Optimization for Deep Learning

Lecture 1-2: 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**

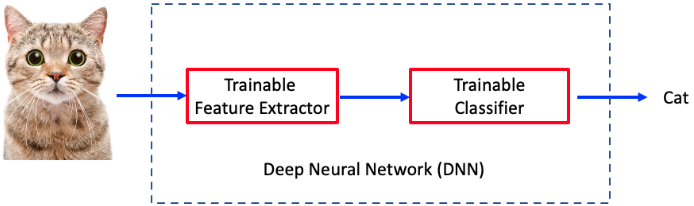
  Mixed precision training; gradient clipping; adversarial learning; multi-task learning; meta learning; bilevel optimization

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

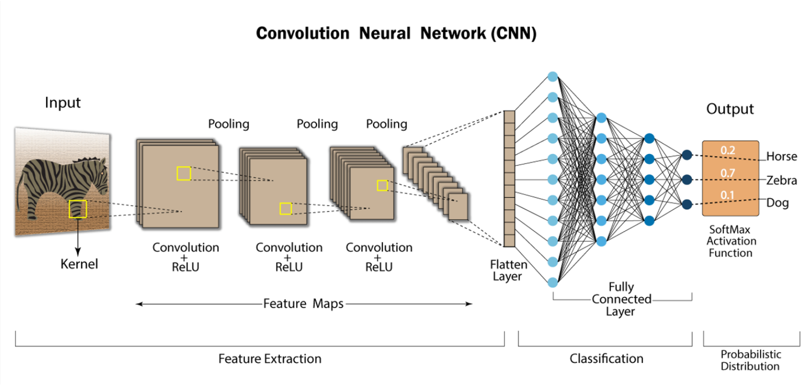
  Communication compression; federated learning; decentralized learning; asynchronous SGD; Byzantine learning;

Deep netrual network (DNN)

- DNN is widely used in almost all AI applications
- A typical DNN model includes a feature extractor and a classifier
- Well-trained DNN can make precise predictions



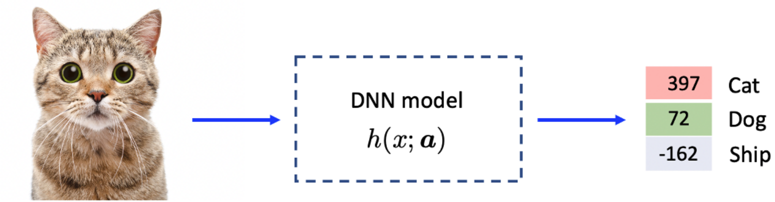
A practical DNN example[1]



[1] Source: analyticsvidhya.com

DNN model

- We model DNN as $h(x; a) : \mathbb{R}^d \to \mathbb{R}^c$
    - $x \in \mathbb{R}^d$ is the DNN model parameter to be trained
    - a is a random input data sample
    - c is the number of classes
- Given the model parameter x, DNN outputs prediction scores $\hat{b}$ for input ξ_i



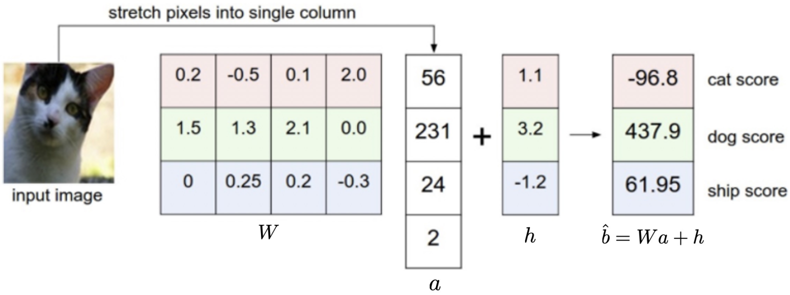
397	Cat
72	Dog
-162	Ship

$\boldsymbol{a}$

$\hat{\boldsymbol{b}} = h(x; \boldsymbol{a}) \in \mathbb{R}^c$

DNN model: a trivial example

- Given model parameter $x = [W; h]$, and a linear model $h(x; a) = Wa + h$,
- An illustration of the trivial DNN model and its output is as follows[2]



$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{a} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{h} \rightarrow \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{\hat{b} = Wa + h}$$

[2] Source: https://cs231n.github.io/linear-classify/

How to train a DNN model?

- Given good model x, DNN $h(x;a)$ can make precise predictions
- But how to train/achieve the model parameter x ?
- Given a dataset $\{a_i, b_i\}_{i=1}^{m}$ where b_i is the ground-truth label for data a_i
- Define $L(\hat{b}_i, b_i) = L(h(x;a_i), b_i)$ as a loss function to measure the difference/mismatch between predictions and ground-truth labels
- DNN training is to find a model parameter x such that the mismatch (between pred and real) are minimized across the entire dataset:

$$\boxed{x^\star = \arg\min_{x \in \mathbb{R}^d} \left\{ \frac{1}{m} \sum_{i=1}^{m} L(h(x;a_i), b_i) \right\}}$$

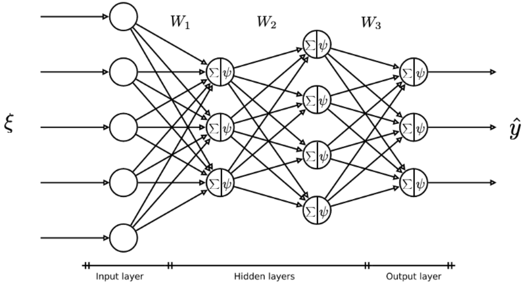
DNN model is notoriously difficult to train

- DNN model $L(h(x;a),b)$ is highly non-convex, and probably non-smooth

$$h(x;a) = \psi(\cdots\psi(W_2\cdot\psi(W_1 a + h_1) + h_2)\cdots)$$

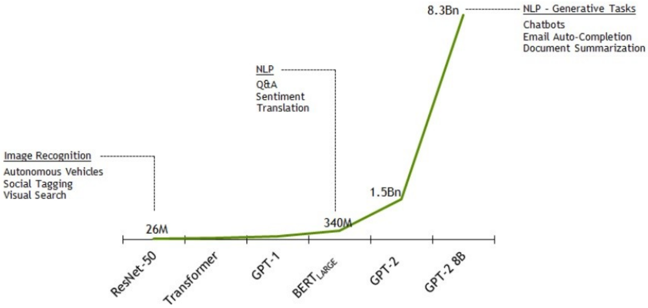
$$L(\hat{b};b) = \frac{1}{2}\|b-\hat{b}\|^2 \text{ or } -b\log(\hat{b_i}) \text{ or others}$$

where $x = \{W_i, h_i\}$ and $\psi(\cdot)$ is a non-linear activation function



DNN model is notoriously difficult to train

- Cannot find global minima; trapped into local minima and saddle points
- The dimension of model parameter $x = \{W_i, h_i\}$ (or model size) is huge[3]



[3] Image source: neowin.net

DNN model is notoriously difficult to train

- Cannot find global minima; trapped into local minima and saddle points
- The dimension of model parameter $x = \{W_j, h_j\}$ (or model size) is huge
- The size of the dataset $\{a_i, b_i\}_{i=1}^{m}$ is huge

$$\boxed{\text{DNN Trainig} = \text{Non-convexity training} + \text{Huge dimension} + \text{Huge dataset}}$$

- Our lectures will focus on algorithms to train DNN

Stochastic optimization: problem

- Consider the stochastic optimization problem:

$$\min_{x \in \mathbb{R}^d} \quad f(x) = \mathbb{E}_{\boldsymbol{\xi} \sim D}[F(x; \boldsymbol{\xi})]$$

  - $\boldsymbol{\xi}$ is a random variable indicating data samples
  - D is the data distribution; unknown in advance
  - $F(x; \boldsymbol{\xi})$ is differentiable in terms of x

- Many applications in signal processing and machine learning

Example: deep neural network

- Recall the DNN training problem

$$\min_{x \in \mathbb{R}^d} \quad \frac{1}{m} \sum_{i=1}^{m} L(h(x; a_i), b_i)$$

  which is a finite-sum problem

- Suppose we have infinite data (a, b) following distribution D, the above problem becomes

$$\min_{x \in \mathbb{R}^d} \quad f(x) = \mathbb{E}_{(\boldsymbol{a},\boldsymbol{b}) \sim D} L(h(x; \boldsymbol{a}), \boldsymbol{b})$$

Stochastic gradient descent: algorithm

- Recall the problem

$$\min_{x \in \mathbb{R}^d} \quad f(x) = \mathbb{E}_{\xi \sim D}[F(x; \boldsymbol{\xi})]$$

- Closed-form of $f(x)$ is unknown; gradient descent is not applicable

- Stochastic gradient descent (SGD):

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

  where ξ_k is a data realization sampled at iteration k.

- Our lecture will explore the following questions
  - Can SGD converge to the desired solution and how fast?
  - How does stochastic noise influence the convergence rate?
  - How to tune learning rate?

SGD Stability

- SGD was first proposed to relieve the computational overhead suffered in GD
- Surprisingly, it is often observed to generalize better than GD
- SGD first escapes from the GD solution and then converges to a better solution that can generalize better[4]

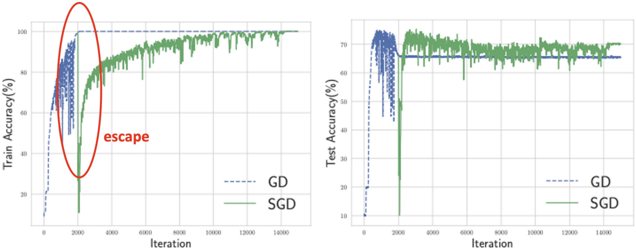
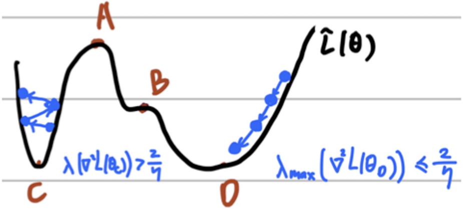


Figure 6: Fast escape phenomenon in fitting corrupted FashionMNIST.

[4] Figure is from (Wu et al., 2018)

SGD Stability

- SGD favors flat solution[5]


A
$\hat{L}(\theta)$
B
$\lambda(\nabla^2 L(\theta_C)) > \frac{2}{\eta}$
$\lambda_{max}(\nabla^2 L(\theta_D)) \leqslant \frac{2}{\eta}$
C
D

- Our lecture will explore the following questions:
  - Why does SGD favor flat solution?
  - Why does flat solution generalize better?
  - Why does SGD outperform GD in generalization?

[5] Figure is from Lei Wu's talk

Momentum SGD

- Momentum SGD can be rewritten into $(m_0 = 0)$

$$
\begin{aligned}
m_k &= \beta m_{k-1} + \nabla F(x_k; \xi_k) \\
x_{k+1} &= x_k - \gamma m_k
\end{aligned}
$$

- Equivalent to

$$x_{k+1} = x_k - \gamma \nabla F(x_k; \xi_k) + \beta \underbrace{(x_k - x_{k-1})}_{\text{momentum}}$$

- Our lecture will explore the following questions:
  - Can momentum SGD converge to the desired solution and how fast?
  - Is moemtnum SGD theoretical faster than vanilla SGD?

Adaptive gradient method (AdaGrad)

- SGD performance is very senstive to the learning rate. A good learning rate schedule can significantly speed up the convergence

- Adaptive stochastic gradient method

$$
\begin{aligned}
g_k &= \nabla F(x_k; \xi_k) \\
s_k &= s_{k-1} + g_k \odot g_k \\
x_{k+1} &= x_k - \frac{\gamma}{\sqrt{s_k} + \epsilon} \odot g_k
\end{aligned}
$$

  where $1/\sqrt{s_k} = \mathrm{col}\{1/\sqrt{s_{k,1}}, \cdots, 1/\sqrt{s_{k,d}}\} \in \mathbb{R}^d$ is an element-wise operation, s_0 is initialized as 0, and a small ϵ is added for safe-guard.

- Learning rate is automatically tuned by s_k

Adaptive gradient method (AdaGrad)

- AdaGrad falls into preconditioned SGD
- If we let $P_k = \mathrm{diag}\{\frac{1}{\sqrt{s_{k,1}}+\epsilon}, \cdots, \frac{1}{\sqrt{s_{k,d}}+\epsilon}\} \in \mathbb{R}^{d\times d}$, AdaGrad becomes

$$x_{k+1} = x_k - \gamma P_k g_k$$

  where P_k is a time-varying preconditioning matrix.

- AdaGrad imposes smaller learning rates for notable gradient directions
- AdaGrad imposes larger learning rates for insignificant gradient directions

Adaptive gradient method (AdaGrad)

AdaGrad alleviates the “Zig-Zag” phenomenon

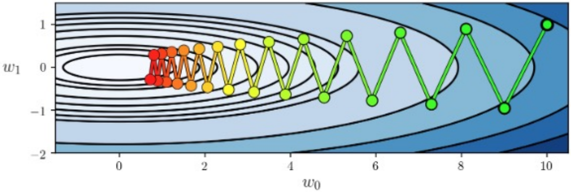


Figure: GD converges slow for ill-conditioned problem

Adaptive gradient method (AdaGrad)

AdaGrad alleviates the “Zig-Zag” phenomenon

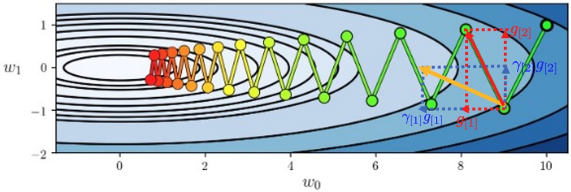
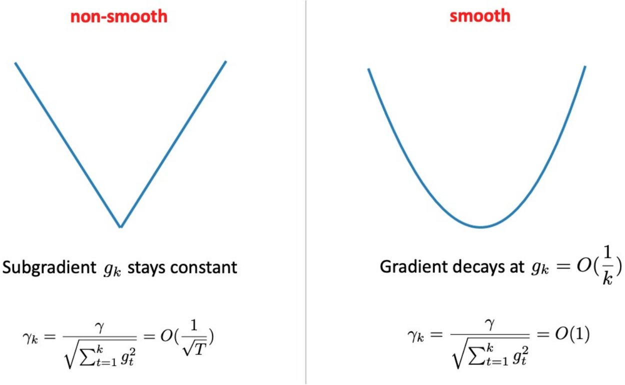


Figure: AdaGrad has alleviated “Zig-Zag” phenomenon

Adaptive gradient method (AdaGrad)

The learning rate in AdaGrad is adaptive; **no need to tune**.



Subgradient g_k stays constant

$$\gamma_k = \frac{\gamma}{\sqrt{\sum_{t=1}^{k} g_t^2}} = O(\frac{1}{\sqrt{T}})$$

Gradient decays at $g_k = O(\frac{1}{k})$

$$\gamma_k = \frac{\gamma}{\sqrt{\sum_{t=1}^{k} g_t^2}} = O(1)$$

Figure: AdaGrad automatically adapts to problem structure[6].

[6] These examples are from https://conferences.mpi-inf.mpg.de/adfocs/material/alina/adaptive-L1.pdf

More adaptive algorithms

- RMSProp, ADAM, AdaBound, etc.
- Animation of different adaptive SGD: `https://imgur.com/a/Hqolp`
- Our lecture will explore the following questions
    - How to prove convergence of these adaptive algorithms?
    - Can the adaptability bring theoretical benefits?

Finite-sum minimization

- In deep learning, we typically have a finite dataset
- If $\boldsymbol{\xi}$ takes samples uniformly from a finite dataset $\{\xi_1, \cdots, \xi_m\}$, then

$$\mathbb{E}[F(x;\boldsymbol{\xi})] = \frac{1}{m}\sum_{i=1}^{m} F(x;\xi_i) = \frac{1}{m}\sum_{i=1}^{m} F_i(x)$$

  where $F_i(x) := F(x;\xi_i)$. Number m is typically very large.

- The stochastic optimization problem becomes

$$\min_{x\in\mathbb{R}^d} \quad f(x), \quad \text{where} \quad f(x) = \frac{1}{m}\sum_{i=1}^{m} F_i(x),$$

  which is called **finite-sum minimization** problem.

Solving finite-sum minimization with SGD

- Since finite-sum minimization is special example of stochastic optimization, we can solve it with SGD

- For any $k = 0, 1, \cdots$, the SGD recursion is as follows

$$\begin{aligned}
&\text{Pick } i_k \in [m] \text{ uniformly randomly;}\\
&x_{k+1} = x_k - \gamma \nabla F_{i_k}(x_k)
\end{aligned}$$

- Unbiased stochastic gradient: $\mathbb{E}[\nabla F_{i_k}(x)] = \frac{1}{m}\sum_{i=1}^m \nabla F_i(x) = \nabla f(x)$

- Convergence property follows SGD showed in Page 5.

Stochastic variance-reduced gradient

- In SGD, stochastic gradient $\nabla F_{i_k}(x)$ has constant variance:

$$\mathbb{E}[\nabla F_{i_k}(x)] = \nabla f(x), \qquad \mathbb{E}\|\nabla F_{i_k}(x) - \nabla f(x)\|^2 \leq \sigma.$$

- The constant variance is the root reason why SGD is slow
- For infinite-data scenario, we can do little to reduce gradient variance
- But for finite-sum scenario, we can construct variance-reduced stochastic gradient with dinimishing variance, and hence significantly improve the convergence rate
- We leave it to the main lecture

Numerical performanc

Stochastic variance-reduced methods are as cheap to update as SGD, and also have a fast exponential convergence like full gradient descent.

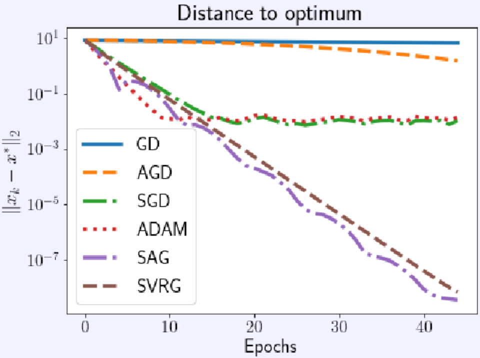


Figure: SGD, SVRG, and SAGA; figure is from (Gower et al., 2020)

Summary

- DNN training can be formulated into a non-convex stochastic optimization problem
- We introduced various fundamental algorithms to train deep neural network: SGD, momentum SGD, adaptive SGD, variance-reduction

References I

L. Wu, C. Ma *et al.*, "How sgd selects the global minima in over-parameterized learning: A dynamical stability perspective," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

R. M. Gower, M. Schmidt, F. Bach, and P. Richtárik, "Variance-reduced methods for machine learning," *Proceedings of the IEEE*, vol. 108, no. 11, pp. 1968–1983, 2020.