CS7301: Advanced Topics in Optimization for Machine Learning

Lecture 5.1: Second Order Methods Cont., Barzelia Borwein GD and Conjugate Gradient Method

Rishabh Iyer

Department of Computer Science
University of Texas, Dallas
https://github.com/rishabhk108/AdvancedOptML

February 26, 2021

Summary of Algorithms so Far

First Order Methods (Unconstrained)

- Gradient Descent
- 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.

First Order Methods (Constrained)

- Projected Gradient Descent (whenever the Projection step is easy)
- Conditional Gradient Descent

Last Class

Second Order Methods

- Newtons Method
- General Quasi Newton Template
- Symmetric Rank One Update
- BFGS Update
- DFP Update

Outline of Lecture 5.1

- LBFGS (Limited Memory BFGS)
- Conjugate Gradient Descent
- Barzelia Borowein Gradient Descent
- Coordinate Descent Algorithms

Recall: BFGS Update

- BFGS update is a rank two update:

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

- Recall BFGS update:

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

- BFGS Inverse Update:

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

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. super-linear) 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. super-linear) convergence of BFGS/Newton but with linear or sublinear per iteration complexity!
- Idea of LBFGS: Do not store B_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. super-linear) convergence of BFGS/Newton but with linear or sublinear per iteration complexity!
- Idea of LBFGS: Do not store B_k^{-1} explicitly.
- Instead store upto l (e.g. $l = 30$) values of s_k and y_k and compute B_k^{-1} using the recursive BFGS equation and assuming $B_{k-l} = I$:

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

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. super-linear) convergence of BFGS/Newton but with linear or sublinear per iteration complexity!
- Idea of LBFGS: Do not store B_k^{-1} explicitly.
- Instead store upto l (e.g. $l = 30$) values of s_k and y_k and compute B_k^{-1} using the recursive BFGS equation and assuming $B_{k-l} = I$:

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

- 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 (uwing 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 (uwing 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 (uwing 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

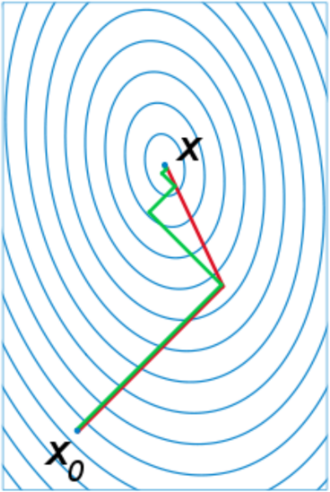
A comparison of

* 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–Ribière:

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

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–Ribière:

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

- 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 $l = 1$!

Conjugate Gradient and Quasi Newton

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

Hessian Free Optimization (James Martin, ICML 2010)

- Recall the quadratic expansion:

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

Hessian Free Optimization (James Martin, ICML 2010)

- 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!

Hessian Free Optimization (James Martin, ICML 2010)

- 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

Hessian Free Optimization (James Martin, ICML 2010)

- 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!

Hessian Free Optimization (James Martin, ICML 2010)

- 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 $(\nabla^2 f(x) v)_i = \sum_{j=1}^d \frac{\partial^2 f}{\partial x_i \partial x_j}(x).v_j = \nabla \frac{\partial f}{\partial x_i}(x).v$

Hessian Free Optimization (James Martin, ICML 2010)

- 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)
- Its 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)
- Its 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 the 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 the 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 the 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:

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

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:

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

  - Approach 2: Least squares in α directly:

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

BB Gradient Descent in Practice

- BB Method:

BB Gradient Descent in Practice

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

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

BB Gradient Descent in Practice

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

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

  - Approach 2: Least squares in α directly:

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

BB Gradient Descent in Practice

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

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

  - Approach 2: Least squares in α directly:

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

- 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:

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

  - Approach 2: Least squares in α directly:

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

- 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:

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

  - Approach 2: Least squares in α directly:

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

- 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!