北京大学
PEKING UNIVERSITY

Optimization for Deep Learning (Part I)

Kun Yuan

Center for Machine Learning Research @ Peking University

Fudan University June 8, 2024

About me

北京大学 PEKING UNIVERSITY

- 2014 – 2019, Ph.D. in ECE@UCLA
- 2019 – 2022, Algorithm engineer in Alibaba Group
- 2022 – Present, Assistant professor, Peking University
- Research interests:

  Fast, **scalable**, **reliable**, and **distributed** algorithms for large-scale machine learning

- Recipient of 2017 IEEE Signal Processing Society Young Author Best Paper Award (joint with Dr. Wei Shi)

We are hiring summer interns

北京大学 PEKING UNIVERSITY

MELON(MachinE Learning and OptimizatioN)研究组主要从事优化算法与机器学习交叉领域的理论与算法研究。研究组拥有充足的 GPU 计算资源，且与国内外高校与工业界(例如 Princeton、UCLA、Upenn、Google、Meta、阿里巴巴等)有紧密合作。本次暑研课题有:

- 大模型鲁棒且高效的预训练/微调/推理算法
- 大模型智能体的探究及应用
- 分布式优化理论与算法

研究组在北大叉院有博士名额，在北大数院大数据专业有硕士名额。欢迎有保研、出国意向的同学申请。2023-2024 申请季，课题组的三位本研同学每人都有顶会顶刊论文产出，其中两位被 UC Berkeley 的博士项目录取，一位被 CMU 的博士项目录取。研究组承诺将为每位暑研的同学提供充分指导和充足的计算资源。

感兴趣的大二同学请发送简历至kunyuan@pku.edu.cn

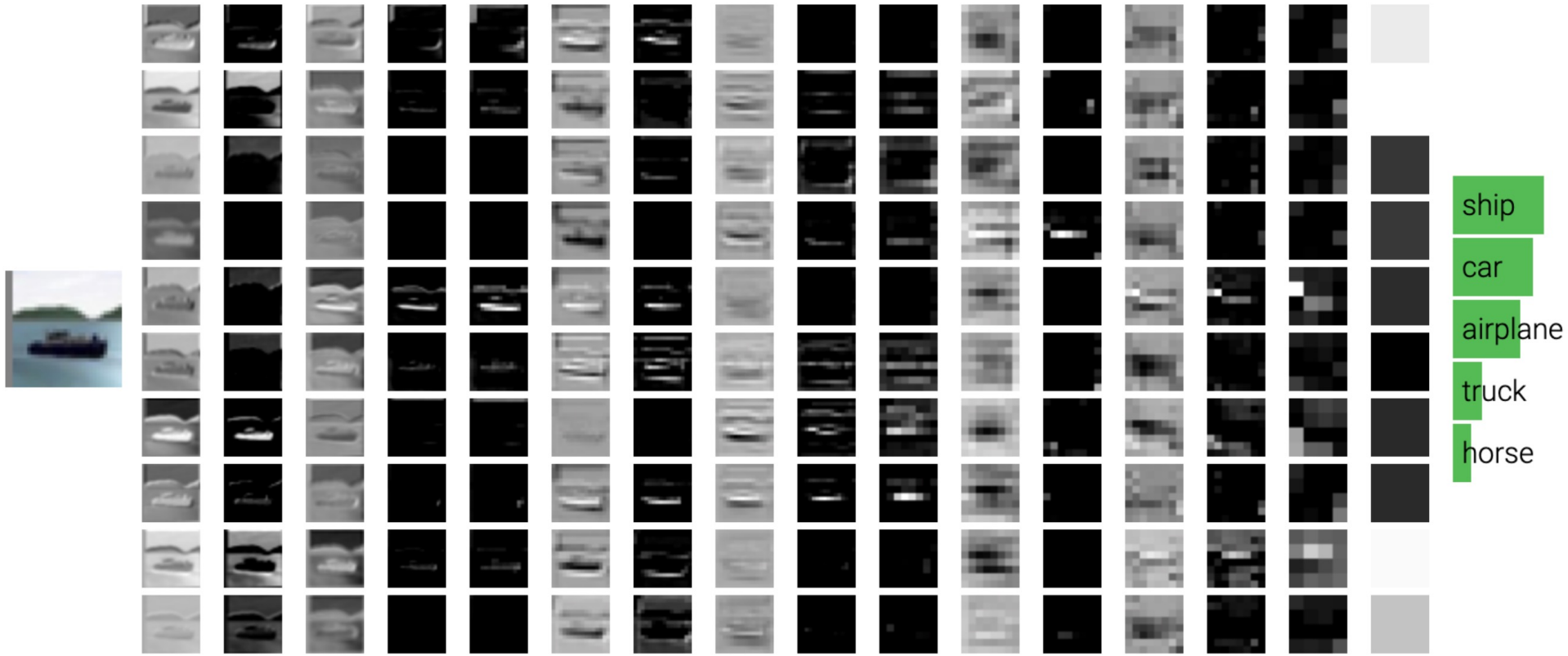
北京大学
PEKING UNIVERSITY

Part 01

Problem formulation

Multi-classification

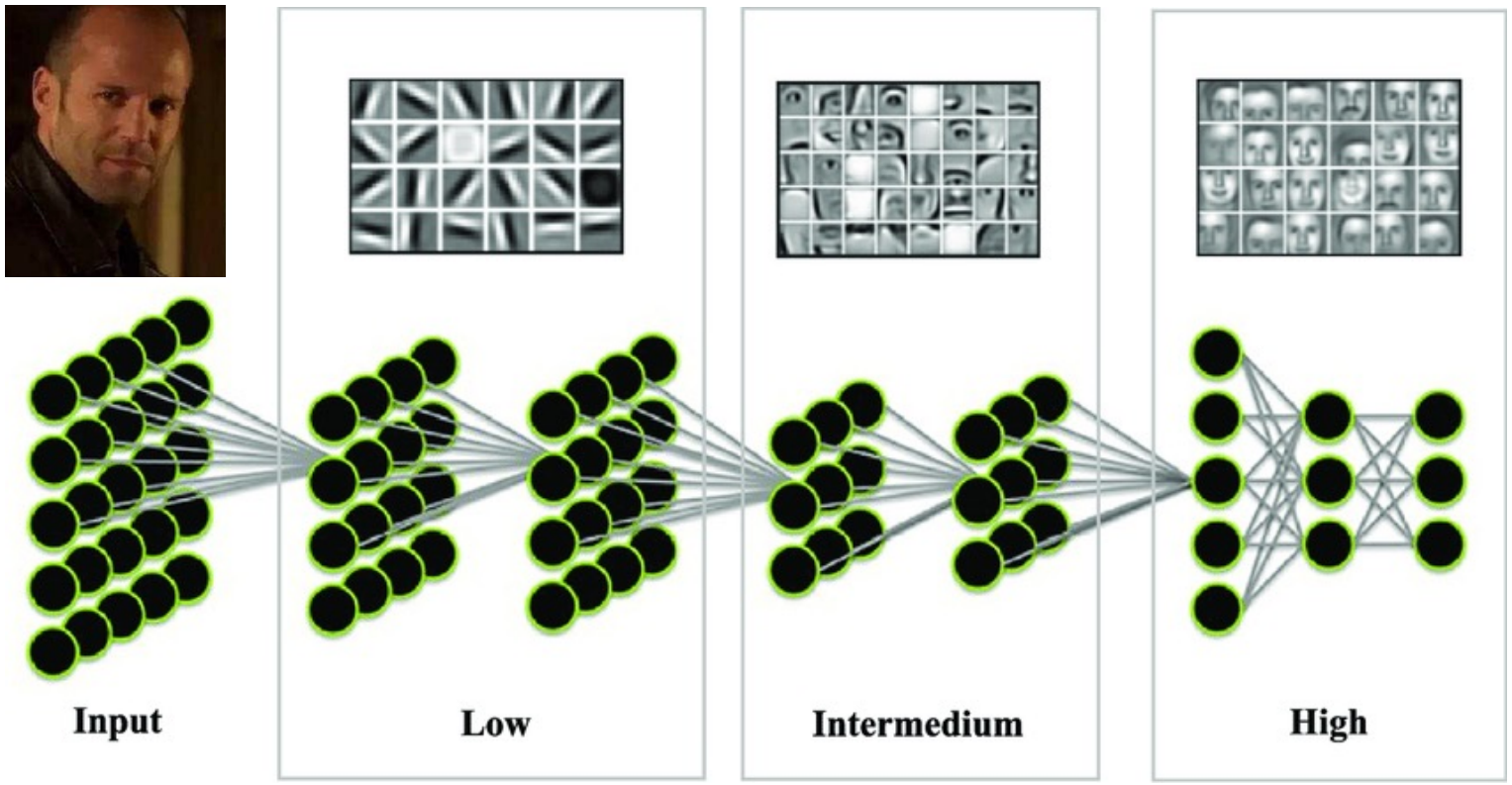
- Multi-classification is very common in real life

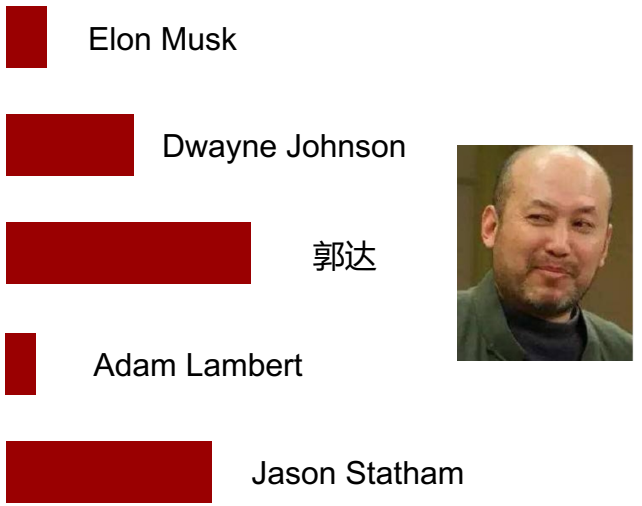


[CS231n: Deep Learning for Computer Vision]

Multi-classification

- Face recognition is one of the most successful multi-classification tasks





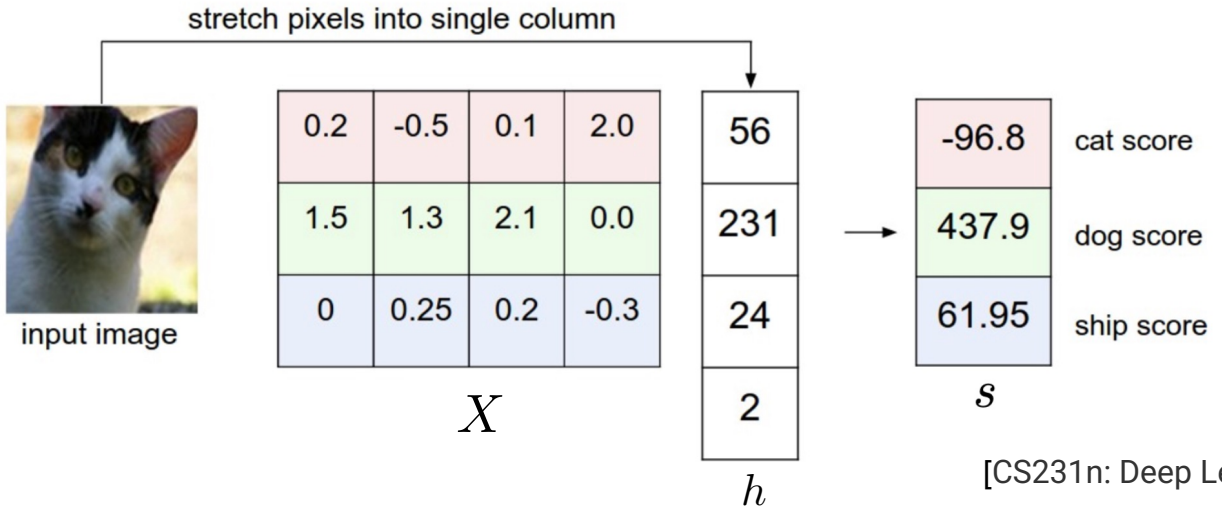
Predict the score

- We first collect the dataset $\{h_i, y_i\}_{i=1}^N$ where $h_i \in R^d$ is the feature and $y_i \in \{1, \cdots, C\}$ is the label

- We consider a linear model to predict the score of each class

$$s = Xh \in \mathbb{R}^C$$

where $X \in \mathbb{R}^{C \times d}$ is the model parameters to learn and s_i is the score that h belongs to class i

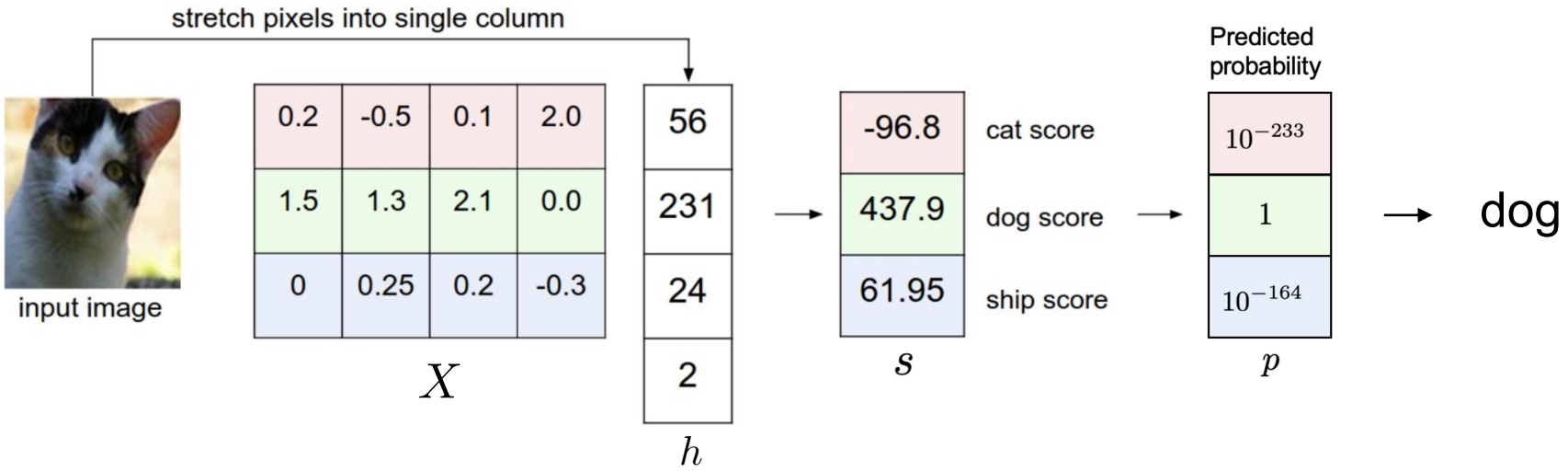


[CS231n: Deep Learning for Computer Vision]

Predict the probability

- Given the score vector s, the probability of each class with the softmax function is as follows

$$p_i = \frac{\exp(s_i)}{\sum_{j=1}^{C} \exp(s_j)} \in (0, 1)$$

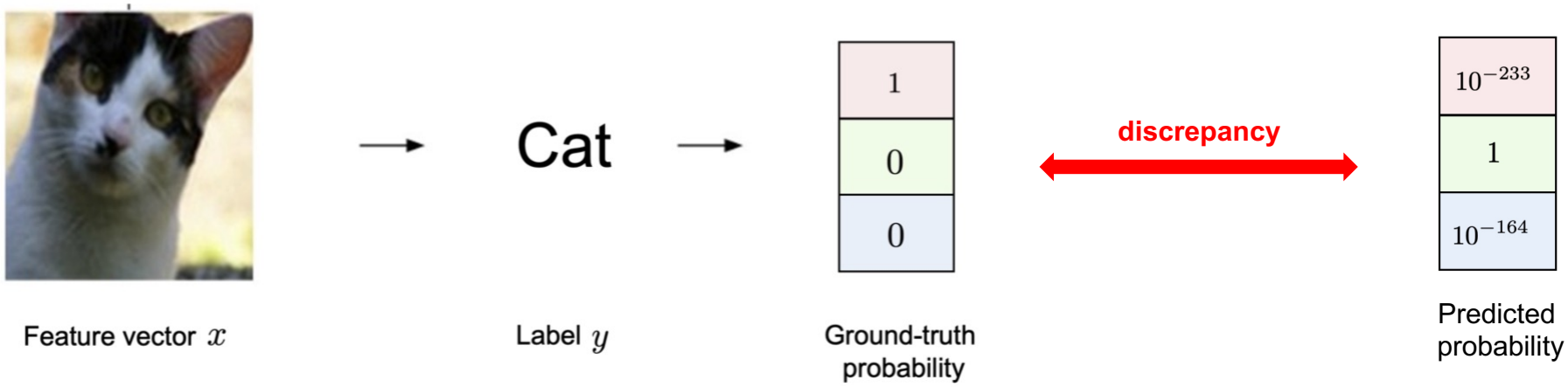


- Given the model parameters X, we can predict the class that feature h belongs to
- But a bad model X will result in incorrect predictions

Cross-entropy

- How to achieve a good model X that can provide accurate predictions?

- We need to train the model so that the discrepancy between ground-truth and predictions are minimized



- How to measure the discrepancy?

Cross-entropy

北京大学 PEKING UNIVERSITY

- Cross entropy can measure the difference between two distributions $p \in R^d$ and q $\in R^d$

$$H(\boldsymbol{p}, \boldsymbol{q}) = -\sum_{j=1}^{d} p_j \log(q_j)$$

Smaller cross entropy indicates smaller difference between p and q

- Examples:

$$\boldsymbol{p} = (1, 0, 0, 0) \quad \boldsymbol{q} = (0.25, 0.25, 0.25, 0.25) \quad \longrightarrow \quad H(\boldsymbol{p}, \boldsymbol{q}) = 2$$

$$\boldsymbol{p} = (1, 0, 0, 0) \quad \boldsymbol{q} = (0.91, 0.03, 0.03, 0.03) \quad \longrightarrow \quad H(\boldsymbol{p}, \boldsymbol{q}) = 0.136$$

Multi-classification: Loss function

北京大学 PEKING UNIVERSITY

- A good model X will minimize the discrepancy between predictions and ground-truth

$$X^\star = \arg\min_X \left\{ \frac{1}{N} \sum_{i=1}^{N} H(p_i, q_i) \right\}, \quad \text{where} \quad p_i = \text{softmax}(Xh_i)$$

prediction **ground-truth** **feature of the i-th sample**

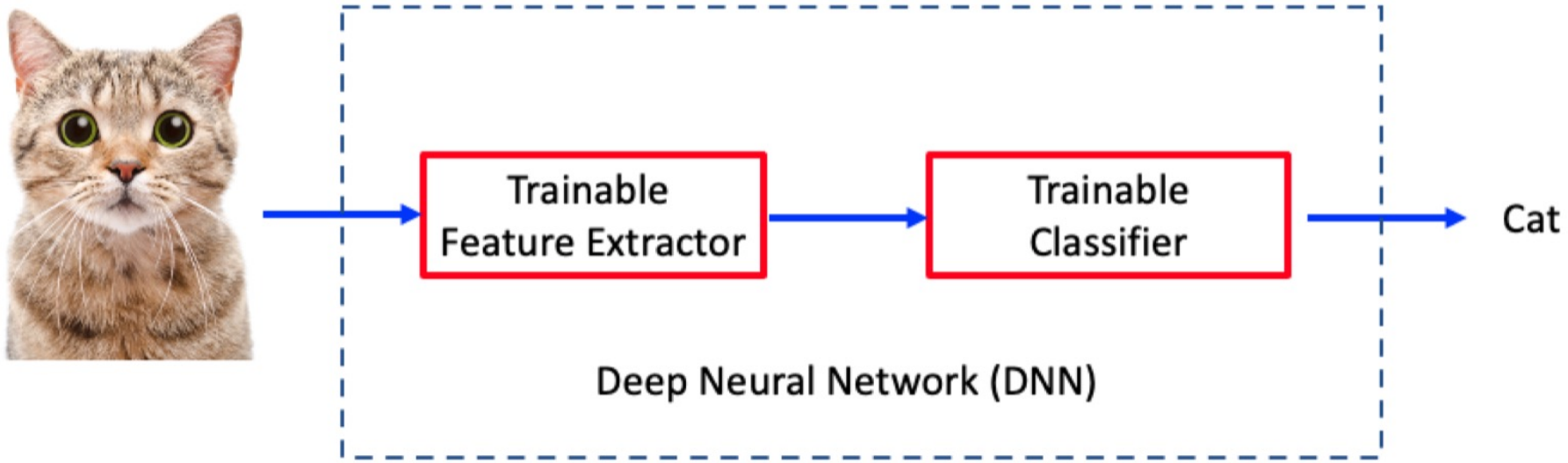
- Once a good model $X^\star$ is achieved, we can make predictions as follows

$$p = \text{softmax}(X^\star h) \longrightarrow h \text{ belongs to class } j^\star = \arg\max_j \{p_j\}$$

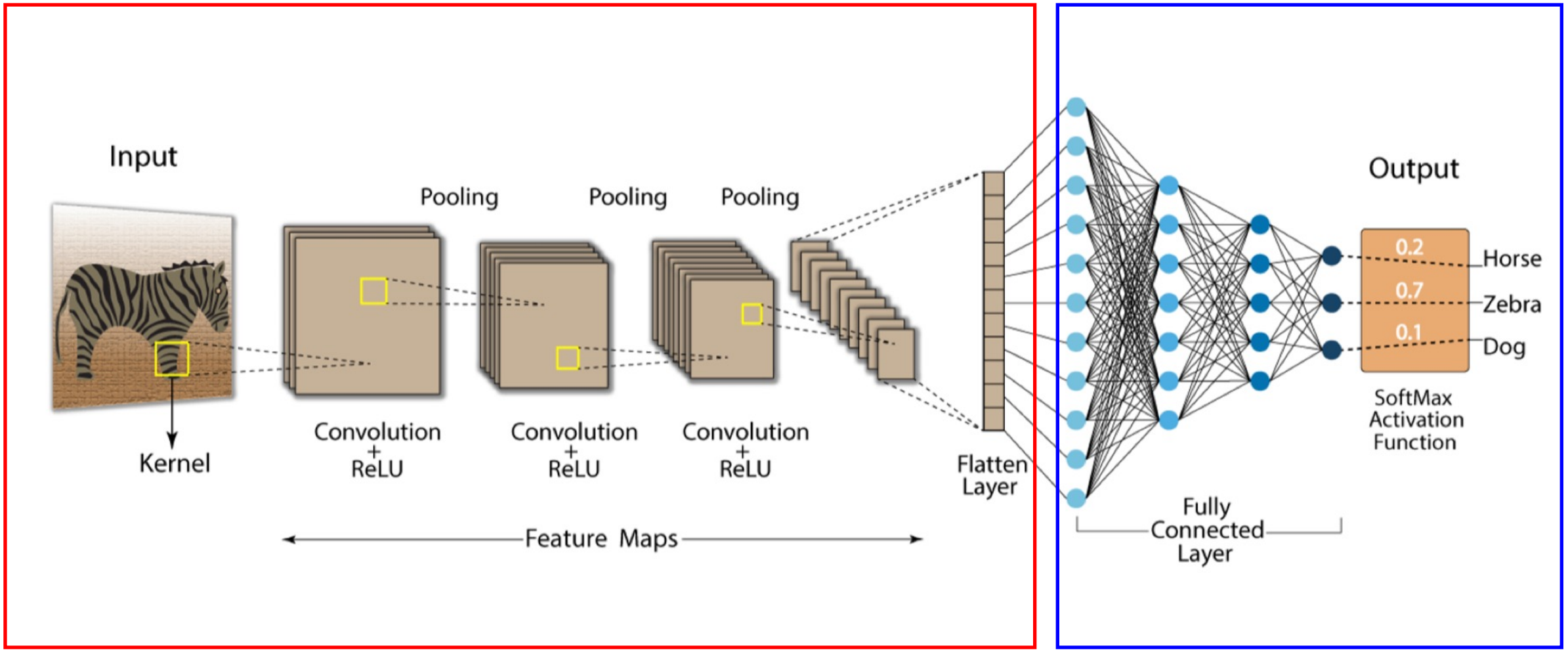
Deep neural network (DNN)

- A deep neural network (DNN) typically includes a **feature extractor** and a **classifier**

- Well-trained DNN can make precise predictions



Example: convolutional neural network

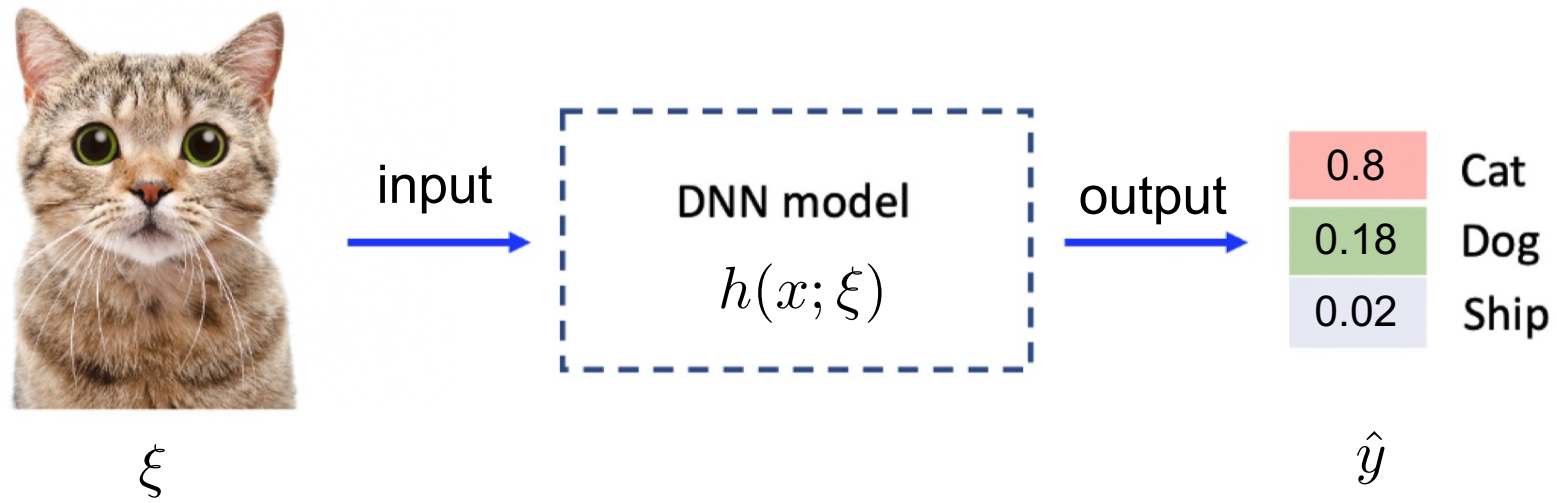


Feature extractor

Multi-Classifier

DNN model

- We model DNN as $h(x;\xi) : \mathbb{R}^d \rightarrow \mathbb{R}^c$ that maps input data ξ to a probability $\hat{y}$
  - $x \in \mathbb{R}^d$ is the DNN model parameter to be trained
  - ξ is a random input data sample
  - c is the number of classes



Training DNN can be formulated into an optimization problem

- Define $L(\hat{y}, y) = -\sum_{j=1}^{d} y_{[j]} \log(\hat{y}_{[j]})$ as the loss function to measure the difference between predictions and the ground-truth label, where $y_{[j]}$ is the j-th element in y

- The model parameter $x^\star$ can be achieved by solving the following optimization problem

$$x^\star = \arg\min_{x \in \mathbb{R}^d} \left\{ \mathbb{E}_{(\xi,y)\sim\mathcal{D}} \left[L\big(h(x;\xi), y\big) \right] \right\}$$

data distribution prediction real label

Training DNN can be formulated into an optimization problem

北京大学 PEKING UNIVERSITY

- The model parameter $x^\star$ can be achieved by solving the following optimization problem

$$x^\star = \arg\min_{x\in\mathbb{R}^d} \left\{ \mathbb{E}_{(\xi,y)\sim\mathcal{D}} \Big[L\big(h(x;\xi), y\big) \Big] \right\}$$

data distribution prediction real label

- If we define $\boldsymbol{\xi} = (\xi, y)$ and $F(x;\boldsymbol{\xi}) = L(h(x;\xi), y)$, the above problem becomes

$$x^\star = \arg\min_{x\in\mathbb{R}^d} \left\{ \mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{D}} [F(x;\boldsymbol{\xi})] \right\}$$

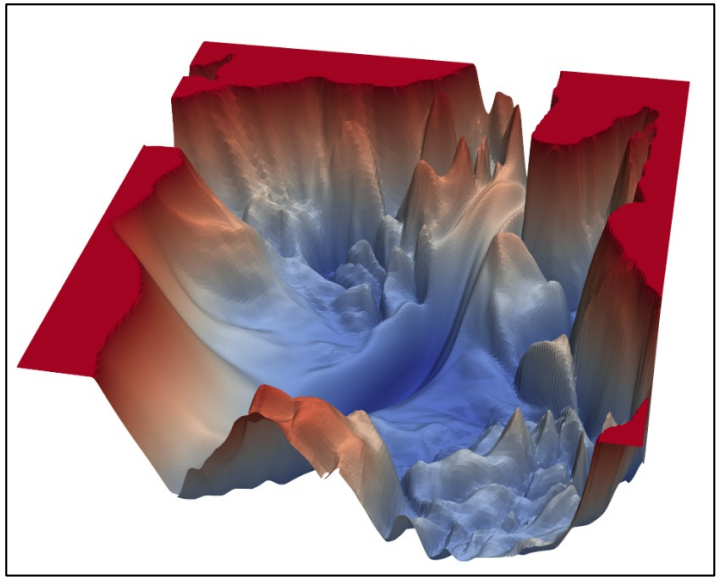
- Most optimization researchers use the second formulation as the staring point to develop algorithms

DNN model is notoriously difficult to train

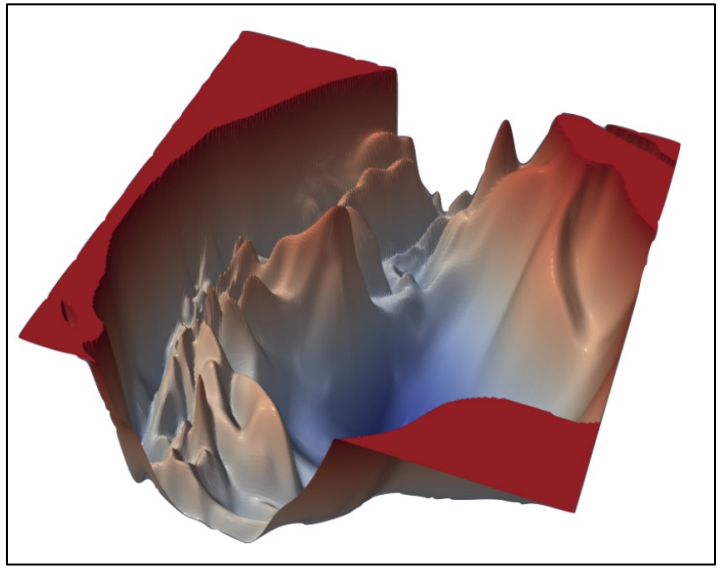
北京大学 PEKING UNIVERSITY

- Highly-nonconvex cost functions; cannot find global minima; trapped into local minimum

VGG-56



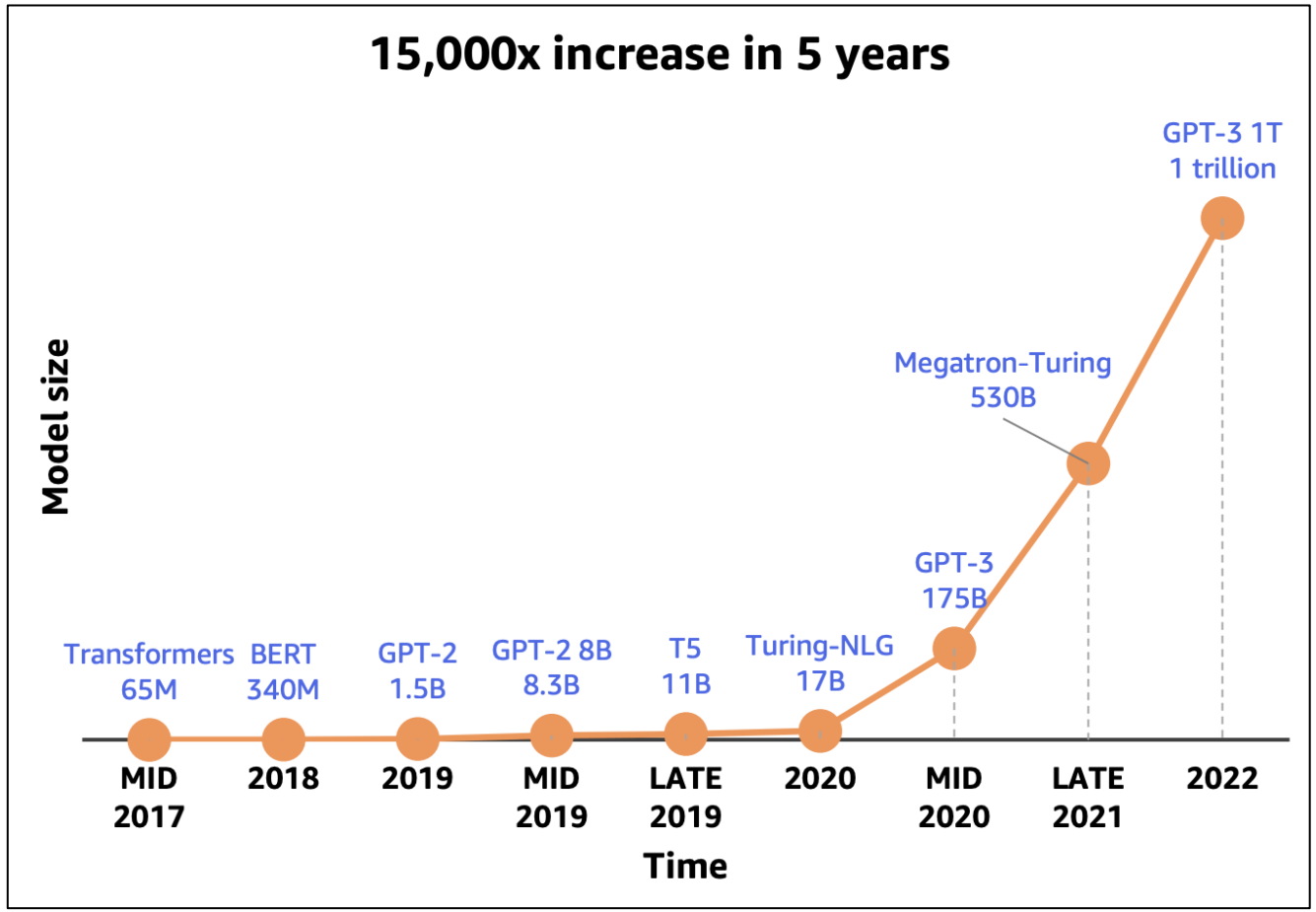
VGG-110



[Visualizing the loss landscape of neural nets]

DNN model is notoriously difficult to train

- The model size is large, i.e., $x \in \mathbb{R}^d$ is of extremely high dimensions



[Train and deploy large language models on Amazon SageMaker]

DNN model is notoriously difficult to train

北京大学 PEKING UNIVERSITY

- The size of the dataset is huge

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Table 1: **Pre-training data.** Data mixtures used for pre-training, for each subset we list the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion.

Crawled data from websites; in both high quality and low quality

High-quality data

[LLaMA: Open and Efficient Foundation Language Models]

DNN model is notoriously difficult to train

DNN training = Non-convex training + Huge dimensions + Huge dataset

Efficient and **scalable** optimization approaches are in urgent need

北京大学
PEKING UNIVERSITY

Part 02-1

Basic optimizers: Stochastic gradient descent

Stochastic optimization

北京大学 PEKING UNIVERSITY

- Recall that DNN model can be formulated into the stochastic optimization problem [see page 16]

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

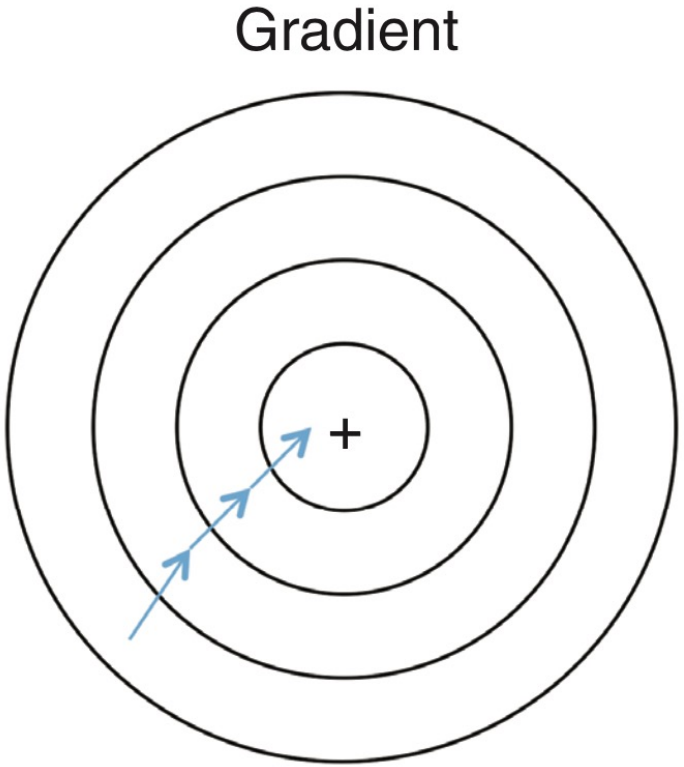
  - ξ is a random variable indicating data samples
  - D is the data distribution; unknown in advance
  - $F(x; \xi)$ is differentiable in terms of x

Gradient descent

- Suppose data distribution D is **known** and the real gradient $\nabla f(x)$ is **available**, the gradient descent method iterates as follows

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

where $\gamma > 0$ is the learning rate



Stochastic gradient descent (SGD)

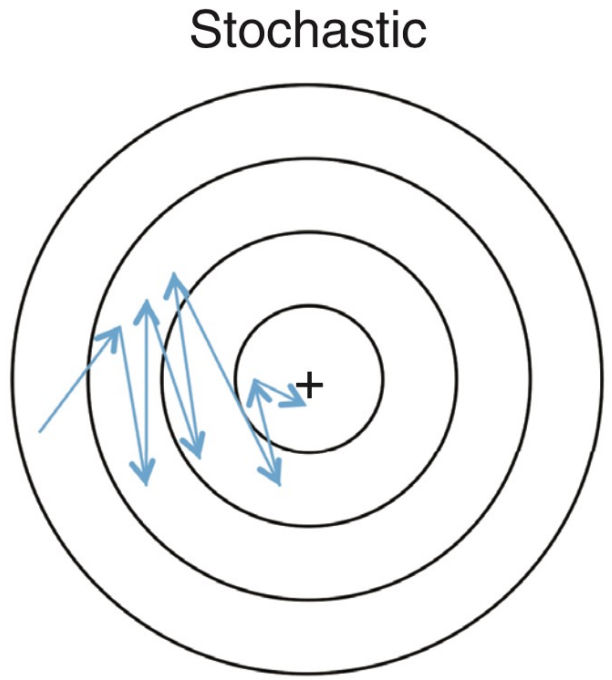
北京大学 PEKING UNIVERSITY

- If data distribution D is **unknown** and the real gradient $\nabla f(x)$ is **unavailable**, stochastic gradient descent iterates as follows

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

where $\gamma > 0$ is the learning rate

- Stochastic gradient $\nabla F(x_k; \xi_k)$ is a stochastic estimate of the real gradient $\nabla f(x_k)$



SGD convergence

Assumption 1. For any iteration k, we assume that

$$\mathbb{E}[\nabla F(x_k;\xi_k)] = \nabla f(x_k) \qquad \text{(unbiased)}$$

$$\mathbb{E}\|\nabla F(x_k;\xi_k) - \nabla f(x_k)\|^2 \leq \sigma^2 \qquad \text{(bounded variance)}$$

This assumption guarantees that each stochastic gradient $\nabla F(x_k;\xi_k)$ is a **nice** estimation of the real gradient $\nabla f(x_k)$ per iteration k

SGD convergence

Theorem 1

Suppose $f(x)$ is L-smooth and Assumption 1 holds. If γ is chosen as

$$\gamma = \left[\left(\frac{2\Delta_0}{(K+1)L\sigma^2} \right)^{-\frac{1}{2}} + L \right]^{-1},$$

SGD will converge at the following rate

$$\frac{1}{K+1} \sum_{k=0}^{K} \mathbb{E}[\|\nabla f(x_k)\|^2] \leq \sqrt{\frac{8L\Delta_0\sigma^2}{K+1}} + \frac{2L\Delta_0}{K+1}.$$

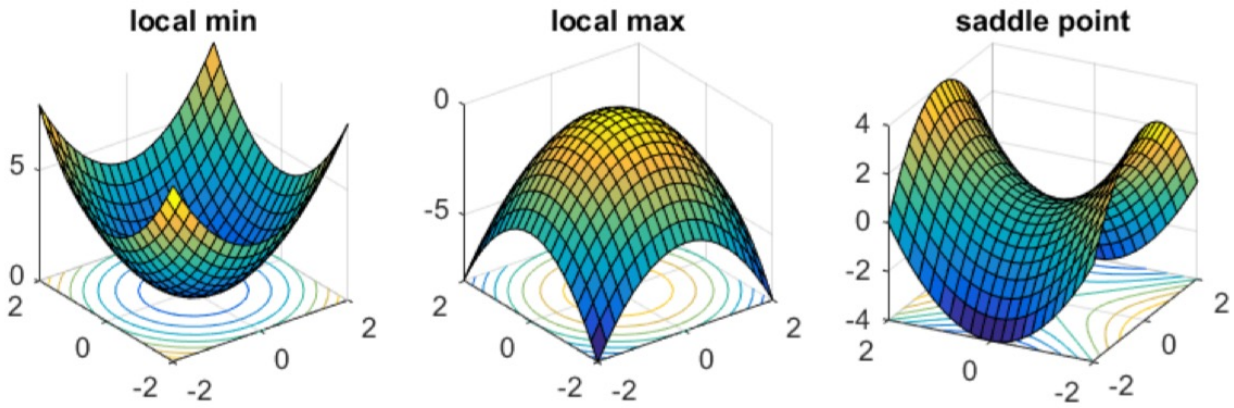
where $\Delta_0 = f(x_0) - f^\star$.

- Decaying rate leads to exact convergence to stationary point
- When $\sigma^2 = 0$, the above rate reduces to GD; rate is tight !
- $O(\sqrt{\sigma^2/K})$ is the dominant rate

SGD convergence

$$\frac{1}{K+1}\sum_{k=0}^{K}\mathbb{E}\|\nabla f(x_k)\|^2 = O\left(\sqrt{\frac{L\sigma^2}{K+1}}+\frac{L}{K+1}\right)$$

- When iteration $K \to \infty$, it holds that $\mathbb{E}\|\nabla f(x_K)\|^2 \to 0$
- $\mathbb{E}\|\nabla f(x_K)\|^2 \to 0$ implies SGD converges to a stationary solution
- A stationary solution can be local min, local max, or saddle point[4]



[From Prof. Rong Ge's online post]

SGD convergence

北京大学 PEKING UNIVERSITY

- Generally speaking, approaching the stationary solution is the best result we can get for SGD; no guarantee to approach the global minimum

- Empirically, SGD performs extremely well when training DNN

- Recent advanced studies show SGD can escape local maximum, saddle point, and even “sharp” local minimum, see, e.g., (Ge et al., 2015; Sun et al., 2015; Jin et al., 2017; Du et al., 2018, 2019; Kleinberg et al., 2018)

- SGD even finds global minimum under certain conditions, e.g. the PL condition (Karimi et al., 2016)

  However, we will skip these interesting results in this lecture

BackPropagation

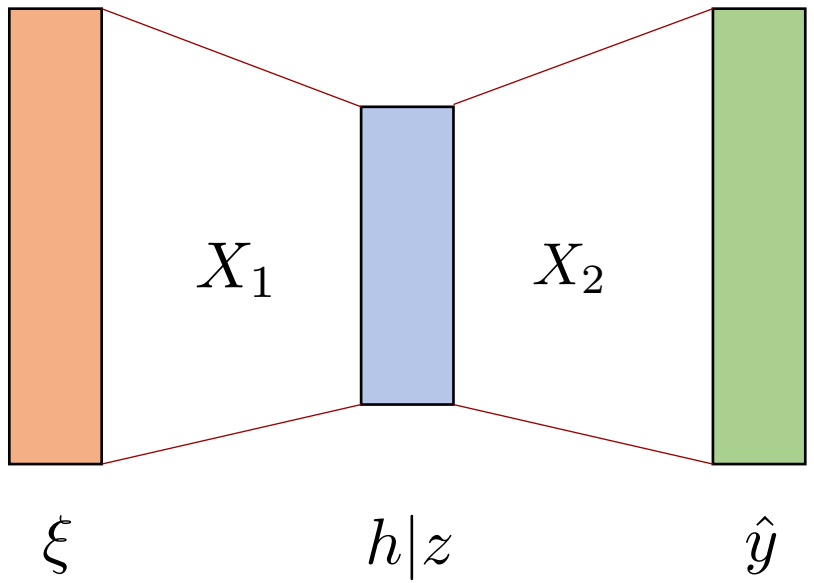
- When training DNN, an efficient implementation for SGD is named **backpropagation**

- Backpropagation essentially calculates the stochastic gradient using the chain rule

- Consider the linear neural network

$$f(X_1, X_2; \xi) = L\big(X_2 \cdot \sigma(X_1 \xi)\big)$$

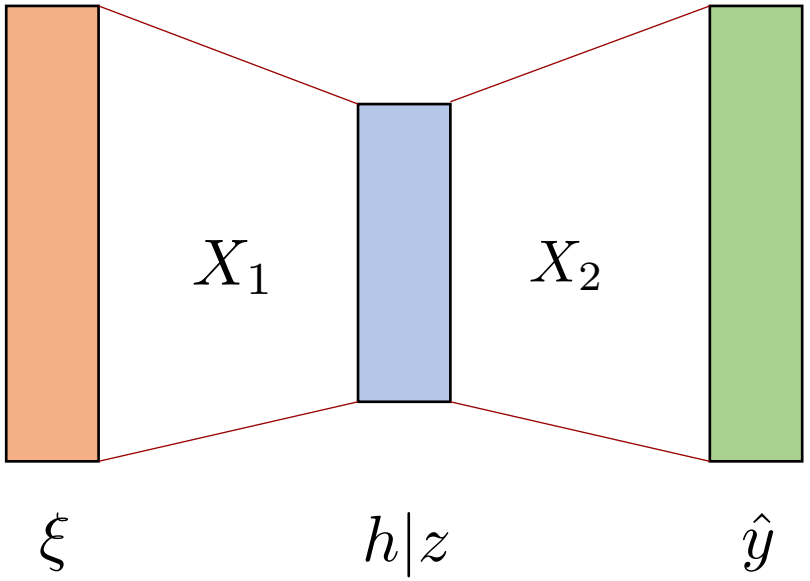
How to calculate $\partial f / \partial X_1$ and $\partial f / \partial X_2$?



dims: d (p,d) p (q,p) q

BackPropagation

$$f(X_1, X_2; \xi) = L\big(X_2 \cdot \sigma(X_1 \xi)\big)$$



dims: d (p,d) p (q,p) q

$$h = X_1 \xi$$

$$z = \sigma(h)$$

$$\hat{y} = X_2 z$$

$$f = L(\hat{y})$$

Forward

$$\frac{\partial f}{\partial X_1} = \frac{\partial f}{\partial h} \xi^T$$

$$\frac{\partial f}{\partial h} = \frac{\partial f}{\partial z} \odot \nabla \sigma(h)$$

$$\frac{\partial f}{\partial X_2} = \frac{\partial L}{\partial \hat{y}} z^T, \quad \frac{\partial f}{\partial z} = X_2^T \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial f}{\partial \hat{y}} = \nabla L(\hat{y})$$

Backward

Store h, z and $\hat{y}$

Store $\nabla_{X_1} f(X_1)$ and $\nabla_{X_2} f(X_2)$

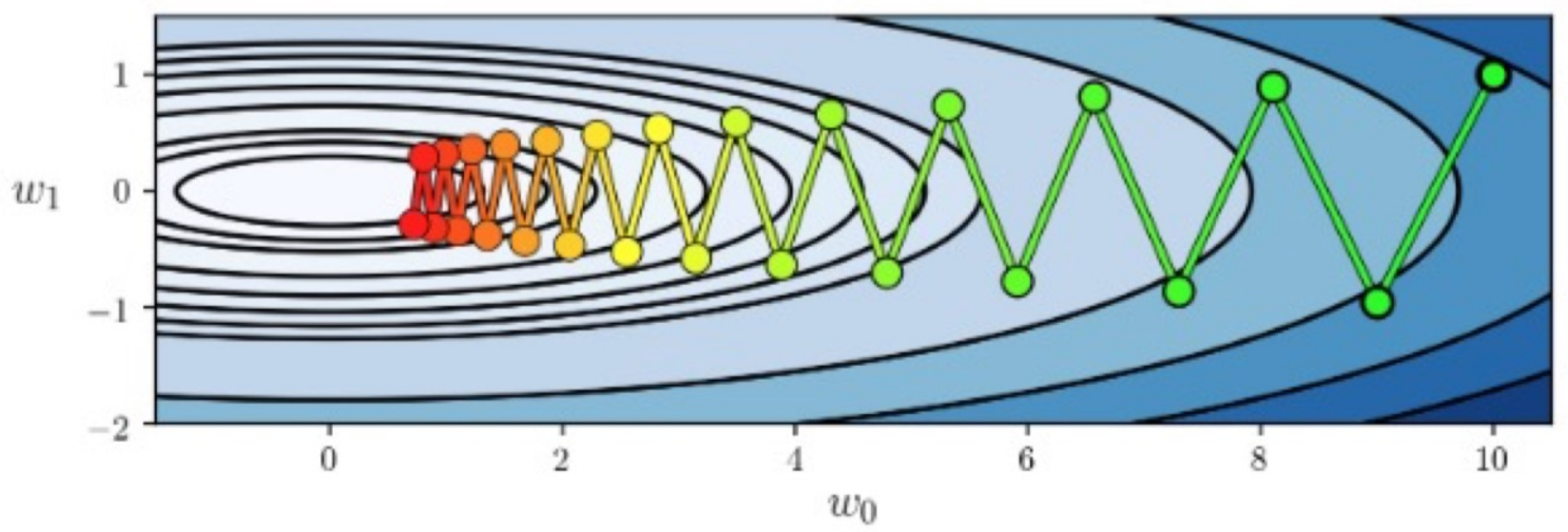
北京大学
PEKING UNIVERSITY

Part 02-2

Basic optimizers: Momentum SGD

Gradient descent can be slow

- Gradient descent can be very slow for ill-conditioned problems



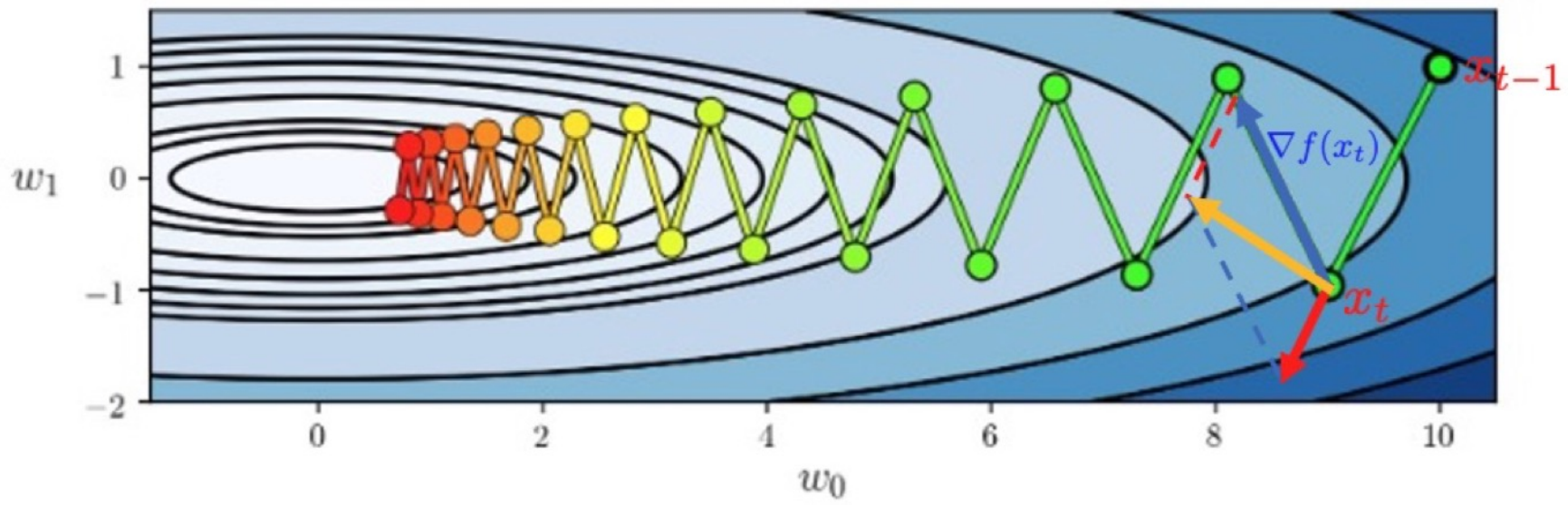
[Machine Learning Refined]

Gradient descent with Polyak's momentum

- We have to alleviate the "Zig-Zag" to accelerate the algorithm
- **Polyak's momentum** method, a.k.a, **heavy-ball** gradient method

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

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



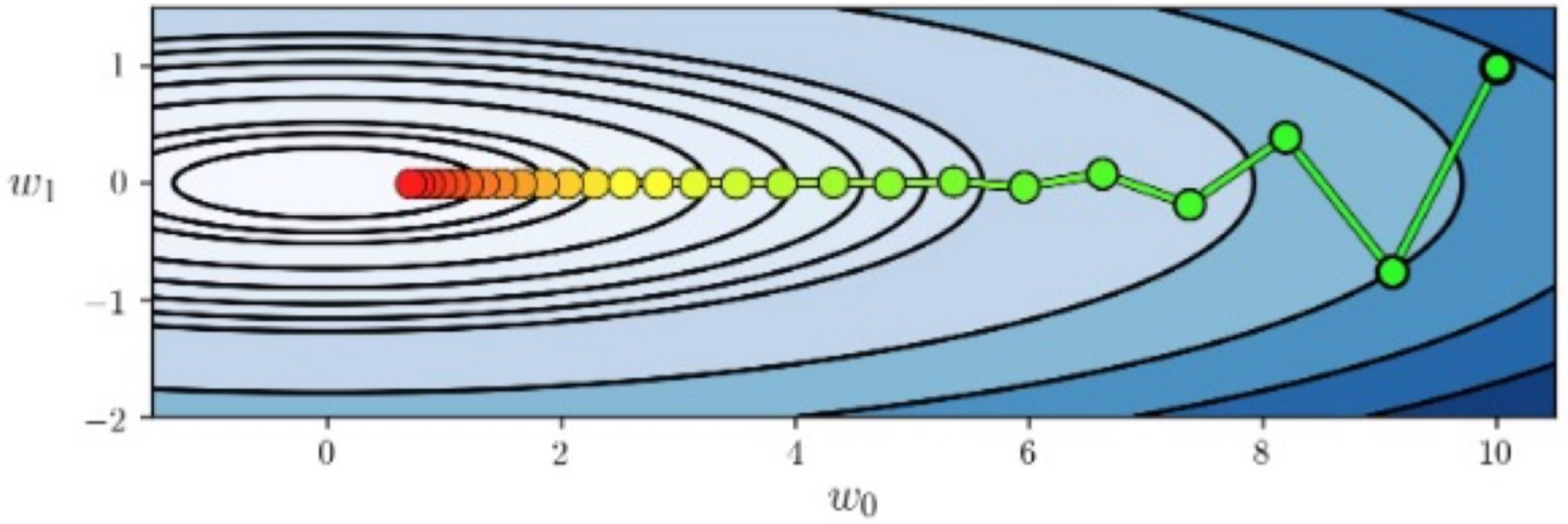
Gradient descent with Polyak's momentum

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

- **Polyak's momentum** method, a.k.a, **heavy-ball** gradient method

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

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



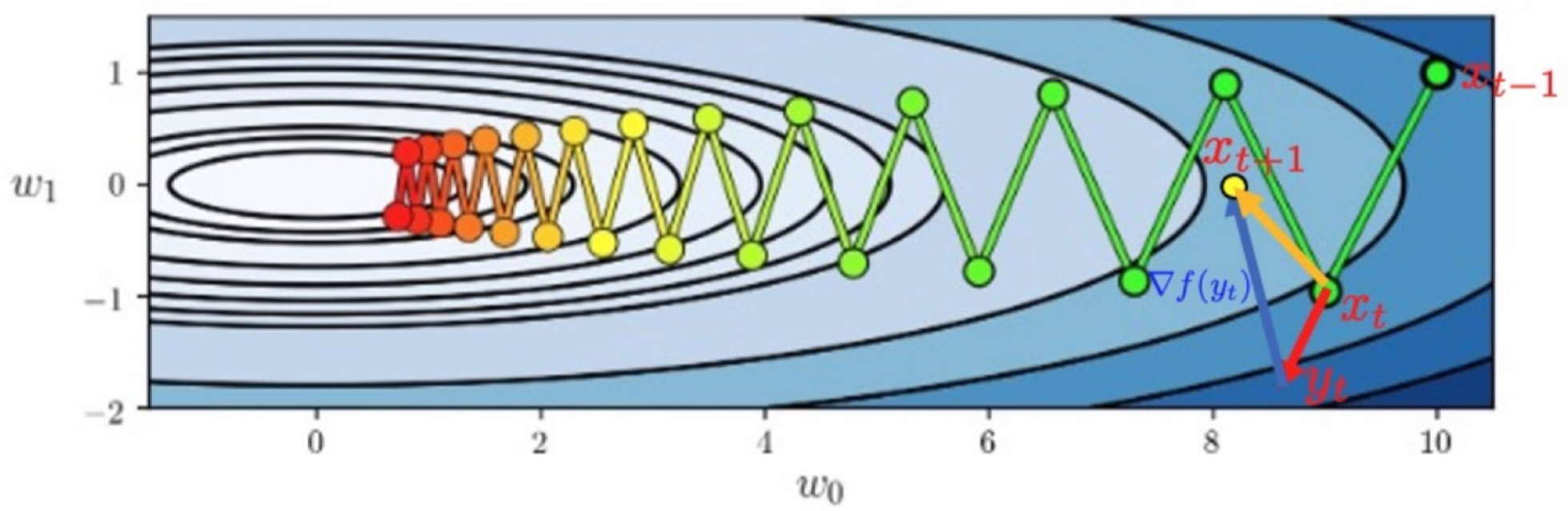
Gradient descent with Polyak's momentum

- Gradient descent with Nesterov's momentum, a.k.a, Nesterov accelerated gradient (NAG) method

$$y_{k-1} = x_{k-1} + \beta(x_{k-1} - x_{k-2})$$

$$x_k = y_{k-1} - \gamma \nabla f(y_{k-1})$$

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



Momentum SGD

北京大学 PEKING UNIVERSITY

- Recall the stochastic optimization

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

- The standard SGD algorithm is

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

- Momentum SGD algorithm is

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

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

Momentum SGD

- Momentum SGD can be rewritten into

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

where $m_0 = 0$. This is how PyTorch implements it.

- To see it, we have the following recursion from the second line

$$\beta x_k = \beta x_{k-1} - \gamma \beta m_{k-1}.$$

Subtract it from the second line, we have

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

Same as momentum SGD in the last page

北京大学
PEKING UNIVERSITY

Part 02-3

Basic optimizers: Adaptive SGD

北京大学
PEKING UNIVERSITY

Thank you!

Kun Yuan homepage: https://kunyuan827.github.io/