CS7301: Advanced Topics in Optimization for Machine Learning

Lecture 3.1: Gradient Descent Continued

Rishabh Iyer

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

February 5, 2021

Outline

- Finish up Gradient Descent
- Lower Bounds on Convergence
- Accelerated Gradient Descent
- Gradient Descent in Practice: Line Search and Demo

Recap

Gradient Descent: Recap

- Goal: Given a convex function f, find a $x \in \mathbb{R}^n$ such that $|f(x) - f(x^*)| \leq \epsilon$
- Iterative algorithm: Initialize x_0 either randomly or say, 0. Then set

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

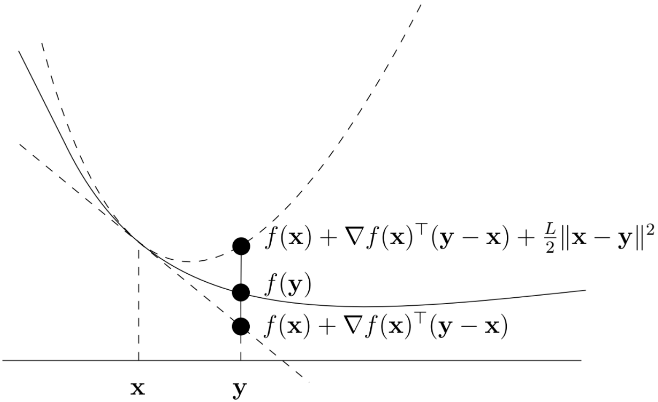
  γ is the step size parameter which needs to be set.
- Critical Question: How much time does it take to reach an ϵ-approximate solution?
- Define x^* as the Global minimizer of f
- Let f be Lipschitz continuous with parameter B. If f is smooth, let ∇f be Lipschitz continuous with parameter L.

Recap: Convergence Result for Lipschitz Continuous Case

- Final Result: Given a B-Lipschitz continuous function convex f, Gradient descent with step size $\gamma = \frac{R}{B\sqrt{T}}$ achieves a solution $\hat{x}$ s.t $|f(\hat{x}) - f(x^*)| \leq \epsilon$ in $\frac{R^2B^2}{\epsilon^2}$ iterations.
- Advantages of this bound: a) Goes to zero as T gets large, and b) Dimension Independent!
- Disadvantages: Slow convergence. To achieve a an error of 0.01, we require $10^4R^2B^2$ iterations. To achieve an error of 0.0001, the number of iterations is $10^8R^2B^2$!

UT DALLAS

Recap: Smooth Functions

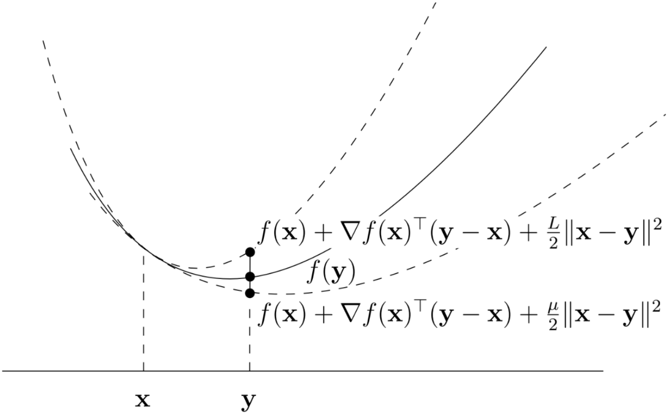


Source: Martin Jaggi (CS 439)

Recap: Convergence rate for Smooth Functions

- Final Result: Given a L smooth convex function f, Gradient descent with step size $\gamma = \frac{1}{L}$ achieves a solution x_T s.t $|f(x_T) - f(x^*)| \leq \epsilon$ in $\frac{R^2 L}{\epsilon}$ iterations.
- To achieve an error of 0.01, we require $50R^2L$ iterations instead of $10^4 R^2 B^2$ in the Lipschitz case!

Recap: Smooth + Strongly Convex Functions



Source: Martin Jaggi (CS 439)

Convergence Rate For Smooth + Strongly Convex

- Final Result: Given a L smooth and μ-strongly convex function f, Gradient descent with step size $\gamma = \frac{1}{L}$ achieves a solution x_T s.t $|f(x_T) - f(x^*)| \leq \epsilon$ in $\frac{L}{\mu}\log(\frac{R^2 L}{2\epsilon})$ iterations.
- To get an error of $\epsilon = 0.01$, we now need only $L/\mu \log(50R^2L)$ iterations as opposed to $50R^2L$ iterations in the smooth case!

Lipschitz Continuous + Strongly Convex

- Final Result: Given a B Lipschitz Continuous and μ-strongly convex function f, Gradient descent with decaying step size achieves a solution x_T s.t $|f(x_T) - f(x^*)| \leq \epsilon$ in $\frac{L}{\mu} \log(\frac{R^2 L}{2\epsilon})$ iterations.

Summary of Results so Far...

- Lipschitz continuous functions (C). With $\gamma = \frac{R}{B\sqrt{T}}$, achieve an ϵ-approximate solution in R^2B^2/ϵ^2 iterations
- Lipschitz continuous functions + Strongly Convex (CS) with $\gamma_t = \mu(1+t)/2$ achieve an an ϵ-approximate solution in $2B^2/\epsilon - 1$ iterations
- Smooth Functions (S): With $\gamma = 1/L$, achieve an ϵ-approximate solution in $\frac{R^2L}{\epsilon}$ iterations.
- Smooth + Strongly Convex (SS): With $\gamma = 1/L$, achieve an ϵ-approximate solution in $\frac{L}{\mu}\log(\frac{R^2L}{2\epsilon})$ iterations.
- Concrete examples. Let $L = B = 10, R = 1, \mu = 1$. Then we have the:
  - $\epsilon = 0.1$, C: 10000, CS: 2000, S = 50, SS = 8.49 iterations
  - $\epsilon = 0.01$, C: 1000000, CS: 20000, S = 500, SS = 13.49 iterations
  - $\epsilon = 0.001$, C: 100000000, CS: 200000, S = 5000, SS = 18.49 iterations

Lower Bounds (No Proof)

- Case I: Lipschitz Continuous: Any black-box procedure will have an error of at least $\frac{RB}{2(1+\sqrt{t})}$ (GD: $\frac{RB}{\sqrt{T}}$)

Lower Bounds (No Proof)

- Case I: Lipschitz Continuous: Any black-box procedure will have an error of at least $\frac{RB}{2(1+\sqrt{t})}$ (GD: $\frac{RB}{\sqrt{T}}$)
- Case II: Lipschitz Continuous + Strongly Convex: Any black-box procedure have an error of at least $\frac{B^2}{2\mu T}$, (GD: $\frac{2B^2}{\mu(T+1)}$)

UT DALLAS

Lower Bounds (No Proof)

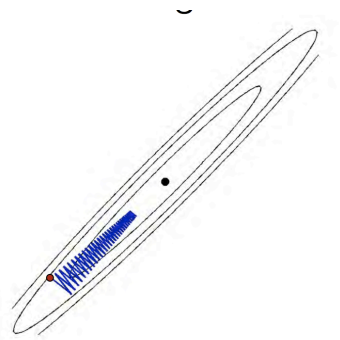
- Case I: Lipschitz Continuous: Any black-box procedure will have an error of at least $\frac{RB}{2(1+\sqrt{t})}$ (GD: $\frac{RB}{\sqrt{T}}$)
- Case II: Lipschitz Continuous + Strongly Convex: Any black-box procedure have an error of at least $\frac{B^2}{2\mu T}$, (GD: $\frac{2B^2}{\mu(T+1)}$)
- Case III: Smooth: Any black box procedure have an error of at least $\frac{3L}{32}\frac{R^2}{(T+1)^2}$ (GD: $\frac{LR^2}{2T}$)

Lower Bounds (No Proof)

- Case I: Lipschitz Continuous: Any black-box procedure will have an error of at least $\frac{RB}{2(1+\sqrt{t})}$ (GD: $\frac{RB}{\sqrt{T}}$)
- Case II: Lipschitz Continuous + Strongly Convex: Any black-box procedure have an error of at least $\frac{B^2}{2\mu T}$, (GD: $\frac{2B^2}{\mu(T+1)}$)
- Case III: Smooth: Any black box procedure have an error of at least $\frac{3L}{32}\frac{R^2}{(T+1)^2}$ (GD: $\frac{LR^2}{2T}$)
- Case IV: Smooth + Strongly Convex: Define $\kappa = \frac{L}{\mu}$. Then Any black box procedure will have an error of at least $\frac{\mu}{2}(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1})^{2(T-1)}$ (GD: $\frac{L}{2}(1-\frac{\mu}{L})^T = \frac{L}{2}(\frac{\kappa-1}{\kappa})^T$)

Why can GD be slow?

- GD has suboptimal rates for smooth and smooth + strongly convex case.
- GD relies just on local gradient information
- Can we add some momentum from the progress made so far to push it faster towards the optimal?



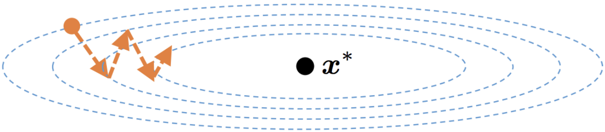
Attempt 1: Heavy Ball Momentum

- Recall standard gradient descent is $x_{t+1} = x_t - \gamma_t \nabla f(x_t)$
- Idea of Momentum: Add inertia to the Ball:

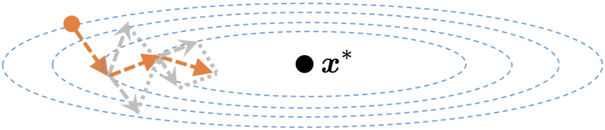
$$x_{t+1} = x_t - \gamma_t \nabla f(x_t) + \beta_t (x_t - x_{t-1})$$

- Heavy Ball result: For smooth + strongly convex functions, the heavy ball algorithm converges in $\frac{R^2}{2}(1 - \sqrt{\frac{1}{\kappa}})^T = \frac{L}{2}(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}})^T$ instead of $\frac{R^2}{2}(1 - \frac{1}{\kappa})^T = \frac{L}{2}(\frac{\kappa-1}{\kappa})^T$ (GD convergence) iterations.
- Heavy Ball momentum not optimal for the Smooth case (though it is optimal for the strongly convex + smooth class).

GD vs Momentum



gradient descent



heavy-ball method

Nesterov's Accelerated Gradient Descent

- There is a gap of a factor of T for the Smooth case! $\frac{3L}{32}\frac{R^2}{(T+1)^2}$ vs $\frac{LR^2}{2T}$!

Nesterov's Accelerated Gradient Descent

- There is a gap of a factor of T for the Smooth case! $\frac{3L}{32}\frac{R^2}{(T+1)^2}$ vs $\frac{LR^2}{2T}$!
- Nesterov's accelerated algorithm fixes this (we will not prove it in the class)

Nesterov's Accelerated Gradient Descent

- There is a gap of a factor of T for the Smooth case! $\frac{3L}{32}\frac{R^2}{(T+1)^2}$ vs $\frac{LR^2}{2T}$!
- Nesterov's accelerated algorithm fixes this (we will not prove it in the class)
- The algorithm follows the following procedure.

Nesterov's Accelerated Gradient Descent

- There is a gap of a factor of T for the Smooth case! $\frac{3L}{32}\frac{R^2}{(T+1)^2}$ vs $\frac{LR^2}{2T}$!
- Nesterov's accelerated algorithm fixes this (we will not prove it in the class)
- The algorithm follows the following procedure.
- Define $\lambda_0 = 0, \lambda_t = \frac{1+\sqrt{1+4\lambda_{t-1}^2}}{2}$ and $\beta_t = \frac{1-\lambda_t}{\lambda_{t+1}}$. Note $\beta_t \leq 0$

Nesterov's Accelerated Gradient Descent

- There is a gap of a factor of T for the Smooth case! $\frac{3L}{32}\frac{R^2}{(T+1)^2}$ vs $\frac{LR^2}{2T}$!
- Nesterov's accelerated algorithm fixes this (we will not prove it in the class)
- The algorithm follows the following procedure.
- Define $\lambda_0 = 0, \lambda_t = \frac{1+\sqrt{1+4\lambda_{t-1}^2}}{2}$ and $\beta_t = \frac{1-\lambda_t}{\lambda_{t+1}}$. Note $\beta_t \leq 0$
- Initialize $x_1 = y_1$ as an arbitrary point

Nesterov's Accelerated Gradient Descent

- There is a gap of a factor of T for the Smooth case! $\frac{3L}{32}\frac{R^2}{(T+1)^2}$ vs $\frac{LR^2}{2T}$!
- Nesterov's accelerated algorithm fixes this (we will not prove it in the class)
- The algorithm follows the following procedure.
- Define $\lambda_0 = 0, \lambda_t = \frac{1+\sqrt{1+4\lambda_{t-1}^2}}{2}$ and $\beta_t = \frac{1-\lambda_t}{\lambda_{t+1}}$. Note $\beta_t \leq 0$
- Initialize $x_1 = y_1$ as an arbitrary point
- Step 1: $y_{t+1} = x_t - \gamma_t \nabla f(x_t)$ (like normal GD: γ_t is learning rate, for e.g. $1/L$).

Nesterov's Accelerated Gradient Descent

- There is a gap of a factor of T for the Smooth case! $\frac{3L}{32}\frac{R^2}{(T+1)^2}$ vs $\frac{LR^2}{2T}$!
- Nesterov's accelerated algorithm fixes this (we will not prove it in the class)
- The algorithm follows the following procedure.
- Define $\lambda_0 = 0, \lambda_t = \frac{1+\sqrt{1+4\lambda_{t-1}^2}}{2}$ and $\beta_t = \frac{1-\lambda_t}{\lambda_{t+1}}$. Note $\beta_t \leq 0$
- Initialize $x_1 = y_1$ as an arbitrary point
- Step 1: $y_{t+1} = x_t - \gamma_t \nabla f(x_t)$ (like normal GD: γ_t is learning rate, for e.g. $1/L$).
- Step 2: $x_{t+1} = (1-\beta_t)y_{t+1} + \beta_t y_t = y_{t+1} - \beta_t(y_{t+1} - y_t)$ (notice the similarity with heavy-ball?)

Nesterov's Accelerated Gradient Descent

- There is a gap of a factor of T for the Smooth case! $\frac{3L}{32}\frac{R^2}{(T+1)^2}$ vs $\frac{LR^2}{2T}$!
- Nesterov's accelerated algorithm fixes this (we will not prove it in the class)
- The algorithm follows the following procedure.
- Define $\lambda_0 = 0, \lambda_t = \frac{1+\sqrt{1+4\lambda_{t-1}^2}}{2}$ and $\beta_t = \frac{1-\lambda_t}{\lambda_{t+1}}$. Note $\beta_t \leq 0$
- Initialize $x_1 = y_1$ as an arbitrary point
- Step 1: $y_{t+1} = x_t - \gamma_t \nabla f(x_t)$ (like normal GD: γ_t is learning rate, for e.g. $1/L$).
- Step 2: $x_{t+1} = (1-\beta_t)y_{t+1} + \beta_t y_t = y_{t+1} - \beta_t(y_{t+1} - y_t)$ (notice the similarity with heavy-ball?)
- Theorem (Nesterov 1983): If f is convex and L-smooth, $f(y_T) - f(x^*) \leq \frac{2LR^2}{T^2}$

Nesterov's Accelerated Gradient Descent

- There is a gap of a factor of T for the Smooth case! $\frac{3L}{32}\frac{R^2}{(T+1)^2}$ vs $\frac{LR^2}{2T}$!
- Nesterov's accelerated algorithm fixes this (we will not prove it in the class)
- The algorithm follows the following procedure.
- Define $\lambda_0 = 0, \lambda_t = \frac{1+\sqrt{1+4\lambda_{t-1}^2}}{2}$ and $\beta_t = \frac{1-\lambda_t}{\lambda_{t+1}}$. Note $\beta_t \leq 0$
- Initialize $x_1 = y_1$ as an arbitrary point
- Step 1: $y_{t+1} = x_t - \gamma_t \nabla f(x_t)$ (like normal GD: γ_t is learning rate, for e.g. $1/L$).
- Step 2: $x_{t+1} = (1-\beta_t)y_{t+1} + \beta_t y_t = y_{t+1} - \beta_t(y_{t+1} - y_t)$ (notice the similarity with heavy-ball?)
- Theorem (Nesterov 1983): If f is convex and L-smooth, $f(y_T) - f(x^*) \leq \frac{2LR^2}{T^2}$
- Matches the lower bound upto contant factors!

Summary of Results so Far...

- Lipschitz continuous functions (C): R^2B^2/ϵ^2 iterations
- Lipschitz continuous functions + Strongly Convex (CS): $2B^2/\epsilon - 1$ iterations
- Smooth Functions GD (SGD): $\frac{R^2L}{\epsilon}$ iterations.
- Smooth Functions AGD (SAGD): $\sqrt{\frac{2LR^2}{\epsilon}}$ iterations
- Smooth + Strongly Convex (SS): With $\gamma = 1/L$, achieve an ϵ-approximate solution in $\frac{L}{\mu}\log(\frac{R^2L}{2\epsilon})$ iterations.
- Concrete examples. Let $L = B = 10, R = 1, \mu = 1$. Then we have the:
  - $\epsilon = 0.1$, C: 10000, CS: 2000, SGD = 50, SAGD = 14.4, SS = 8.49 iterations
  - $\epsilon = 0.01$, C: 1000000, CS: 20000, SGD = 500, SAGD: 44.72, SS = 13.49 iterations
  - $\epsilon = 0.001$, C: 100000000, CS: 200000, SGD = 5000, SAGD = 141.42, SS = 18.49 iterations

Does this Matter in Practice?

- Convergence results and Lower bounds are often worst case!

Does this Matter in Practice?

- Convergence results and Lower bounds are often worst case!
- Though there exists family of functions where the bounds are tight, it is not necessary that the same intuition carries over in practice!

Does this Matter in Practice?

- Convergence results and Lower bounds are often worst case!
- Though there exists family of functions where the bounds are tight, it is not necessary that the same intuition carries over in practice!
- Difference between Theory and Practice and the need to connect the two!

Does this Matter in Practice?

- Convergence results and Lower bounds are often worst case!
- Though there exists family of functions where the bounds are tight, it is not necessary that the same intuition carries over in practice!
- Difference between Theory and Practice and the need to connect the two!
- Next, we will implement some of these algorithms for various ML Loss Functions!

Gradient Descent in Practice: Basic Version

- Credits to Mark Schmidt from UBC for this (I converted his Matlab based tutorial to python)

Gradient Descent in Practice: Basic Version

- Credits to Mark Schmidt from UBC for this (I converted his Matlab based tutorial to python)
- Let us first initialize the Logistic Loss on a dataset

Gradient Descent in Practice: Basic Version

- Credits to Mark Schmidt from UBC for this (I converted his Matlab based tutorial to python)
- Let us first initialize the Logistic Loss on a dataset
- Next, implement a simple gradient descent algorithm.

```
def gd(funObj,w,maxEvals,alpha, ...
X,y,lam,verbosity, freq):
    ...
```

[python]

Gradient Descent in Practice: Basic Version

- Credits to Mark Schmidt from UBC for this (I converted his Matlab based tutorial to python)
- Let us first initialize the Logistic Loss on a dataset
- Next, implement a simple gradient descent algorithm.

```
def gd(funObj,w,maxEvals,alpha, ...
X,y,lam,verbosity, freq):
        ...
```

[python]

  - ‘funObj‘ is the Objective function.

Gradient Descent in Practice: Basic Version

```
def gd(funObj,w,maxEvals,alpha,X,y,lam, verbosity):
    [f,g] = funObj(w,X,y,lam)
    funEvals = 1
    funVals = []
    while(1):
        [f,g] = funObj(w,X,y,lam)
        optCond = LA.norm(g, np.inf)
        if (verbosity > 0):
            print(funEvals,alpha,f,optCond)
        w = w - alpha*g
        funEvals = funEvals+1
        if ((optCond < 1e-2) and (funEvals > maxEvals)
            break
        funVals.append(f)
    return funVals
```

Gradient Descent in Practice: Basic Version

- Run this by invoking:

```
funV = gd(LogisticLoss,w,200,1e-1,X,y,1,1,10)
```

- Try running this with different values of learning rates: $\alpha = 1e-1, 1e-3, 1e-5, ...$
- How do we find the optimal learning rate every time?
- Can there be better strategies to adapt the learning rates?
- Next, we shall see a few line search based strategies.

Armijo Backtracking Line-Search V1

- We don't want to tune α every time
- This is the idea behind line search
- Simple Line search strategy:
  - Start with a large value of α
  - Divide α by 1/2 if it doesn't satisfy Armijo's condition:

$$f(w - \alpha g) \leq f(w) - \gamma\alpha||g||^2$$

  - Basically find α such that there is a reduction in function value by atleast $\gamma\alpha||g||^2$
  - Idea: Choose α and γ such that this happens.

Armijo Backtracking Line-Search V2

- Danger with the simple backtracking is that α may quickly become very small quickly
- Easy fix: Reset α every time!
- Issue with this: Too many function evaluations lost in repeated backtracking!

Armijo Backtracking Line-Search V3

- Just halving the step size ignores the information collected during line search!
- Reduce the number of backtracks using a polynomial interpolation!
- Minimize a quadratic passing through $f(w), f'(w)$ and $f(w - \alpha g)$
- Choose α using a polynomial interpolation as follows:

$$\alpha = \frac{\alpha^2 g^T g}{2(fcurr + \alpha g^T g - f)}$$

- Here *fcurr* is the function evaluation with the current value of α and f is the function value before starting backtracking!

Armijo Backtracking Line-Search V4

- Final Issue to fix is better initialization of α.
- Initializing $\alpha = 1$ is too large in practice
- Wasted backtracks because of this.
- Use a hueristic like $\alpha = 1/||g||$
- On subsequent iterations again use a polynomial interpolation:

$$\alpha = \min(1, 2(f_{old} - f)/g^T g)$$

- A lot of this is tried empirically and based on empirical knowledge..

Finally: Accelerated Gradient Descent

- Algorithm:
  - Define $\lambda_0 = 0, \lambda_t = \frac{1+\sqrt{1+4\lambda_{t-1}^2}}{2}$ and $\gamma_t = \frac{1-\lambda_t}{\lambda_{t+1}}$.
  - Note $\gamma_t \leq 0$
  - Initialize $x_1 = y_1$ as an arbitrary point
  - Step 1: $y_{t+1} = x_t - \alpha \nabla f(x_t)$ (like normal GD)
  - Step 2: $x_{t+1} = (1-\gamma_t)y_{t+1} + \gamma_t y_t = y_{t+1} - \gamma_t(y_{t+1} - y_t)$ (slide a little bit further than y_{t+1} towards the previous point y_t!)
- In practice, we club this with Armijo line search for tuning the learning rate α.