

ILLINOIS INSTITUTE OF TECHNOLOGY

Introduction to 1st Order and 2nd Order Optimization Methods

Nathan Cao (MSC-514)
Professor Priyanka Sharma
April 8th, 2021

First order and second order optimization application

Firstly, review of optimization knowledge learnt from last semester
– Gradient Vector

Gradient Vector

All the partial derivatives of a function $y = f(x_1, x_2, \dots, x_n)$ can be collected under a single mathematical entity called the *gradient vector*, or simply the *gradient*, of function f .

Gradient Vector

$$\text{grad}f(x_1, x_2, \dots, x_n) = (f_1, f_2, \dots, f_n) = \nabla f(x_1, x_2, \dots, x_n)$$

where $f_i = \frac{\partial f}{\partial x_i}$.

- The mathematical symbol ∇ is the inverted version of the Greek letter Δ and is read as: “del”.

First order and second order optimization application

Firstly, review of optimization knowledge learnt from last semester
- Higher Order derivatives

Higher order partial derivatives

- **Second partial derivatives**

$$\frac{\partial^2 f}{\partial x_i^2} = \frac{\partial}{\partial x_i} \left(\frac{\partial f}{\partial x_i} \right)$$

- **Mixed partial derivatives**

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i} \left(\frac{\partial f}{\partial x_j} \right)$$

for $i \neq j$.

First order and second order optimization application

Firstly, review of optimization knowledge learnt from last semester

- Extreme value: optimal maximizer, minimizer or inflection point

Finding extreme values of functions of two variables

Consider $z = f(x, y)$.

- The first order optimization condition requires

$$dz = f_x dx + f_y dy = 0 \text{ for arbitrary value of } dx \text{ and } dy, \text{ not both zero}$$

Then the first order optimization condition can be expressed as:

$$f_x = f_y = 0$$

- Through some additional work, it can be shown that
 - $d^2z < 0$ iff $f_{xx} < 0$; $f_{yy} < 0$; and $f_{xx}f_{yy} > f_{xy}^2$.
 - $d^2z > 0$ iff $f_{xx} > 0$; $f_{yy} > 0$; and $f_{xx}f_{yy} > f_{xy}^2$.

First order and second order optimization application

In practical operation use cases, we usually ask how to calculate minimum or maximum value given a series of inputs, these inputs can be serial and/or stochastic

PROBLEM 1: COMPOSITE OPTIMIZATION PROBLEM

$$\min_{\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d} F(\mathbf{x}) = f(\mathbf{x}) + h(\mathbf{x})$$

- f : Convex and Smooth
- h : Convex and (Non-)Smooth

PROBLEM 2: MINIMIZING FINITE SUM PROBLEM

$$\min_{\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d} F(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x})$$

- f_i : (Non-)Convex and Smooth
- $n \gg 1$

First order and second order optimization application

To solve optimization question, start with first order and second order optimization technologies are good options.

FIRST ORDER METHODS

- Variants of Gradient Descent (GD):
 - Reduce the per-iteration cost of GD \Rightarrow Efficiency
 - Achieve the convergence rate of the GD \Rightarrow Effectiveness



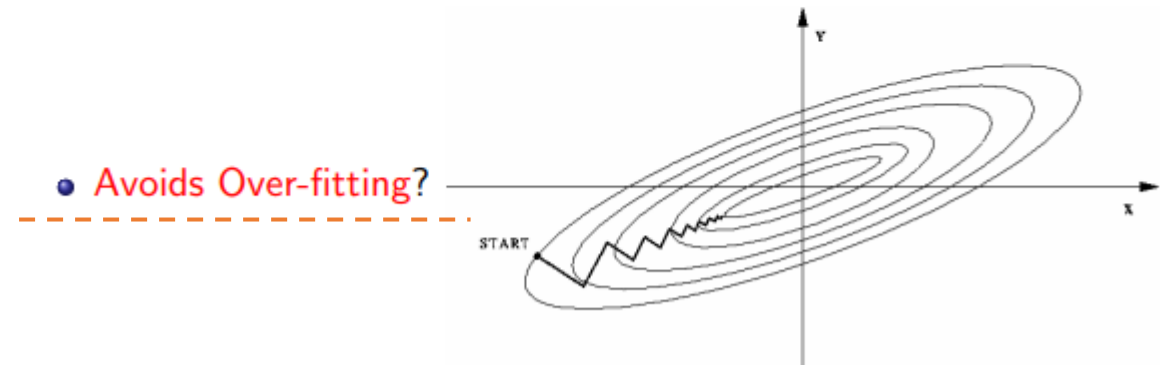
$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \nabla F(\mathbf{x}^{(k)})$$

- E.g.: SAG, SDCA, SVRG, Prox-SVRG, Acc-Prox-SVRG, Acc-Prox-SDCA, S2GD, mS2GD, MISO, SAGA, AMSVRG, ...

Q: Why do we use (stochastic) 1st order method?

- Cheaper Iterations? i.e., $n \gg 1$ and/or $d \gg 1$

- Avoids Over-fitting?



First order and second order optimization application

1st Order method & Pseudo algorithm

There are several methods:

- Plain gradient descent with adaptive step size
- Steepest descent
- Conjugate gradient
- Rprop (heuristic)

In this project submission, using plain gradient descent as example

Gradient descent with stepsize adaptation

Input: initial $x \in \mathbb{R}^n$, functions $f(x)$ and $\nabla f(x)$, initial step-size α , tolerance θ

Output: x

```

1: repeat
2:    $y \leftarrow x - \alpha \frac{\nabla f(x)}{|\nabla f(x)|}$ 
3:   if [ then step is accepted ]  $f(y) \leq f(x)$ 
4:      $x \leftarrow y$ 
5:      $\alpha \leftarrow 1.2\alpha$  // increase stepsize
6:   else [ step is rejected ]
7:      $\alpha \leftarrow 0.5\alpha$  // decrease stepsize
8:   end if
9: until  $|y - x| < \theta$  [perhaps for 10 iterations in sequence]
```

("magic numbers")

α determines the absolute stepsize

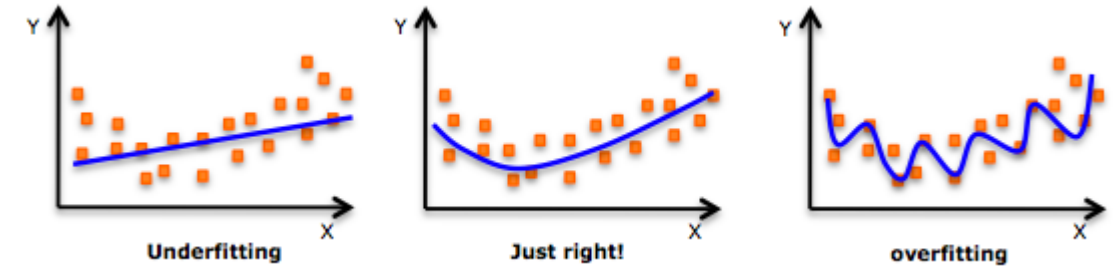
stepsize is automatically adapted

First order and second order optimization application

1st Order method and Overfitting

In designing optimization algorithm, we do pursue a goodness of fit so that a set of observations are fitted

But simple 1st order method can not achieve when having following challenges

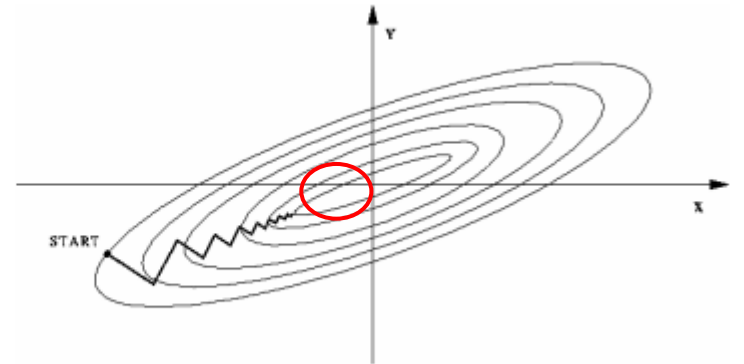


Challenges with “simple” 1st order method for “over-fitting”:

- Highly sensitive to ill-conditioning
- Very difficult to tune (many) hyper-parameters

“Over-fitting” is difficult with “simple” 1st order method!

- Avoids Over-fitting?



First order and second order optimization application

There are several 2nd order methods:

- Newton
- Gauss-newton
- Quasi-Newton

Remedy?

① “Not-So-Simple” 1st order method, e.g., *accelerated and adaptive*

② 2nd order methods, e.g.,



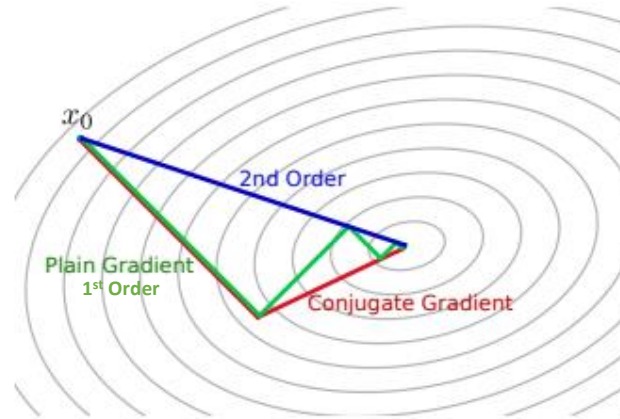
methods

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - [\nabla^2 F(\mathbf{x}^{(k)})]^{-1} \nabla F(\mathbf{x}^{(k)})$$

First order and second order optimization application

2nd order algorithm advantage

- Better direction:



- Better stepsize:
 - a *full step* jumps directly to the minimum of the local squared approx.
 - often this is already a good heuristic
 - additional stepsize reduction and dampening are straight-forward

First order and second order optimization application

**Before 2nd order optimization methods,
Let's review the constrained optimization (Lagrangian we already familiar with)**

- General constrained optimization problem:

Let $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$, $h : \mathbb{R}^n \rightarrow \mathbb{R}^l$ find

$$\min_x f(x) \quad \text{s.t.} \quad g(x) \leq 0, \quad h(x) = 0$$

The Lagrangian

- Given a constraint problem

$$\min_x f(x) \quad \text{s.t.} \quad g(x) \leq 0$$

we define the **Lagrangian** as

$$L(x, \lambda) = f(x) + \sum_{i=1}^m \lambda_i g_i(x)$$

- The $\lambda_i \geq 0$ are called **dual variables** or *Lagrange multipliers*

What's the point of this definition?

- The Lagrangian is useful to compute optima analytically, on paper – that's why physicist learn it early on
- The Lagrangian implies the KKT conditions of optimality
- Optima are necessarily at saddle points of the Lagrangian
- The Lagrangian implies a dual problem, which is sometimes easier to solve than the primal

First order and second order optimization application

Before deep dive into 2nd order optimization methods

Let's recap constrained optimization (Lagrangian we are familiar with)

The Lagrange dual problem

- We define the Lagrange **dual function** as

$$l(\lambda) = \min_x L(x, \lambda)$$

- This implies two problems

$$\min_x f(x) \quad \text{s.t.} \quad g(x) \leq 0 \quad \text{primal problem}$$

$$\max_{\lambda} l(\lambda) \quad \text{s.t.} \quad \lambda \geq 0 \quad \text{dual problem}$$

The dual problem is convex, even if the primal is non-convex!

- Written more symmetric:

$$\min_x \max_{\lambda \geq 0} L(x, \lambda) \quad \text{primal problem}$$

$$\max_{\lambda \geq 0} \min_x L(x, \lambda) \quad \text{dual problem}$$

because the $\max_{\lambda \geq 0} L(x, \lambda)$ ensures the constraints (previous slide).

- The dual function is always a lower bound (for any $\lambda_i \geq 0$)

$$l(\lambda) = \min_x L(x, \lambda) \leq \left[\min_x f(x) \quad \text{s.t.} \quad g(x) \leq 0 \right]$$

And consequently

$$\max_{\lambda \geq 0} \min_x L(x, \lambda) \leq \min_x \max_{\lambda \geq 0} L(x, \lambda)$$

- We say **strong duality** holds iff

$$\max_{\lambda \geq 0} \min_x L(x, \lambda) = \min_x \max_{\lambda \geq 0} L(x, \lambda)$$

- If the primal is convex, and there exist an interior point

$$\exists x : \forall_i : g_i(x) < 0$$

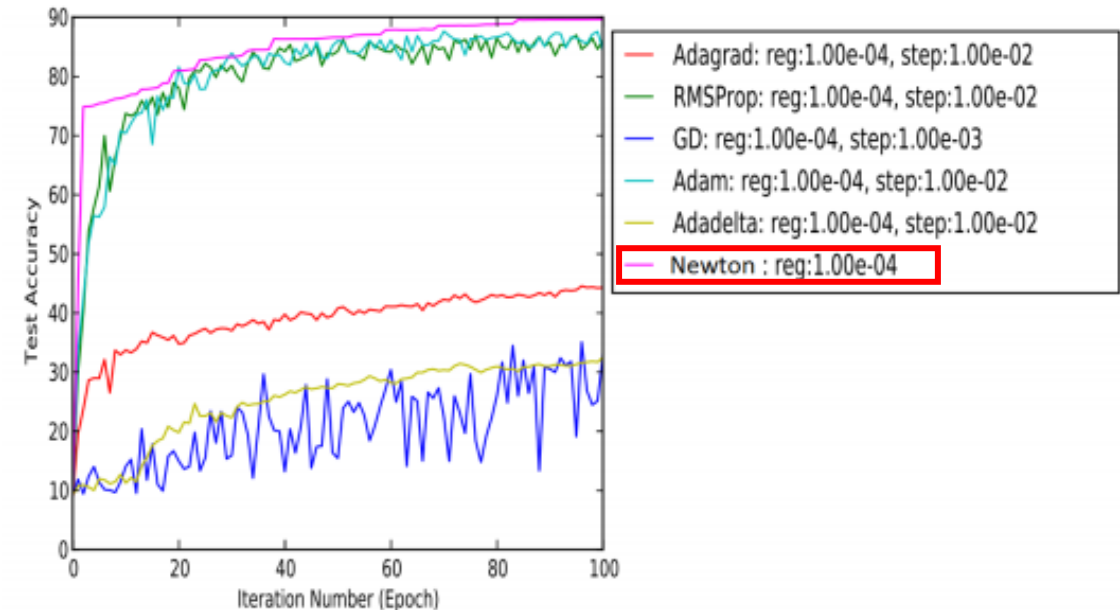
(which is called **Slater condition**), then we have *strong duality*

First order and second order optimization application

2nd order optimization methods features:

- Use both gradient and **Hessian** information
- **Fast** convergence rate
- Resilient to **ill-conditioning**
- They "**over-fit**" nicely!
- However, **per-iteration cost** is **high**!

$n : 50,000, p = 528, \text{No. Classes} = 11, \lambda : 0.0001$



First order and second order optimization application

2nd order optimization methods features:

- *Notation:*

objective function: $f : \mathbb{R}^n \rightarrow \mathbb{R}$

gradient vector: $\nabla f(x) = \left[\frac{\partial}{\partial x} f(x) \right]^\top \in \mathbb{R}^n$

Hessian (symmetric matrix):

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2}{\partial x_1 \partial x_1} f(x) & \frac{\partial^2}{\partial x_1 \partial x_2} f(x) & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} f(x) \\ \frac{\partial^2}{\partial x_1 \partial x_2} f(x) & & & \vdots \\ \vdots & & & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} f(x) & \cdots & \cdots & \frac{\partial^2}{\partial x_n \partial x_n} f(x) \end{pmatrix} \in \mathbb{R}^{n \times n}$$

Taylor expansion:

$$f(x') = f(x) + (x' - x)^\top \nabla f(x) + \frac{1}{2} (x' - x)^\top \nabla^2 f(x) (x' - x)$$

- *Problem:*

$$\min_x f(x)$$

where we can evaluate $f(x)$, $\nabla f(x)$ and $\nabla^2 f(x)$ for any $x \in \mathbb{R}^n$

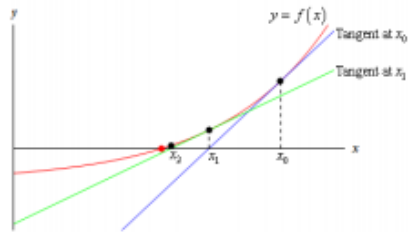
First order and second order optimization application

2nd order optimization methods

- Newton method & Pseudo algorithm

Newton method

- For finding roots (zero points) of $f(x)$



$$x \leftarrow x - \frac{f(x)}{f'(x)}$$

- For finding optima of $f(x)$ in 1D:

$$x \leftarrow x - \frac{f'(x)}{f''(x)}$$

For $x \in \mathbb{R}^n$:

$$x \leftarrow x - \nabla^2 f(x)^{-1} \nabla f(x)$$

- Often, $\nabla^2 f(x)$ is banded (non-zero around diagonal only)
 $\rightarrow Ax = b$ becomes super fast using `dpbsv` (Dynamic Programming)

Newton method with adaptive stepsize α

Input: initial $x \in \mathbb{R}^n$, functions $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)$, tolerance θ

Output: x

```

1: initialize stepsize  $\alpha = 1$  and damping  $\lambda = 10^{-10}$ 
2: repeat
3:   compute  $\Delta$  to solve  $(\nabla^2 f(x) + \lambda \mathbf{I}) \Delta = -\nabla f(x)$ 
4:   repeat                                     // "line search"
5:      $y \leftarrow x + \alpha \Delta$ 
6:     if  $f(y) \leq f(x)$  then                     // step is accepted
7:        $x \leftarrow y$ 
8:        $\alpha \leftarrow \alpha^{0.5}$              // increase stepsize towards  $\alpha = 1$ 
9:     else                                       // step is rejected
10:       $\alpha \leftarrow 0.1 \alpha$              // decrease stepsize
11:   end if
12: until step accepted or (in bad case)  $\alpha \|\Delta\|_\infty < \theta/1000$ 
13: until  $\|\Delta\|_\infty < \theta$ 
  
```

Notes:

- Line 3 computes the Newton step $\Delta = \nabla^2 f(x)^{-1} \nabla f(x)$, use special Lapack routine `dpbsv` to solve $Ax = b$ (using Cholesky decomposition)
- λ is called **damping**, makes the parabola more "steep" around current x
 for $\lambda \rightarrow \infty$: Δ becomes colinear with $-\nabla f(x)$ but $|\Delta| = 0$

First order and second order optimization application

2nd order optimization methods

To be continued