Optimization for Deep Learning

Lecture 1-1: Introduction

Kun Yuan

Peking University

Instructor

- **Kun Yuan** (kunyuan@pku.edu.cn)
- 2019.6. Ph.D. in ECE@UCLA
- 2019.8∼2022.7 DAMO Academy @ Alibaba US
- 2017. IEEE Signal Processing Society Young Author Best Paper Award
- Research interest: optimization and machine learning

Teaching assistants

- **Yutong He** (yutonghe@pku.edu.cn)
- **Jinghua Huang** (jinghua@stu.pku.edu.cn)
- **Peifeng Wu** (pengfeiwu1999@stu.pku.edu.cn)
- **Hao Yuan** (pkuyuanhao@pku.edu.cn)

Office hour: 2$\sim$3 pm every Thursday @ Jingyuan Liuyuan 220

Class policy

- **Homework assignments (30%)**

  8 homework assignments, each with $2 \sim 3$ questions. Examples:

  - show $f(x) = \|x\|_1$ is a convex function;
  - derive the gradient of logistic regression;

- **Projects (30%)**

  One coding project with a report and a 10-min presentation. Examples:

  - simulate SGD for ResNet-18 with Cifar-10; test how learning rate affects the convergence rate, training error, and test accuracy
  - compare existing popular adaptive SGD algorithms in NN training

  Each project team shall not exceed three members

- **Take-home exam (40%)**

  Bring the paper home, and submit it within one week

Reference

Optimization for Machine Learning, EPFL Class CS-439
Martin Jaggi and Nicolas Flammarion

Advanced Machine Learning Systems, Cornell CS6787
Chris De Sa

Main contents in the class

- **Part I: Fundamental algorithms for optimization**

  Gradient descent; projected gradient descent; proximal gradient descent; Nesterov acceleration; quasi-Newton algorithms; zeroth-order methods

- **Part II: Fundamental algorithms for deep learning**

  Stochastic gradient descent (SGD); SGD stability; momentum SGD; adaptive SGD; variance reduction

- **Part III: Advanced algorithms for deep learning**

  Mixed precision training; gradient clipping; adversarial learning; multi-task learning; meta learning; bilevel learning

- **Part IV: Distributed algorithm for deep learning**

  Communication compression; federated learning; decentralized learning; asynchronous SGD; Byzantine learning

Optimization

The general optimization problem

$$\begin{aligned} \underset{x \in \mathbb{R}^d}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & x \in \mathcal{X} \end{aligned}$$

- x is a continuous **optimization variable**
- $f(x)$ is the **objective function**
- $x \in \mathcal{X}$ is the **constraint**

Optimization variable, objective function, and constraint are **three pillar elements** in optimization problems

Optimization is everywhere

Optimization is used everywhere

> machine learning, big data, statistics, finance, logistics, planning, control theory, robotics, energy, aeronautics, signal processing, etc.

The procedure to solve real-world problems with optimizaiton

- **mathematical modeling:** formulate into an optimization problem
- **computational methods:** develop efficient numerical algorithms
- **implementations:** write codes and solve the problem

Core philosophy in our class

Libraries are available, algorithms treated as “black box” by most practitioners

Not here! In our class, we will

- Understand why an algorithm work and the insight behind it
- Understand how fast the algorithm can converge
- Develop novel algorithms to solve new problems

Let’s preview the main contents of our class

Optimization algorithms: Gradient descent

- Consider the following **smooth** and **unconstrained** optimization

$$\operatorname*{minimize}_{x \in \mathbb{R}^d} \quad f(x) \tag{1}$$

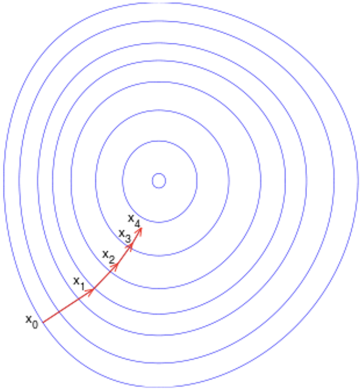
- **Gradient descent (GD)** is very effective to solve problem (1)

$$x_{k+1} = x_k - \gamma \nabla f(x_k), \quad \forall k = 0, 1, \cdots$$

  where γ is the learning rate (or step size)

- We will explore the following questions in lectures
  - Under what condition can GD converge to the desired solution?
  - How fast can GD converge?
  - How to tune γ and how does it affect the convergece rate?

Optimization algorithms: Gradient descent



Optimization algorithms: Projected gradient descent

- Consider a **smooth** but **constrained** optimization problem

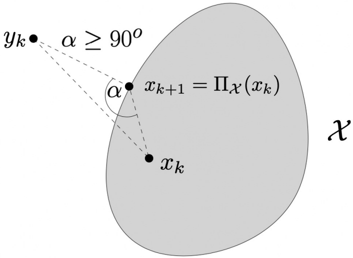
$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \quad f(x) \qquad \text{subject to} \quad x \in \mathcal{X} \tag{2}$$

- Several examples are:
  - non-negative constraint: $\mathcal{X} = \{x | x \geq 0\}$
  - probability simplex constraint: $\mathcal{X} = \{x | x \geq 0 \text{ and } 1^\top x = 1\}$
  - ℓ_1 ball constraint: $\mathcal{X} = \{x | \|x\|_1 \leq c\}$

- A simple yet effective algorithm to solve problem (2) is **projected GD**

$$\begin{aligned} y_k &= x_k - \gamma \nabla f(x_k) && \text{(Gradient descent)} \\ x_{k+1} &= \Pi_{\mathcal{X}}(y_k) := \underset{x \in \mathcal{X}}{\arg\min} \|x - y_k\|^2 && \text{(Projection)} \end{aligned}$$

Optimization algorithms: Projected gradient descent



We will explore the following questions in lectures

- Can projected GD converge to the desired solution and how fast?
- How the projection step affect its convergence rate?
- In what scenario is the projection step easy to calculate?

Optimization algorithms: Proximal gradient descent

- Consider a **non-smooth** and **unconstrained** optimization problem

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \quad f(x) + R(x) \tag{3}$$

where $f(x)$ is a smooth loss function but $R(x)$ is a non-smooth regularizer

- Several examples are:
    - ℓ_1 regularizer: $R(x) = \|x\|_1$ is to promote sparsity in variable x
    - nuclear norm regularizer: $R(X) = \|X\|_* = \sum_{i=1}^r \sigma_i(X)$
    - Indicator function regularizer $R(x) = \delta_{\mathcal{X}}(x)$

- A simple yet effective algorithm to solve problem (3) is **proximal GD**

$$\begin{aligned} y_k &= x_k - \gamma \nabla f(x_k) && \text{(Gradient descent)} \\ x_{k+1} &= \arg\min_{x \in \mathcal{X}} \{ \frac{1}{2\gamma} \|x - y_k\|^2 + R(x) \} && \text{(Proximal mapping)} \end{aligned}$$

Optimization algorithms: Proximal gradient descent

We will explore the following questions in lectures

- Can proximal GD converge to the desired solution and how fast?
- How the proximal step affect its convergence rate?

Optimization algorithms: Accelerated gradient descent

Gradient descent can be very slow for ill-conditioned problems

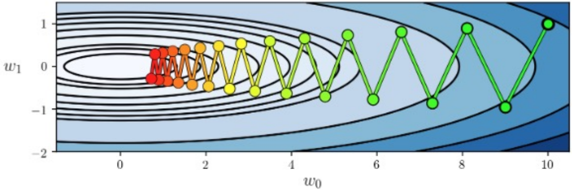


Figure: GD converges slow for ill-conditioned problem

Optimization algorithms: Accelerated gradient descent

- We have to alleviate the “Zig-Zag” to accelerate the algorithm

- **Polyak’s momentum** gradient descent

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

where $\beta \in (0, 1)$ is the momentum parameter

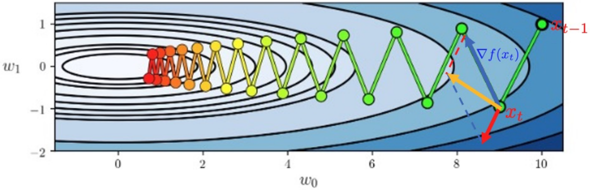


Figure: Momentum can alleviate the “Zig-Zag”

Optimization algorithms: Accelerated gradient descent

- We have to alleviate the "Zig-Zag" to accelerate the algorithm

- **Polyak's momentum** gradient descent

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

where $\beta \in (0, 1)$ is the momentum parameter

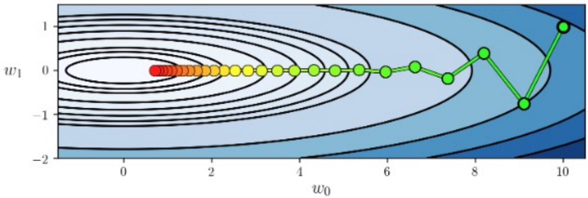


Figure: Momentum can alleviate the "Zig-Zag"

Optimization algorithms: Accelerated gradient descent

- We have to alleviate the "Zig-Zag" to accelerate the algorithm

- **Nesterov accelerated gradient (NAG)** method

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

where $\beta \in (0, 1)$ is the momentum parameter

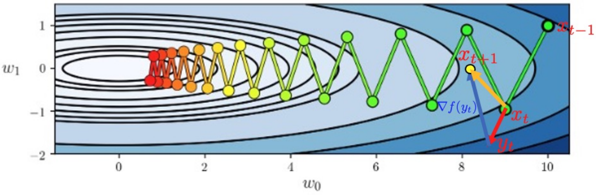


Figure: Nesterov method can alleviate the "Zig-Zag"

Optimization algorithms: Accelerated gradient descent

We will explore the following questions in lectures

- Can accelerated GD converge to the desired solution?
- Can we theoretically prove that accelerated GD gets faster in rate?
- Does accelerated GD achieve the optimal rate? Can we further improve it with more history states?

Optimization algorithms: Preconditoned GD

- Another way to accelerate GD is to use **preconditioning**. Consider a general ill-conditioned optimization problem

$$\min_{x \in \mathbb{R}^d} \quad f(x)$$

- We let $x = P^{\frac{1}{2}} w$ so that $g(w) = f(P^{\frac{1}{2}} w)$ is a nice function, where P is a positive definite matrix

- Use gradient descent to minimize $g(w)$, i.e.,

$$w_{k+1} = w_k - \gamma \nabla g(w_k) = w_k - \gamma P^{\frac{1}{2}} \nabla f(P^{\frac{1}{2}} w_k)$$

- Left-multiplying $P^{\frac{1}{2}}$ to both sides, we achieve

$$\begin{aligned} P^{\frac{1}{2}} w_{k+1} &= P^{\frac{1}{2}} w_k - \gamma P \nabla f(P^{\frac{1}{2}} w_k) \\ \iff \quad x_{k+1} &= x_k - \gamma P \nabla f(x_k) \quad \text{(Preconditioned GD)} \end{aligned}$$

  where P is called the **preconditioning matrix**.

Optimization algorithms: Preconditoned GD

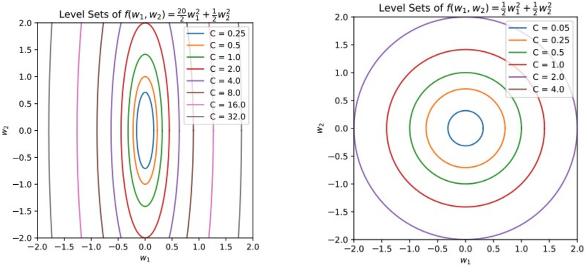


Figure: Left: an ill-conditioned problem $f(x)$. Right: a benign problem $g(w)$ after transformation.(From Prof. Chris De Sa's lecture notes)

Minimizing $g(w)$ is much easier than minimizing $f(x)$, which is the main insight behind preconditioned GD

Optimization algorithms: Newton method

- The preconditioned GD algorithm

$$x_{k+1} = x_k - \gamma P \nabla f(x_k)$$

- It is critical to choose the preconditioning matrix P

- If $P = [\nabla^2 f(x_k)]^{-1}$, then preconditioned GD reduces to **Newton method**

- Newton method converge much faster than gradient descent

Optimization algorithms: Newton method

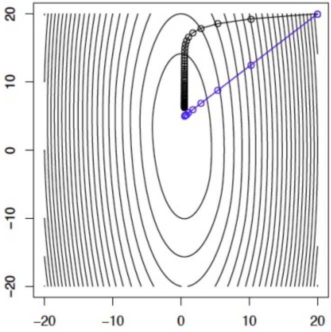


Figure: Convergence comparison between GD and Newton.

Optimization algorithms: Newton method

We will explore the following questions in lectures

- Can Newton method converge to the desired solution and how fast?
- Is Newton method provably faster than GD or even accelerated GD?
- Construct $P = [\nabla^2 f(x_k)]^{-1}$ is very expensive. Is there any efficient strategy to constrcut effecitve P?

Optimization algorithms: Zeroth-order method

- Consider an unconstrained optimization problem

$$\min_{x \in \mathbb{R}^d} \quad f(x)$$

- The gradient of $\nabla f(x)$ cannot be easily computed in certain scenarios
    - the objective function is implicit: hyper-parameter tuning; adversarial attacks on neural networks
    - too expensive to compute the gradient
    - save memory: no need of backpropagation in DNN training

- **Evaluate the gradient** with zeroth-order information

$$\widehat{\nabla} f(x) = \sum_{i=1}^{d} \frac{f(x + \epsilon e_i) - f(x - \epsilon e_i)}{2\epsilon} e_i$$

Optimization algorithms: Zeroth-order method

- **Zeroth-order optimization**

$$x_{k+1} = x_k - \gamma \widehat{\nabla f}(x_k)$$

- We will explore the following questions in lectures
  - Any smarter approaches to evaluate the gradient?
  - Can zeroth-order methods converge to the solution and how fast?
  - How does the variable dimension d influence the rate?

Summary

- Optimization is used everywhere
- We focus on the theoretical understanding of optimization algorithms
- We previewed fundamental optimization algorihthms covered in our lectures. They are: GD, Projected GD, Proximal GD, Accelerated GD, Preconditioned GD, Newton method, zeroth-order algorithms
- In the next lecture, we will preview fundamental deep learning algorithms