Optimization for Deep Learning

Lecture 1-3: Introduction

Kun Yuan

Peking University

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 optimization

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

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

Mixed precision training

- Efficient DNN training relies on using lower precision data types
- Float16 matrix multiplication is $16\times$ faster than float32 in A100
- Using lower precision can save memory
- Training large DNN is infeasible without using mixed precision

Mixed precision training

- Mixed precision training:

$$x_{k+1} = x_k - \gamma Q\big(\nabla F(x_k, \xi_k)\big)$$

  where $Q(\cdot)$ is a quantization operator using lower precision

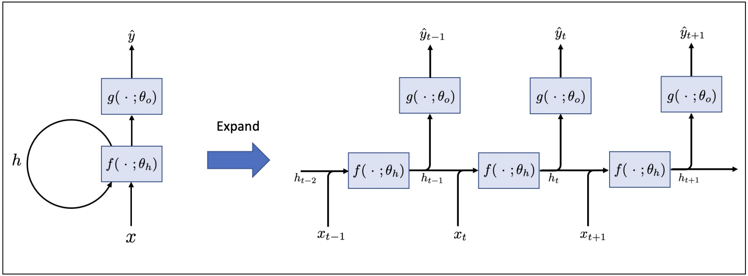
- Our lecture will explore the following questions:
  - Can the algorithm converge to the desired solution?
  - How does $Q(\cdot)$ influence the convergence rate?
  - How to design $Q(\cdot)$?
  - How to use mixed precision for momentum SGD or adaptive SGD?

Gradient clipping: recurrent neural network (RNN)

- RNN has the following recursion:

$$h_t = f(x_t, h_{t-1}; \theta_h)$$
$$\hat{y}_t = g(h_t; \theta_o)$$

where θ_h and θ_o are the parameters of $f(\cdot)$ and $g(\cdot)$, respectively, and h_0 can be initialized to arbitrary values.



Gradient clipping: vanishing and exploding gradients

- The gradient in linear RNN is:

$$\frac{\partial F}{\partial W_x} = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{t} (W_h^\intercal)^{t-i} W_o^\intercal \frac{\partial L(\hat{y}_t, y_t)}{\partial \hat{y}_t} x_i^\intercal \in \mathbb{R}^{n\times d}.$$

- $(W_h^\intercal)^t$ will cause a significant numerical issue in $\partial F / \partial W_x$

- If the largest magnitude of the eigenvalue is less than 1, i.e., $|\lambda(W_h^\intercal)| < 1$, it holds that $(W_h^\intercal)^{t-i} \to 0$ as t (or T) gets large; Gradient vanishing!

- If the largest magnitude of the eigenvalue is greater than 1, i.e., $|\lambda(W_h^\intercal)| > 1$, it holds that $(W_h^\intercal)^t \to +\infty$ as t (or T) gets large; Gradient exploding!

- Activation functions may also amplify gradient vanishing and exploding

Gradient clipping: algorithm

- Consider the following non-convex optimization problem

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

- The gradient clipping algorithm iterate as follows

$$x_t = x_t - \gamma g_t \quad \text{where} \quad g_t = \text{clip}(\nabla f(x_t), c)$$

for some positive constant $c > 0$.

- The clipping operator is defined as

$$\begin{aligned} \text{clip}(u, c) &= \min\{1, \frac{c}{\|u\|}\}u \quad \forall u \in \mathbb{R}^d \\ &= \begin{cases} u & \text{if } \|u\| \le c \\ \frac{c}{\|u\|}u & \text{if } \|u\| > c \end{cases} \end{aligned}$$

where $\|\cdot\|$ is an ℓ_2-norm.

Gradient clipping: resolving gradient exploding

- Clipping operator does not change the gradient direction; just scales gradient.
- Clipping operator squeezes large gradient when $\|\nabla f(x)\| > c$, but does nothing to small gradient.
- After clipping, it is guaranteed that $\|u\| \leq c$ for any $u \in \mathbb{R}^d$.
- Our lectures will explore the following questions:
    - Can gradient clipping overcome gradient exploding?
    - Under what conditions can gradient clipping perform better than GD?

Adversarial learning: DNN is fragile to adversarial attacks

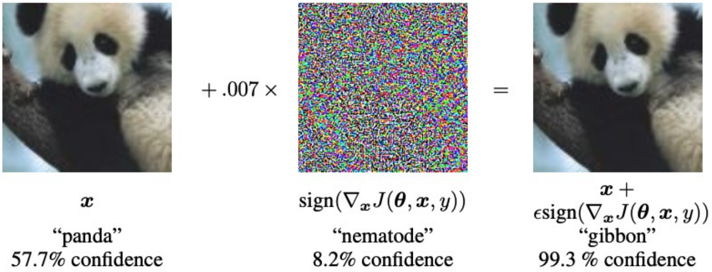
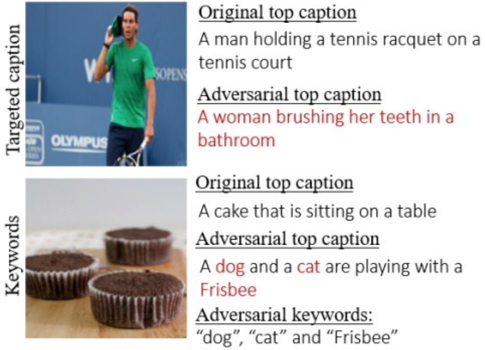


Figure: A demonstration of the adversarial example [Goodfellow et.al., 2015].

Adversarial learning: Adversarial stop sign

Adversarial learning: Adversarial T-shirt

Adversarial learning: Adversarial attacks in NLP



Adversarial learning: Adversarial attacks in NLP

Original Input	Connoisseurs of Chinese film will be pleased to discover that Tian's meticulous talent has not withered during his enforced hiatus.	Prediction: **Positive (77%)**
Adversarial example [Visually similar]	**Aonnoisseurs** of Chinese film will be pleased to discover that Tian's meticulous talent has not withered during his enforced hiatus.	Prediction: **Negative (52%)**
Adversarial example [Semantically similar]	Connoisseurs of Chinese **footage** will be pleased to discover that Tian's meticulous talent has not withered during his enforced hiatus.	Prediction: **Negative (54%)**

Adversarial learning: Construct adversarial examples

- An adversarial example is a perturbation η to maximize misclassification
- Given an input pair (ξ, y), its adversarial example $\eta \in \mathbb{R}^d$ is defined as

$$\eta \in \arg\max_{\eta:\|\eta\|\leq\epsilon} L\Big(h(x^\star, \xi+\eta), y\Big)$$

  where $x^\star$ is the optimal DNN model.
- How to solve the above problem? We leave it to the main lecture

Adversarial learning: problem formulation

- Adversarial machine learning is to make models robust to attacks

$$\min_{x \in \mathbb{R}^d} \quad \frac{1}{m} \sum_{i=1}^{m} f_i(x) \quad \text{where} \quad f_i(x) = \max_{\eta : \|\eta\|_\infty \leq \epsilon} L(h(x; \xi_i + \eta), y_i)$$

- We maximize η to construct adversarial examples but minimize x to construct robust machine learning models; minimax optimization!

- How to solve the above problem? We leave it to the main lecture

Meta learning

- Deep learning algorithms have too many hyper-parameters
    - learning rate, momentum coefficient, network architecture, etc.
- Manually tuning the hyper-parameters is neither efficient nor effective
- Can we tune the hyper-parameter automatically? Yes! **Meta learning**!

Meta learning

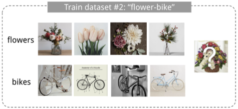
Task 1

train task 1 by $\theta_{k+1}^{(1)} = \theta_k^{(1)} - \gamma g(\theta_k^{(1)})$

to get a good classifier $\theta_\star^{(1)}$

⋮

Task N



train task N by $\theta_{k+1}^{(N)} = \theta_k^{(N)} - \gamma g(\theta_k^{(N)})$

to get a good classifier $\theta_\star^{(N)}$

Learn a good learning rate $\gamma_\star$ that performs well in tall tasks

New task

use $\gamma_\star$

train new task by $\theta_{k+1} = \theta_k - \gamma_\star g(\theta_k)$

Meta learning

- Suppose we have a collection of M tasks $\{\mathcal{T}_i\}_{i=1}^{M}$. Each task is associated with a dataset pair $(\mathcal{D}_i^{\text{tr}}, \mathcal{D}_i^{\text{test}})$.

- Let ϕ be the hyper-parameter to learn, which is common to all tasks.

- Let θ_i be the model for task i. Given an specific algorithm $\mathcal{A}lg$ (such as SGD), the hyper-parameter ϕ (such as the learning rate), and the training dataset $\mathcal{D}_i^{\text{tr}}$, θ_i can be learned by

$$\theta_i = \mathcal{A}lg(\phi, \mathcal{D}_i^{\text{tr}}) = \operatorname*{arg\,min}_{\theta \in \mathbb{R}^d} \left\{ \mathbb{E}_{\xi_i \sim \mathcal{D}_i^{\text{tr}}} [F(\theta; \phi, \xi_i)] \right\}$$

- The hyper-parameter ϕ can be learned by the mata-learning problem

$$\phi^\star = \operatorname*{arg\,min}_{\phi \in \mathbb{R}^s} \left\{ \frac{1}{M} \sum_{i=1}^{M} L(\theta_i, \mathcal{D}_i^{\text{test}}) \right\} \text{ where } \theta_i = \mathcal{A}lg(\phi, \mathcal{D}_i^{\text{tr}})$$

Meta learning

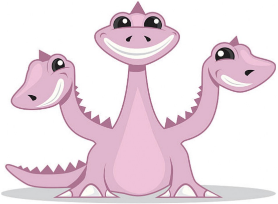
- Meta learning is essentially a bi-level optimization problem:

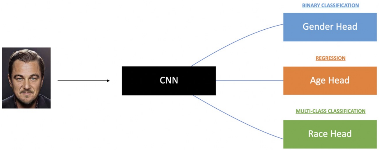
$$\min_{x \in \mathbb{R}^p} \Phi(x) := f(x, y^\star(x)), \quad \text{where} \quad y^\star(x) = \arg\min_{y \in \mathbb{R}^q} \{g(x, y)\}$$

- How to solve bilevel optimization? We leave it to the main lecture

Multi-task learning

- Multi-task learning solves many tasks simultaneously with one neural network
    - Dataset of each task can be shared with each other
    - More efficient than train each individual network independently
- One body with multiple heads





Multi-task learning

- Multi-task learning can be formulated as

$$\min_{u,\{v_i\}} \quad \frac{1}{M}\sum_{i=1}^{M} f(u, v_i) \quad \text{where} \quad f(u, v_i) = \mathbb{E}[F(u, v_i, \xi_i)]$$

where u is the shared model while v_i is the specific model for task i.

- How to solve the above problem? We leave it to the main lecture

More advanced deep learning problems

- Large-batch learning
- Self-supervised learning
- Contrastive learning
- Multi-model learning

We will discuss them in lectures if time allows

Summary

- Many deep learning tasks can be formulated into more advanced optimization problems
- We previewed mixed precision learning, gradient clipping, adversarial learning, meta learning, and multi-task learning in this lecture