History of Quasi Newton Methods

- In the mid 1950s, William Davidon was a mathematician/physicist at Argonne National Lab
- He was using coordinate descent on an optimization problem and his computer kept crashing before finishing
- He figured out a way to accelerate the computation, leading to the first quasi-Newton method (soon Fletcher and Powell followed up on his work)
- Although Davidon's contribution was a major breakthrough in optimization, his original paper was rejected (for dubious notation and no convergence analysis)
- In 1991, after more than 30 years, his paper was published in the first issue of the SIAM Journal on Optimization
- In addition to his remarkable work in optimization, Davidon was a peace activist (see the book "The Burglary")



History of Quasi Newton Methods

- Davidon's original paper:



History of Quasi Newton Methods

- Davidon's original paper:
- William C Davidon, Variable Metric Method for Minimization, Technical Report, ANL, 1959



History of Quasi Newton Methods

- Davidon's original paper:
- William C Davidon, Variable Metric Method for Minimization, Technical Report, ANL, 1959
- Davidon's 1991 publication:



History of Quasi Newton Methods

- Davidon's original paper:
- William C Davidon, Variable Metric Method for Minimization, Technical Report, ANL, 1959
- Davidon's 1991 publication:
- William C Davidon, Variable Metric Method for Minimization, SIAM Journal of Optimization, 1(1), 1-17, 1991 (**This is one of the best journals on optimization**).



Quasi newton Template

Let $x_0 \in \mathbb{R}^n$, $B_0 \succ 0$. For $k = 1, 2, \dots$, repeat:

- Define s_{k-1} as the descent step.



Quasi newton Template

Let $x_0 \in \mathbb{R}^n$, $B_0 \succ 0$. For $k = 1, 2, \dots$, repeat:

- Define s_{k-1} as the descent step.
- Solve $B_{k-1}s_{k-1} = -\nabla f(x_{k-1})$



Quasi newton Template

Let $x_0 \in \mathbb{R}^n$, $B_0 \succ 0$. For $k = 1, 2, \dots$, repeat:

- Define s_{k-1} as the descent step.
- Solve $B_{k-1}s_{k-1} = -\nabla f(x_{k-1})$
- Update $x_k = x_{k-1} + \alpha_k s_{k-1}$



Quasi newton Template

Let $x_0 \in \mathbb{R}^n$, $B_0 \succ 0$. For $k = 1, 2, \dots$, repeat:

- Define s_{k-1} as the descent step.
- Solve $B_{k-1}s_{k-1} = -\nabla f(x_{k-1})$
- Update $x_k = x_{k-1} + \alpha_k s_{k-1}$
- Compute B_k from B_{k-1}



Quasi newton Template

Let $x_0 \in \mathbb{R}^n$, $B_0 \succ 0$. For $k = 1, 2, \dots$, repeat:

- Define s_{k-1} as the descent step.
- Solve $B_{k-1}s_{k-1} = -\nabla f(x_{k-1})$
- Update $x_k = x_{k-1} + \alpha_k s_{k-1}$
- Compute B_k from B_{k-1}



Quasi newton Template

Let $x_0 \in \mathbb{R}^n$, $B_0 \succ 0$. For $k = 1, 2, \dots$, repeat:

- Define s_{k-1} as the descent step.
- Solve $B_{k-1}s_{k-1} = -\nabla f(x_{k-1})$
- Update $x_k = x_{k-1} + \alpha_k s_{k-1}$
- Compute B_k from B_{k-1}

Different Quasi Newton methods implement step three differently. Also it makes sense to update B_k^{-1} directly from B_{k-1}^{-1} .



Quasi newton Template

Let $x_0 \in \mathbb{R}^n$, $B_0 \succ 0$. For $k = 1, 2, \dots$, repeat:

- Define s_{k-1} as the descent step.
- Solve $B_{k-1}s_{k-1} = -\nabla f(x_{k-1})$
- Update $x_k = x_{k-1} + \alpha_k s_{k-1}$
- Compute B_k from B_{k-1}

Different Quasi Newton methods implement step three differently. Also it makes sense to update B_k^{-1} directly from B_{k-1}^{-1} .

Also, a reasonable requirement for B_k is the secant method discussed earlier: $\nabla f(x_k) = \nabla f(x_{k-1}) + B_k s_{k-1}$.



Quasi newton Template

Let $x_0 \in \mathbb{R}^n$, $B_0 \succ 0$. For $k = 1, 2, \dots$, repeat:

- Define s_{k-1} as the descent step.
- Solve $B_{k-1}s_{k-1} = -\nabla f(x_{k-1})$
- Update $x_k = x_{k-1} + \alpha_k s_{k-1}$
- Compute B_k from B_{k-1}

Different Quasi Newton methods implement step three differently. Also it makes sense to update B_k^{-1} directly from B_{k-1}^{-1} .

Also, a reasonable requirement for B_k is the secant method discussed earlier: $\nabla f(x_k) = \nabla f(x_{k-1}) + B_k s_{k-1}$.

Additionally, we want a) B_k to be symmetric, b) B_k to be close to B_{k-1} and c) $B_{k-1} \succ 0 \Rightarrow B_k \succ 0$.



Symmetric Rank-One Update

- Easiest choice is a rank one update of the kind:

$$B_k = B_{k-1} + \alpha u u^T$$



Symmetric Rank-One Update

- Easiest choice is a rank one update of the kind:

$$B_k = B_{k-1} + \alpha uu^T$$

- Denote $y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})$.



Symmetric Rank-One Update

- Easiest choice is a rank one update of the kind:

$$B_k = B_{k-1} + \alpha u u^T$$

- Denote $y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})$.
- The secant condition is $B_k s_{k-1} = y_{k-1}$ and this implies:

$$\alpha(u^T s_{k-1})u = y_{k-1} - B_{k-1}s_{k-1}$$



Symmetric Rank-One Update

- Easiest choice is a rank one update of the kind:

$$B_k = B_{k-1} + a u u^T$$

- Denote $y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})$.
- The secant condition is $B_k s_{k-1} = y_{k-1}$ and this implies:

$$a(u^T s_{k-1})u = y_{k-1} - B_{k-1}s_{k-1}$$

- Define $u = y_{k-1} - B_{k-1}s_{k-1}$. Then solving for the above, we get:
 $a = 1/(y_{k-1} - B_{k-1}s_{k-1})^T s_{k-1}$ which leads:

$$B_k = B_{k-1} + \frac{(y_{k-1} - B_{k-1}s_{k-1})^T(y_{k-1} - B_{k-1}s_{k-1})}{(y_{k-1} - B_{k-1}s_{k-1})^T s_{k-1}}$$



Symmetric Rank-One Update

- This is great! But we still need to solve for s_k : $B_k s_k = -\nabla f(x_k)$.
Recall that we obtain $x_k = x_{k-1} + \alpha_k s_{k-1}$



Symmetric Rank-One Update

- This is great! But we still need to solve for s_k : $B_k s_k = -\nabla f(x_k)$.
Recall that we obtain $x_k = x_{k-1} + \alpha_k s_{k-1}$
- Fortunately, we can propagate inverses using the Sherman-Morrison formula:

$$(A + uu^T)^{-1} = \frac{A^{-1} - A^{-1}uu^TA^{-1}}{1 + v^TA^{-1}u}$$



Symmetric Rank-One Update

- This is great! But we still need to solve for s_k : $B_k s_k = -\nabla f(x_k)$.
Recall that we obtain $x_k = x_{k-1} + \alpha_k s_{k-1}$
- Fortunately, we can propagate inverses using the Sherman-Morrison formula:

$$(A + uu^T)^{-1} = \frac{A^{-1} - A^{-1}uu^TA^{-1}}{1 + v^TA^{-1}u}$$

- Symmetric Rank One Update is cheap(er) since now it is $O(d^2)$ in compute and memory!



Symmetric Rank-One Update

- This is great! But we still need to solve for s_k : $B_k s_k = -\nabla f(x_k)$.
Recall that we obtain $x_k = x_{k-1} + \alpha_k s_{k-1}$
- Fortunately, we can propagate inverses using the Sherman-Morrison formula:

$$(A + uu^T)^{-1} = \frac{A^{-1} - A^{-1}uu^TA^{-1}}{1 + v^TA^{-1}u}$$

- Symmetric Rank One Update is cheap(er) since now it is $O(d^2)$ in compute and memory!
- However, the key shortcomming is it does not preserve the positive definiteness!



Broyden-Fletcher-Goldfarb-Shanno (BFGS) update

- BFGS update is a rank two update:

$$B_k = B_{k-1} + \alpha uu^T + \beta vv^T$$



Broyden-Fletcher-Goldfarb-Shanno (BFGS) update

- BFGS update is a rank two update:

$$B_k = B_{k-1} + auu^T + bv v^T$$

- Key idea is to use the Secant rule and then define $u = y_{k-1}$ and $v = B_k s_{k-1}$.



Broyden-Fletcher-Goldfarb-Shanno (BFGS) update

- BFGS update is a rank two update:

$$B_k = B_{k-1} + auu^T + bvv^T$$

- Key idea is to use the Secant rule and then define $u = y_{k-1}$ and $v = B_k s_{k-1}$.
- We then get the BFGS update:

$$B_k = B_{k-1} - \frac{B_{k-1} s_{k-1} s_{k-1}^T B_{k-1}}{s_{k-1}^T B_{k-1} s_{k-1}} + \frac{y_{k-1} y_{k-1}^T}{y_{k-1}^T s_{k-1}}$$



Broyden-Fletcher-Goldfarb-Shanno (BFGS) update

- BFGS update is a rank two update:

$$B_k = B_{k-1} + auu^T + bv v^T$$

- Key idea is to use the Secant rule and then define $u = y_{k-1}$ and $v = B_k s_{k-1}$.
- We then get the BFGS update:

$$B_k = B_{k-1} - \frac{B_{k-1} s_{k-1} s_{k-1}^T B_{k-1}}{s_{k-1}^T B_{k-1} s_{k-1}} + \frac{y_{k-1} y_{k-1}^T}{y_{k-1}^T s_{k-1}}$$

- Similar to SR1 update, we can propagate the Inverse Matrices and the resulting update is $O(d^2)$ in time and memory!



Broyden-Fletcher-Goldfarb-Shanno (BFGS) update

- BFGS update is a rank two update:

$$B_k = B_{k-1} + auu^T + bv v^T$$

- Key idea is to use the Secant rule and then define $u = y_{k-1}$ and $v = B_k s_{k-1}$.
- We then get the BFGS update:

$$B_k = B_{k-1} - \frac{B_{k-1} s_{k-1} s_{k-1}^T B_{k-1}}{s_{k-1}^T B_{k-1} s_{k-1}} + \frac{y_{k-1} y_{k-1}^T}{y_{k-1}^T s_{k-1}}$$

- Similar to SR1 update, we can propagate the Inverse Matrices and the resulting update is $O(d^2)$ in time and memory!
- BFGS update also preserves the positive definiteness!



Davidon-Fletcher-Powell (DFP) update

- We have the rank two update:

$$B_k = B_{k-1} + \alpha uu^T + \beta vv^T$$



Davidon-Fletcher-Powell (DFP) update

- We have the rank two update:

$$B_k = B_{k-1} + \alpha uu^T + \beta vv^T$$

- Recall in BFGS, we set: $u = y_{k-1}$ and $v = B_k s_{k-1}$.



Davidon-Fletcher-Powell (DFP) update

- We have the rank two update:

$$B_k = B_{k-1} + auu^T + bvv^T$$

- Recall in BFGS, we set: $u = y_{k-1}$ and $v = B_{k-1}s_{k-1}$.
- We can optionally also set $u = s_{k-1}$ and $v = B_k y_{k-1}$. We get an update analogous to BFGS but with s_{k-1} and y_{k-1} exchanged!

$$B_k = B_{k-1} - \frac{B_{k-1}y_{k-1}y_{k-1}^T B_{k-1}}{y_{k-1}^T B_{k-1} y_{k-1}} + \frac{s_{k-1}s_{k-1}^T}{s_{k-1}^T y_{k-1}}$$



Davidon-Fletcher-Powell (DFP) update

- We have the rank two update:

$$B_k = B_{k-1} + auu^T + bvv^T$$

- Recall in BFGS, we set: $u = y_{k-1}$ and $v = B_{k-1}s_{k-1}$.
- We can optionally also set $u = s_{k-1}$ and $v = B_{k-1}y_{k-1}$. We get an update analogous to BFGS but with s_{k-1} and y_{k-1} exchanged!

$$B_k = B_{k-1} - \frac{B_{k-1}y_{k-1}y_{k-1}^T B_{k-1}}{y_{k-1}^T B_{k-1}y_{k-1}} + \frac{s_{k-1}s_{k-1}^T}{s_{k-1}^T y_{k-1}}$$

- DFP update also preserves the positive definiteness!



Davidon-Fletcher-Powell (DFP) update

- We have the rank two update:

$$B_k = B_{k-1} + auu^T + bvv^T$$

- Recall in BFGS, we set: $u = y_{k-1}$ and $v = B_{k-1}s_{k-1}$.
- We can optionally also set $u = s_{k-1}$ and $v = B_{k-1}y_{k-1}$. We get an update analogous to BFGS but with s_{k-1} and y_{k-1} exchanged!

$$B_k = B_{k-1} - \frac{B_{k-1}y_{k-1}y_{k-1}^T B_{k-1}}{y_{k-1}^T B_{k-1}y_{k-1}} + \frac{s_{k-1}s_{k-1}^T}{s_{k-1}^T y_{k-1}}$$

- DFP update also preserves the positive definiteness!
- In practice, BFGS often outperforms DFP!



Convergence of BFGS

- Global Convergence: If f is strongly convex, BFGS with backtracking line search converges from any x_0



Convergence of BFGS

- Global Convergence: If f is strongly convex, BFGS with backtracking line search converges from any x_0
- Superlinear local convergence: Under similar assumptions to the Newton's method, BFGS converges locally at a superlinear rate!
Empirically BFGS vs Newton: BFGS converges to the same error rate in only a constant number of iterations more than Newton, while being $O(n)$ faster in each iteration!

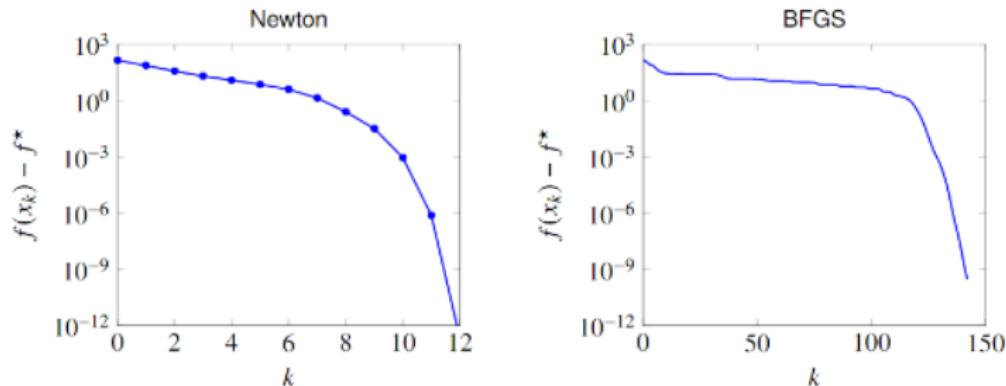


BFGS vs Newton: Empirically

Example

$$\text{minimize} \quad c^T x - \sum_{i=1}^m \log(b_i - a_i^T x)$$

$n = 100, m = 500$



- cost per Newton iteration: $O(n^3)$ plus computing $\nabla^2 f(x)$
- cost per BFGS iteration: $O(n^2)$

ALLAS

Limited Memory BFGS

- The memory and run-time cost of $O(d^2)$ is itself not scalable for large number of features!



Limited Memory BFGS

- The memory and run-time cost of $O(d^2)$ is itself not scalable for large number of features!
- We need the improved (e.g. superlinear) convergence of BFGS/Newton but with linear or sublinear per iteration complexity!



Limited Memory BFGS

- The memory and run-time cost of $O(d^2)$ is itself not scalable for large number of features!
- We need the improved (e.g. superlinear) convergence of BFGS/Newton but with linear or sublinear per iteration complexity!
- Idea of LBFGS: Do not store H_k^{-1} explicitly.



Limited Memory BFGS

- The memory and run-time cost of $O(d^2)$ is itself not scalable for large number of features!
- We need the improved (e.g. superlinear) convergence of BFGS/Newton but with linear or sublinear per iteration complexity!
- Idea of LBFGS: Do not store H_k^{-1} explicitly.
- Instead store upto l (e.g. $l = 30$) values of s_k and y_k and compute H_k^{-1} using the recursive BFGS equation and assuming $H_{k-l} = I$.



Limited Memory BFGS

- The memory and run-time cost of $O(d^2)$ is itself not scalable for large number of features!
- We need the improved (e.g. superlinear) convergence of BFGS/Newton but with linear or sublinear per iteration complexity!
- Idea of LBFGS: Do not store H_k^{-1} explicitly.
- Instead store upto l (e.g. $l = 30$) values of s_k and y_k and compute H_k^{-1} using the recursive BFGS equation and assuming $H_{k-l} = I$.
- This makes the cost per iteration of $O(dl)$ and storage also as $O(dl)$ instead of $O(d^2)$.



Summary so Far...

- Newton's method: Faster than any other algorithm seen so far.
However costly $O(d^3)$ to Invert Hessian



Summary so Far...

- Newton's method: Faster than any other algorithm seen so far.
However costly $O(d^3)$ to Invert Hessian
- SR1, BFGS and DFP are simpler updates (using the Secant condition) which are Newton like (hence Quasi-Newton) but do this in $O(d^2)$ complexity!



Summary so Far...

- Newton's method: Faster than any other algorithm seen so far.
However costly $O(d^3)$ to Invert Hessian
- SR1, BFGS and DFP are simpler updates (using the Secant condition) which are Newton like (hence Quasi-Newton) but do this in $O(d^2)$ complexity!
- LBFGS is an approximation to BFGS to make the algorithm linear!



Summary so Far...

- Newton's method: Faster than any other algorithm seen so far.
However costly $O(d^3)$ to Invert Hessian
- SR1, BFGS and DFP are simpler updates (using the Secant condition) which are Newton like (hence Quasi-Newton) but do this in $O(d^2)$ complexity!
- LBFGS is an approximation to BFGS to make the algorithm linear!
- Next: Can we achieve Newton like behaviour with Gradient Descent like algorithms?



Conjugate Gradient Descent

- Linear Conjugate Gradient is an important Algorithm for quadratic optimization



Conjugate Gradient Descent

- Linear Conjugate Gradient is an important Algorithm for quadratic optimization
- Conjugate gradient solves the problem $Ax = b$ (which can be seen as minimizing $\frac{1}{2}x^T Ax - b^T x$)



Conjugate Gradient Descent

- Linear Conjugate Gradient is an important Algorithm for quadratic optimization
- Conjugate gradient solves the problem $Ax = b$ (which can be seen as minimizing $\frac{1}{2}x^T Ax - b^T x$)
- Non-Linear Conjugate Gradient is an extension to general convex functions!



Conjugate Gradient Descent

- Linear Conjugate Gradient is an important Algorithm for quadratic optimization
- Conjugate gradient solves the problem $Ax = b$ (which can be seen as minimizing $\frac{1}{2}x^T Ax - b^T x$)
- Non-Linear Conjugate Gradient is an extension to general convex functions!
- Several variants of conjugate gradient descent. Basic idea: Instead of using $-\nabla f(x_k)$ as the update direction (as in gradient descent), do the following:



Conjugate Gradient Descent

- Linear Conjugate Gradient is an important Algorithm for quadratic optimization
- Conjugate gradient solves the problem $Ax = b$ (which can be seen as minimizing $\frac{1}{2}x^T Ax - b^T x$)
- Non-Linear Conjugate Gradient is an extension to general convex functions!
- Several variants of conjugate gradient descent. Basic idea: Instead of using $-\nabla f(x_k)$ as the update direction (as in gradient descent), do the following:
 - Set $x_{k+1} = x_k + \alpha_k d_k$



Conjugate Gradient Descent

- Linear Conjugate Gradient is an important Algorithm for quadratic optimization
- Conjugate gradient solves the problem $Ax = b$ (which can be seen as minimizing $\frac{1}{2}x^T Ax - b^T x$)
- Non-Linear Conjugate Gradient is an extension to general convex functions!
- Several variants of conjugate gradient descent. Basic idea: Instead of using $-\nabla f(x_k)$ as the update direction (as in gradient descent), do the following:
 - Set $x_{k+1} = x_k + \alpha_k d_k$
 - Set $d_{k+1} = -g_{k+1} + \beta_k d_k$ where β_k is appropriately set!



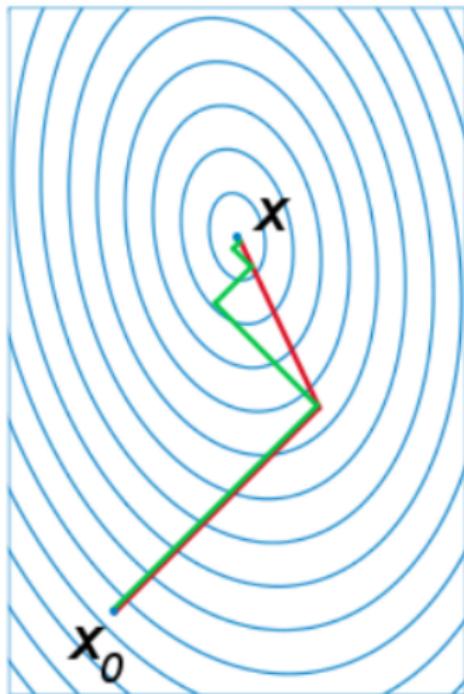
Conjugate Gradient Descent for Quadratic Functions

A comparison of

- * gradient descent with optimal step size (in green) and
- * conjugate vector (in red)

for minimizing a quadratic function.

Conjugate gradient converges in at most n steps (here n=2).



Variants of Conjugate gradient Descent

- Conjugate Gradient Framework:



Variants of Conjugate gradient Descent

- Conjugate Gradient Framework:
 - Set $x_{k+1} = x_k + \alpha_k d_k$



Variants of Conjugate gradient Descent

- Conjugate Gradient Framework:
 - Set $x_{k+1} = x_k + \alpha_k d_k$
 - Set $d_{k+1} = -g_{k+1} + \beta_k d_k$ where β_k is appropriately set!



Variants of Conjugate gradient Descent

- Conjugate Gradient Framework:
 - Set $x_{k+1} = x_k + \alpha_k d_k$
 - Set $d_{k+1} = -g_{k+1} + \beta_k d_k$ where β_k is appropriately set!
- Fletcher–Reeves CG algorithm:

$$\beta_k = \frac{||\nabla f(x_k)||^2}{||\nabla f(x_{k-1})||^2}$$



Variants of Conjugate gradient Descent

- Conjugate Gradient Framework:

- Set $x_{k+1} = x_k + \alpha_k d_k$
- Set $d_{k+1} = -g_{k+1} + \beta_k d_k$ where β_k is appropriately set!

- Fletcher–Reeves CG algorithm:

$$\beta_k = \frac{||\nabla f(x_k)||^2}{||\nabla f(x_{k-1})||^2}$$

- Polak–Ribi re:

$$\beta_k = \frac{\nabla f(x_k)^T (\nabla f(x_k) - \nabla f(x_{k-1}))}{||\nabla f(x_{k-1})||^2}$$



Variants of Conjugate gradient Descent

- Conjugate Gradient Framework:

- Set $x_{k+1} = x_k + \alpha_k d_k$
- Set $d_{k+1} = -g_{k+1} + \beta_k d_k$ where β_k is appropriately set!

- Fletcher–Reeves CG algorithm:

$$\beta_k = \frac{||\nabla f(x_k)||^2}{||\nabla f(x_{k-1})||^2}$$

- Polak–Ribi  re:

$$\beta_k = \frac{\nabla f(x_k)^T (\nabla f(x_k) - \nabla f(x_{k-1}))}{||\nabla f(x_{k-1})||^2}$$

- Hestenes–Stiefel:

$$\beta_k = \frac{\nabla f(x_k)^T (\nabla f(x_k) - \nabla f(x_{k-1}))}{d_k^T (\nabla f(x_k) - \nabla f(x_{k-1}))}$$



Conjugate Gradient and Quasi Newton

- Hestenes–Stiefel conjugate gradient update is equivalent to LBFGS with $m = 1$!



Conjugate Gradient and Quasi Newton

- Hestenes–Stiefel conjugate gradient update is equivalent to LBFGS with $m = 1$!
- Moreover, similar to Newton and BFGS, one can also show superlinear local convergence and global convergence results for CG!



Conjugate Gradient and Quasi Newton

- Hestenes–Stiefel conjugate gradient update is equivalent to LBFGS with $m = 1$!
- Moreover, similar to Newton and BFGS, one can also show superlinear local convergence and global convergence results for CG!
- Advantage of CG is that the complexity is exactly identical to Gradient Descent and yet it has some of the properties of Quasi Newton algorithms!



- Recall the quadratic expansion:

$$f(x) \approx f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2} (x - x_t)^T \nabla^2 f(x_t) (x - x_t)$$



- Recall the quadratic expansion:

$$f(x) \approx f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2} (x - x_t)^T \nabla^2 f(x_t) (x - x_t)$$

- Compute x_{t+1} by optimizing the quadratic function above using the (linear) conjugate gradient!



- Recall the quadratic expansion:

$$f(x) \approx f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2} (x - x_t)^T \nabla^2 f(x_t) (x - x_t)$$

- Compute x_{t+1} by optimizing the quadratic function above using the (linear) conjugate gradient!
- Repeat these steps till converged



- Recall the quadratic expansion:

$$f(x) \approx f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2} (x - x_t)^T \nabla^2 f(x_t) (x - x_t)$$

- Compute x_{t+1} by optimizing the quadratic function above using the (linear) conjugate gradient!
- Repeat these steps till converged
- Moreover, we can do this Hessian Free(!) because we just need to compute $\nabla^2 f(x_t)v$, i.e. we don't need to compute $\nabla^2 f$ for this!



- Recall the quadratic expansion:

$$f(x) \approx f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2} (x - x_t)^T \nabla^2 f(x_t) (x - x_t)$$

- Compute x_{t+1} by optimizing the quadratic function above using the (linear) conjugate gradient!
- Repeat these steps till converged
- Moreover, we can do this Hessian Free(!) because we just need to compute $\nabla^2 f(x_t)v$, i.e. we don't need to compute $\nabla^2 f$ for this!
- Note that $(\nabla^2 f(x)v)_i = \sum_{j=1}^d \frac{\partial^2 f}{\partial x_i \partial x_j}(x) \cdot v_j = \nabla \frac{\partial f}{\partial x_i}(x) \cdot v$



- Recall the quadratic expansion:

$$f(x) \approx f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2} (x - x_t)^T \nabla^2 f(x_t) (x - x_t)$$

- Compute x_{t+1} by optimizing the quadratic function above using the (linear) conjugate gradient!
- Repeat these steps till converged
- Moreover, we can do this Hessian Free(!) because we just need to compute $\nabla^2 f(x_t)v$, i.e. we don't need to compute $\nabla^2 f$ for this!
- Note that $(\nabla^2 f(x)v)_i = \sum_{j=1}^d \frac{\partial^2 f}{\partial x_i \partial x_j}(x).v_j = \nabla \frac{\partial f}{\partial x_i}(x).v$
- This is the directional derivative of $\frac{\partial f}{\partial x_i}$ in the direction of v ! and can be computed numerically!



Barzelai Borwein (BB) Gradient Descent

- The BB Method was introduced in an 8-page paper in 1988 (J. Barzilao and J. Borwein. Two point step size gradient method, IMA J. Numerical Analysis, 1988)



Barzelai Borwein (BB) Gradient Descent

- The BB Method was introduced in an 8-page paper in 1988 (J. Barzilao and J. Borwein. Two point step size gradient method, IMA J. Numerical Analysis, 1988)
- Its a Gradient method with special step sizes.



Barzelai Borwein (BB) Gradient Descent

- The BB Method was introduced in an 8-page paper in 1988 (J. Barzilao and J. Borwein. Two point step size gradient method, IMA J. Numerical Analysis, 1988)
- Its a Gradient method with special step sizes.
- Motivated by Newtons method but does not compute the Hessian!



Barzelai Borwein (BB) Gradient Descent

- The BB Method was introduced in an 8-page paper in 1988 (J. Barzilao and J. Borwein. Two point step size gradient method, IMA J. Numerical Analysis, 1988)
- It's a Gradient method with special step sizes.
- Motivated by Newtons method but does not compute the Hessian!
- At no extra cost over the standard gradient descent, this method can significantly outperform GD!



Barzelai Borwein (BB) Gradient Descent

- The BB Method was introduced in an 8-page paper in 1988 (J. Barzilao and J. Borwein. Two point step size gradient method, IMA J. Numerical Analysis, 1988)
- It's a Gradient method with special step sizes.
- Motivated by Newtons method but does not compute the Hessian!
- At no extra cost over the standard gradient descent, this method can significantly outperform GD!
- Typically used with non-monotone line search as convergence safeguard against non-quadratic problems (we won't cover this in class).



Barzelai Borwein (BB) Gradient Descent

- Recall Gradient Descent:

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t)$$

- . Pros: Simple! Cons: No second order information at all!



Barzelai Borwein (BB) Gradient Descent

- Recall Gradient Descent:

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t)$$

- . Pros: Simple! Cons: No second order information at all!
- Newton's method (similarly BFGS):

$$x_{t+1} = x_t - \alpha_t H_t^{-1} \nabla f(x_t)$$

- . Pros: Very Fast Convergence! Cons: Creating or updating H_k^{-1} superlinear in cost!



Barzelai Borwein (BB) Gradient Descent

- Recall Gradient Descent:

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t)$$

- . Pros: Simple! Cons: No second order information at all!
- Newton's method (similarly BFGS):
$$x_{t+1} = x_t - \alpha_t H_t^{-1} \nabla f(x_t)$$
 - . Pros: Very Fast Convergence! Cons: Creating or updating H_k^{-1} superlinear in cost!
 - BB Gradient Descent: Choose $\alpha_t \nabla f(x_t)$ such that it approximates $H_t^{-1} \nabla f(x_t)$



Barzelai Borwein (BB) Gradient Descent

- Recall Secant Condition:

$$H_k s_{k-1} = y_{k-1}$$

where $s_{k-1} = x_k - x_{k-1}$ and $y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})$



Barzelai Borwein (BB) Gradient Descent

- Recall Secant Condition:

$$H_k s_{k-1} = y_{k-1}$$

where $s_{k-1} = x_k - x_{k-1}$ and $y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})$

- Another way to see the secant condition is the fact that this is the expression of the Hessian for quadratic functions!



Barzelai Borwein (BB) Gradient Descent

- Recall Secant Condition:

$$H_k s_{k-1} = y_{k-1}$$

where $s_{k-1} = x_k - x_{k-1}$ and $y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})$

- Another way to see the secant condition is the fact that this is the expression of the Hessian for quadratic functions!
- Now, the BB update is very simple: Select α_k such that:

$$(\alpha_k^{-1} I) s_{k-1} \approx y_{k-1}$$



Barzelai Borwein (BB) Gradient Descent

- Recall Secant Condition:

$$H_k s_{k-1} = y_{k-1}$$

where $s_{k-1} = x_k - x_{k-1}$ and $y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})$

- Another way to see the secant condition is the fact that this is the expression of the Hessian for quadratic functions!
- Now, the BB update is very simple: Select α_k such that:

$$(\alpha_k^{-1} I) s_{k-1} \approx y_{k-1}$$

- Solve this using least squares!



Barzelai Borwein (BB) Gradient Descent

- Goal: Select α_k such that:

$$(\alpha_k^{-1} I) s_{k-1} \approx y_{k-1}$$



Barzelai Borwein (BB) Gradient Descent

- Goal: Select α_k such that:

$$(\alpha_k^{-1} I) s_{k-1} \approx y_{k-1}$$

- BB Method:



Barzelai Borwein (BB) Gradient Descent

- Goal: Select α_k such that:

$$(\alpha_k^{-1} I) s_{k-1} \approx y_{k-1}$$

- BB Method:

- Approach 1: Let $\beta = \alpha^{-1}$. Then:

$$\alpha_k^{-1} = \operatorname{argmin}_\alpha \frac{1}{2} \|s_{k-1}\beta - y_{k-1}\|^2 \Rightarrow \alpha_k = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T y_{k-1}}$$



Barzelai Borwein (BB) Gradient Descent

- Goal: Select α_k such that:

$$(\alpha_k^{-1} I) s_{k-1} \approx y_{k-1}$$

- BB Method:

- Approach 1: Let $\beta = \alpha^{-1}$. Then:

$$\alpha_k^{-1} = \operatorname{argmin}_\alpha \frac{1}{2} \|s_{k-1}\beta - y_{k-1}\|^2 \Rightarrow \alpha_k = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T y_{k-1}}$$

- Approach 2: Least squares in α directly:

$$\alpha_k = \operatorname{argmin}_\alpha \frac{1}{2} \|s_{k-1} - y_{k-1}\alpha\|^2 \Rightarrow \alpha_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$$



BB Gradient Descent in Practice

- BB Method:



BB Gradient Descent in Practice

- BB Method:
 - Approach 1: Let $\beta = \alpha^{-1}$. Then:

$$\alpha_k^{-1} = \operatorname{argmin}_\alpha \frac{1}{2} \|s_{k-1}\beta - y_{k-1}\|^2 \Rightarrow \alpha_k = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T y_{k-1}}$$



BB Gradient Descent in Practice

- BB Method:

- Approach 1: Let $\beta = \alpha^{-1}$. Then:

$$\alpha_k^{-1} = \operatorname{argmin}_{\alpha} \frac{1}{2} \|s_{k-1}\beta - y_{k-1}\|^2 \Rightarrow \alpha_k = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T y_{k-1}}$$

- Approach 2: Least squares in α directly:

$$\alpha_k = \operatorname{argmin}_{\alpha} \frac{1}{2} \|s_{k-1} - y_{k-1}\alpha\|^2 \Rightarrow \alpha_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$$



BB Gradient Descent in Practice

- BB Method:
 - Approach 1: Let $\beta = \alpha^{-1}$. Then:

$$\alpha_k^{-1} = \operatorname{argmin}_{\alpha} \frac{1}{2} \|s_{k-1}\beta - y_{k-1}\|^2 \Rightarrow \alpha_k = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T y_{k-1}}$$

- Approach 2: Least squares in α directly:

$$\alpha_k = \operatorname{argmin}_{\alpha} \frac{1}{2} \|s_{k-1} - y_{k-1}\alpha\|^2 \Rightarrow \alpha_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$$

- In practice, use either of these two or pick the better among the two or even alternate by fixing one for a few steps.



BB Gradient Descent in Practice

- BB Method:

- Approach 1: Let $\beta = \alpha^{-1}$. Then:

$$\alpha_k^{-1} = \operatorname{argmin}_{\alpha} \frac{1}{2} \|s_{k-1}\beta - y_{k-1}\|^2 \Rightarrow \alpha_k = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T y_{k-1}}$$

- Approach 2: Least squares in α directly:

$$\alpha_k = \operatorname{argmin}_{\alpha} \frac{1}{2} \|s_{k-1} - y_{k-1}\alpha\|^2 \Rightarrow \alpha_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$$

- In practice, use either of these two or pick the better among the two or even alternate by fixing one for a few steps.
- However, $f(x_k)$ and $\nabla f(x_k)$ are not monotonic!



BB Gradient Descent in Practice

- BB Method:

- Approach 1: Let $\beta = \alpha^{-1}$. Then:

$$\alpha_k^{-1} = \operatorname{argmin}_{\alpha} \frac{1}{2} \|s_{k-1}\beta - y_{k-1}\|^2 \Rightarrow \alpha_k = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T y_{k-1}}$$

- Approach 2: Least squares in α directly:

$$\alpha_k = \operatorname{argmin}_{\alpha} \frac{1}{2} \|s_{k-1} - y_{k-1}\alpha\|^2 \Rightarrow \alpha_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$$

- In practice, use either of these two or pick the better among the two or even alternate by fixing one for a few steps.
- However, $f(x_k)$ and $\nabla f(x_k)$ are not monotonic!
- Non Monotonic line search required for convergence analysis.



Summary So Far...

- Gradient Descent is easy to implement, simple with low per iteration complexity. However, at best linear convergence! For most standard cases, sublinear convergence.



Summary So Far...

- Gradient Descent is easy to implement, simple with low per iteration complexity. However, at best linear convergence! For most standard cases, sublinear convergence.
- Accelerated Gradient Descent is an improvement. Slightly more involved but similar per iteration complexity. Slightly faster but still similar to GD!



Summary So Far...

- Gradient Descent is easy to implement, simple with low per iteration complexity. However, at best linear convergence! For most standard cases, sublinear convergence.
- Accelerated Gradient Descent is an improvement. Slightly more involved but similar per iteration complexity. Slightly faster but still similar to GD!
- Newton Method is superlinear convergence! However very slow per iterations: $O(d^3)$ complexity.



Summary So Far...

- Gradient Descent is easy to implement, simple with low per iteration complexity. However, at best linear convergence! For most standard cases, sublinear convergence.
- Accelerated Gradient Descent is an improvement. Slightly more involved but similar per iteration complexity. Slightly faster but still similar to GD!
- Newton Method is superlinear convergence! However very slow per iterations: $O(d^3)$ complexity.
- BFGS improves it to $O(d^2)$ which is further improved to $O(d)$ by LBFGS!



Summary So Far...

- Gradient Descent is easy to implement, simple with low per iteration complexity. However, at best linear convergence! For most standard cases, sublinear convergence.
- Accelerated Gradient Descent is an improvement. Slightly more involved but similar per iteration complexity. Slightly faster but still similar to GD!
- Newton Method is superlinear convergence! However very slow per iterations: $O(d^3)$ complexity.
- BFGS improves it to $O(d^2)$ which is further improved to $O(d)$ by LBFGS!
- CG and BB are variants of Gradient Descent but try to mimic Newton's methods and in practice are much faster than simple GD while maintaining the simplicity!

