



动手学深度学习 v2

李沐 · AWS





目标

- 介绍深度学习经典和最新模型
 - LeNet, ResNet, LSTM, BERT, ...
- 机器学习基础
 - 损失函数、目标函数、过拟合、优化
- 实践
 - 使用Pytorch实现介绍的知识点
 - 在真实数据上体验算法效果



内容

- 深度学习基础 — 线性神经网络，多层感知机
- 卷积神经网络 — LeNet, AlexNet, VGG, Inception, ResNet
- 循环神经网络 — RNN, GRU, LSTM, seq2seq
- 注意力机制 — Attention, Transformer
- 优化算法 — SGD, Momentum, Adam
- 高性能计算 — 并行，多GPU，分布式
- 计算机视觉 — 目标检测，语义分割
- 自然语言处理 — 词嵌入，BERT



形式

- 每周六、日下午1: 00—2: 30直播
- 每次直播讲~4个小节
- 每个小节讲算法（10min），代码实现（10min），问答（5min）
- 每个月一次Kaggle竞赛
- 视频回放在B站和Youtube



你将学到什么

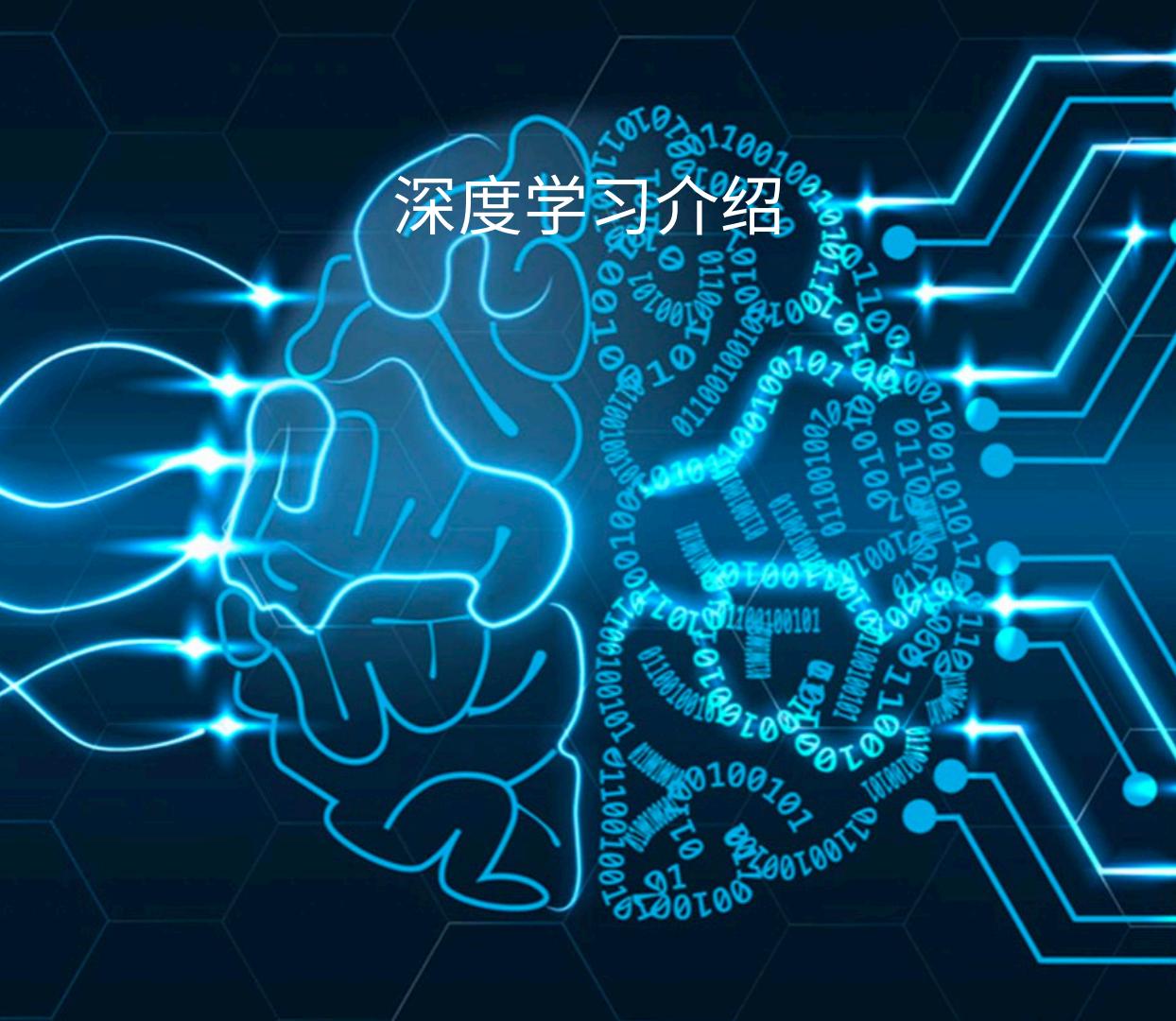
- What
 - 深度学习里有哪些技术
- How
 - 如何实现和调参
- Why
 - 背后的原因为 (直觉、数学)





资源

- 课程主页：<https://courses.d2l.ai/zh-v2>
- 教材：<https://zh-v2.d2l.ai/>
- 课程论坛讨论：<https://discuss.d2l.ai/c/16>
- Pytorch论坛：<https://discuss.pytorch.org/>

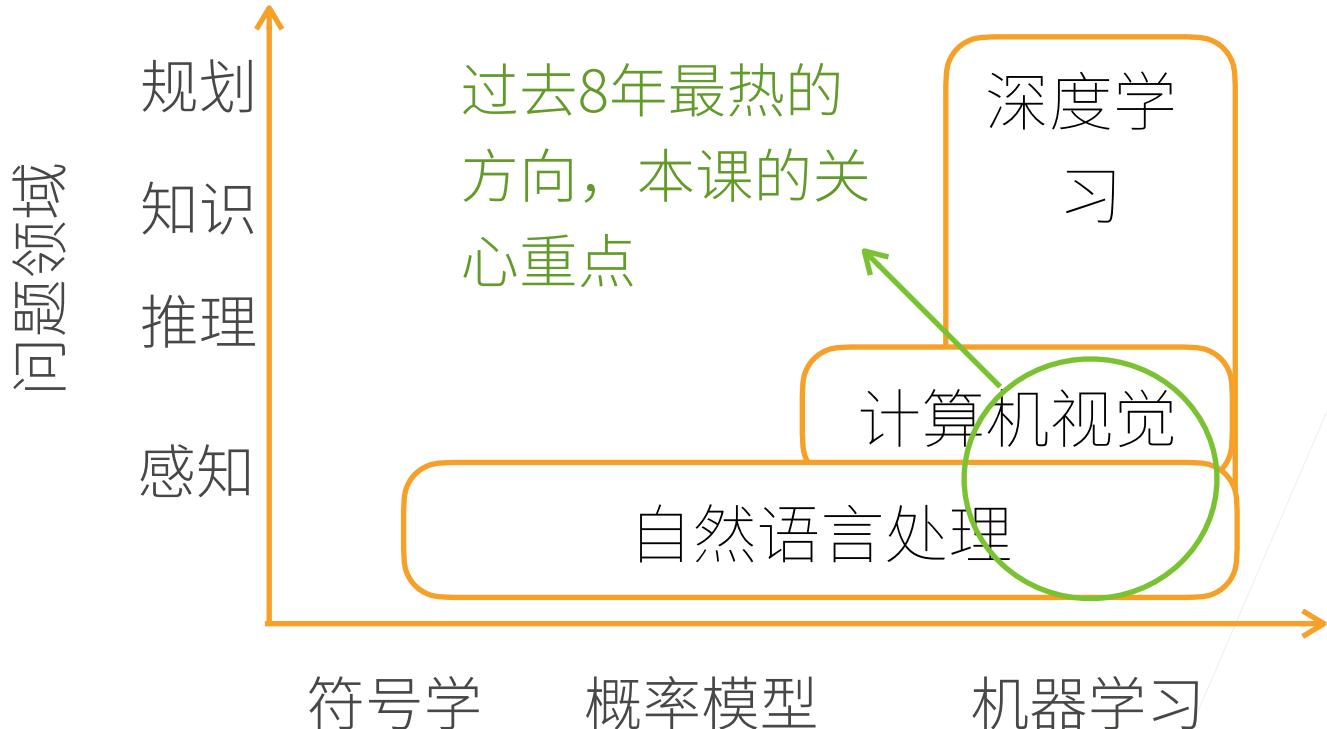


动手学深度学习 v2

李沐 · AWS



AI 地图





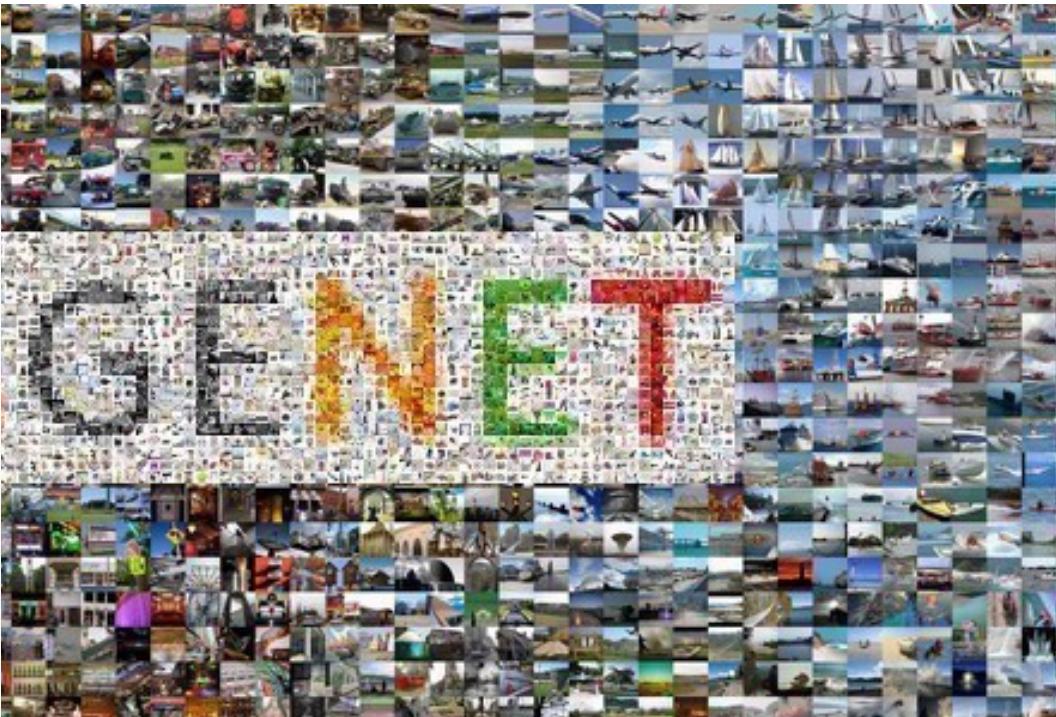
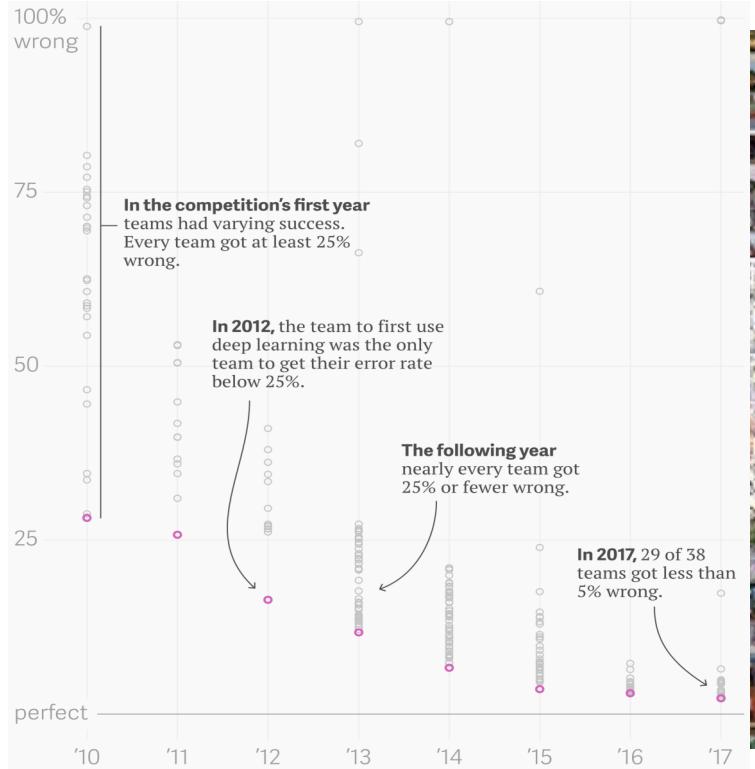
图片分类



<http://www.image-net.org/>



图片分类



<https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/>



物体检测和分割



https://github.com/matterport/Mask_RCNN

动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>



样式迁移





人脸合成





文字生成图片

TEXT PROMPT

an illustration of a baby daikon radish in a tutu walking a dog

AI-GENERATED IMAGES



TEXT PROMPT

an armchair in the shape of an avocado [...]

AI-GENERATED IMAGES



<https://openai.com/blog/dall-e/>



文字生成

Q: How to run an Effective Board Meeting

Step 1: Spend time building an effective board

Startup boards typically include one or more founders representing the common shareholders of the financing round), and one or more independent board directors. Importantly, these independent directors have more reputational skin in the game, actual legal liability, and critical feedback to the CEO.

Most founders I've met don't appreciate the significant impact that high quality board members can have on high quality, however that's not always entirely in your control, since it's dependent on how hot you are picking you. The quality of your independent board members is far more in your control. It can take a lot of effort from a board member but the effort is absolutely worth it.

When you start recruiting independent board members can vary from startup to startup, I typically delay it if your current board is decent enough and covers the areas of knowledge critical to your startup.

A while back, one of the startups we'd invested in had seen some early success with large enterprise clients but encountered some really tough challenges. Mostly because the DNA of the founders was primarily spinning their wheels and rotating through heads of sales, the founders decided to try and accelerate the search for an expert in enterprise sales. The effects were immediate. Within weeks they were making headway on deals and team delivering results. Over the following few years, this independent board member was critical to the startup's success.

Founders also tend to not realize they can stretch even higher in recruiting board members than in-house. It's incredibly flattering, a relatively low time commitment, and the topic can be broached with very little tact. You can shoot for the stars. Typically compensation for an external board member like this should be the current stage, with immediate monthly vesting; no cliff.

Here's `#gpt3` writing some SQL for me.

Text: Select the "Students" from the "School" table join

Code: `SELECT * FROM Students`

`INNER JOIN Class`

`ON Students.ID = Class.StudentID`

无人驾驶



案例研究 – 广告点击



All ▾ baby toy Hello, Mu Account & Lists Orders Prime Cart

Deliver to Mu
Palo Alto 94303 Fresh ▾ Whole Foods Today's Deals Help Best Sellers

Shop deals for the New Year

Sort by: Featured ▾

1-48 of over 30,000 results for "baby toy"

Amazon Prime

prime prime | FREE Same-Day

Get FREE Same-Day Delivery on qualifying orders over \$35

Delivery Day

Get It Today Get It by Tomorrow

Pantry

prime pantry

Prime Wardrobe

prime wardrobe

Local Stores

Amazon Fresh Whole Foods Market

Department

Toys & Games
Baby & Toddler Toys
Baby Musical Toys
Stuffed Animals & Plush Toys
Baby Rattles & Plush Rings
Baby Activity Play Centers
See more

Baby Products
Baby Teether Toys
Baby Bibs
Baby Toothbrushes
Baby Health Care Products

Books
Children's Vocabulary & Spelling Books
Children's Basic Concepts Books

SPONSORED BY SIMILAC

#1 brand chosen by parents [Shop now](#)



Similac Pro-Advance Non-GMO Infant Formula with Iron, with 2'-FL HMO, Baby Formula, Powder, 36 oz, 12 Count (One Month Supply)

Similac Pro-Total Comfort Infant Formula OPTI-GRO, Non-GMO, Baby Formula, Powder, 36 oz, 12 Count (One Month Supply)

5★ 428 Ad feedback

5★ 188 Ad feedback

Price and other details may vary based on size and color


Sponsored ⓘ Baby Einstein Magic Touch Mini Piano Wooden Musical Toy, 3 Months + \$9.99


Sponsored ⓘ Splashin'kids Infant Toys Beginner Crawler Game Ball Drop Maze Tummy Time Activity Center Early Development Jum... \$19.99


Sponsored ⓘ iPlay, iLearn 5 Dinosaur Baby Rattles, Teether, Shaker, Grab and Spin Rattle, Musical Toy Set, Early Educational Toys, Unique Gifts... \$19.99

4 stars and above Sponsored

Page 1 of 3


TOP BRIGHT Toddler Toys For 1 2 Year Old Boy And Girl Gifts Wooden Race Track Car R... \$21.99 Ad feedback


HOMOFY Homof Baby Toys Musical Learning Table 6 Months Up-Early Education Music Act... \$26.99 Ad feedback


RenFox Kids Musical Mats, Music Piano Keyboard Dance Floor Mat Carpet Animal Blanket... \$36.99 Ad feedback


fisca 3 in 1 Musical Instruments Toys, Electronic Piano Keyboard Xylophone Drum Set... \$15.89 Ad feedback


LIGHTDESIRE Baby Toys Musical Caterpillar, Infant Toys Crinkle Rattle Soft with Rin... \$21 Ad feedback

Page 1 of 254

Sponsored products related to this item


Baby Einstein Magic Touch Curiosity Tablet Wooden Musical Toy, 6 Months + \$19.99 Ad feedback


BAOLI Baby Bear Bell Toy Music Song Light (Colors May Vary Pink/Green/Yellow) \$9.99 Ad feedback


london-kate Wooden Musical Table - Xylophone, Drum, Maraca and 2 Beaters \$27.58 Ad feedback


Woby Multifunctional Musical Elephant Keyboard Piano Educational Learning Light Up... \$19.98 Ad feedback


NextX Baby Toys 6 to 12 Months Infant Musical Learning Toys \$26.99 Ad feedback



案例研究 – 广告点击

触发



1

点击率预估

$$p = 0.11 \quad p = 0.08 \quad p = 0.03 \quad p = 0.02$$



2

$$p = 0.06 \quad p = 0.12 \quad p = 0.02 \quad p = 0.04$$



3

排序 (点击率 x 竞价)





预测与训练



Baby Einstein Magic Touch
Curiosity Tablet Wooden
Musical Toy, 6 Months +
★★★★★ 25
\$19.99 prime

特征提取

广告主

产品描述

产品图片

模型

点击率预测

$p=0.11$

训练数据
(过去广告展现和用户点击)



特征和用户点击



模型





完整的故事

领域专家

点击



展现



模型控制广告展现，这些又用来训练新的模型



数据



模型



提升模型精度和性能

数据科学家

AI 专家



动手学深度学习 v2

李沐 · AWS





本地安装

- [可选]使用 conda/miniconda 环境

```
conda env remove d2l-zh
```

```
conda create -n -y d2l-zh python=3.8 pip
```

```
conda activate d2l-zh
```

- 安装需要的包

```
pip install -y jupyter d2l torch torchvision
```

- 下载代码并执行

```
wget https://zh-v2.d2l.ai/d2l-zh.zip
```

```
unzip d2l-zh.zip
```

```
jupyter notebook
```



数据操作



N维数组样例

- N维数组是机器学习和神经网络的主要数据结构

0-d (标量)



1.0

一个类别

1-d (向量)



[1.0, 2.7, 3.4]

一个特征向量

2-d (矩阵)



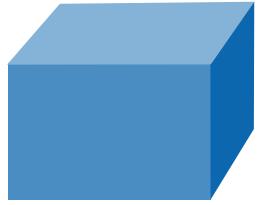
[[1.0, 2.7, 3.4],
 [5.0, 0.2, 4.6],
 [4.3, 8.5, 0.2]]

一个样本—特征矩阵



N维数组样例 (续)

3-d



```
[[[0.1, 2.7, 3.4]  
 [5.0, 0.2, 4.6]  
 [4.3, 8.5, 0.2]]  
 [[3.2, 5.7, 3.4]  
 [5.4, 6.2, 3.2]  
 [4.1, 3.5, 6.2]]]
```

RGB图片 (宽x
高x通道)

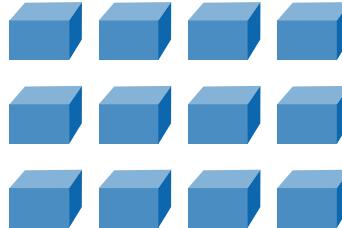
4-d



```
[[[[. . .  
 . . .  
 . . .]]]
```

一个RGB图片
批量 (批量大小
x宽x高x通道)

5-d



```
[[[[. . .  
 . . .  
 . . .]]]
```

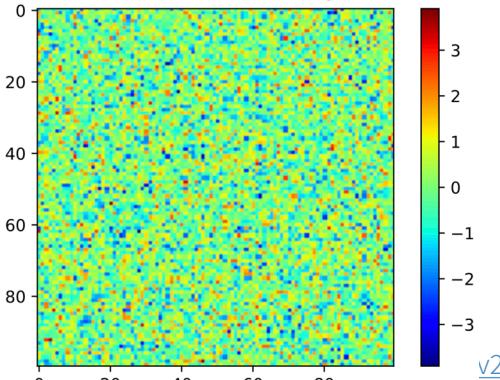
一个视频批量 (批量大
小x时间x宽x高x通道)



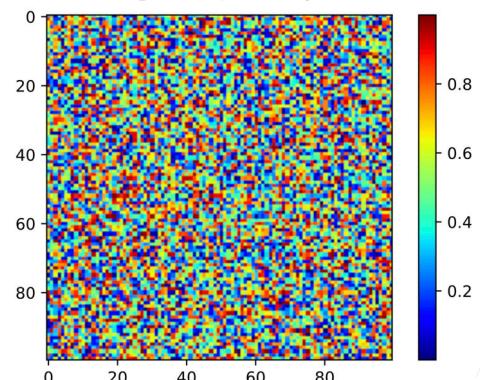
创建数组

- 创建数组需要
 - 形状：例如 3×4 矩阵
 - 每个元素的数据类型：例如32位浮点数
 - 每个元素的值，例如全是0，或者随机数

正态分布



均匀分布





访问元素

一个元素: [1, 2]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

一行: [1, :]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

一列: [:,1]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

子区域: [1:3, 1:]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

子区域: [::3, ::2]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16



线性代数

$Ax = \lambda x$ "Eigen Vector"
 ↑ "Eigen Value"
 $n \times n$ Matrix scalar

$(A - \lambda I)x = 0$
 same thing
 $\det(A - \lambda I) = 0$ ← "characteristic equation of A"
 $(A - \lambda I)$ is singular
 $(A - \lambda I)$ is not invertable

$P(\lambda) = \det(A - \lambda I)$ "characteristic polynomial"

$A = \begin{bmatrix} 4 & 8 \\ 6 & 26 \end{bmatrix}$ $A - I = \begin{bmatrix} 4 & 8 \\ 6 & 26 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 4-1 & 8 \\ 6 & 26-1 \end{bmatrix} = \begin{bmatrix} 3 & 8 \\ 6 & 25 \end{bmatrix}$

$\lambda I = \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$

$A - \lambda I = \begin{bmatrix} 4 & 8 \\ 6 & 26 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} = \begin{bmatrix} 4-\lambda & 8 \\ 6 & 26-\lambda \end{bmatrix}$

$\det(A - \lambda I) = \begin{vmatrix} 4-\lambda & 8 \\ 6 & 26-\lambda \end{vmatrix} = (4-\lambda)(26-\lambda) - (8)(6) = 104 - 30\lambda + \lambda^2 - 48 = \lambda^2 - 30\lambda + 56 = (\lambda - 28)(\lambda - 2) = 0$

$\lambda_1 = 28$
 $\lambda_2 = 2$

$N(A - \lambda_1 I) = \begin{bmatrix} 4-28 & 8 \\ 6 & 26-28 \end{bmatrix} = \begin{bmatrix} 4-2 & 8 \\ 6 & 26-2 \end{bmatrix} = \begin{bmatrix} 2 & 8 \\ 6 & 24 \end{bmatrix} \xrightarrow[R2/6]{R1/4} \begin{bmatrix} 1 & 4 \\ 1 & 4 \end{bmatrix} \xrightarrow[F2-R1]{R2/4} \begin{bmatrix} 1 & 4 \\ 0 & 0 \end{bmatrix}$

$x_1 + 4x_2 = 0 \rightarrow x_1 = -4x_2 \rightarrow \begin{bmatrix} -4x_2 \\ x_2 \end{bmatrix} \xrightarrow[Eigen space]{} \begin{bmatrix} -4 \\ 1 \end{bmatrix} \rightarrow x \begin{bmatrix} -4 \\ 1 \end{bmatrix}$ Eigen Vector



标量

- 简单操作

$$c = a + b$$

$$c = a \cdot b$$

$$c = \sin a$$

- 长度

$$|a| = \begin{cases} a & \text{if } a > 0 \\ -a & \text{otherwise} \end{cases}$$

$$|a + b| \leq |a| + |b|$$

$$|a \cdot b| = |a| \cdot |b|$$



向量

- 简单操作

$$c = a + b \quad \text{where } c_i = a_i + b_i$$

$$c = \alpha \cdot b \quad \text{where } c_i = \alpha b_i$$

$$c = \sin a \quad \text{where } c_i = \sin a_i$$

- 长度

$$\|a\|_2 = \left[\sum_{i=1}^m a_i^2 \right]^{\frac{1}{2}}$$

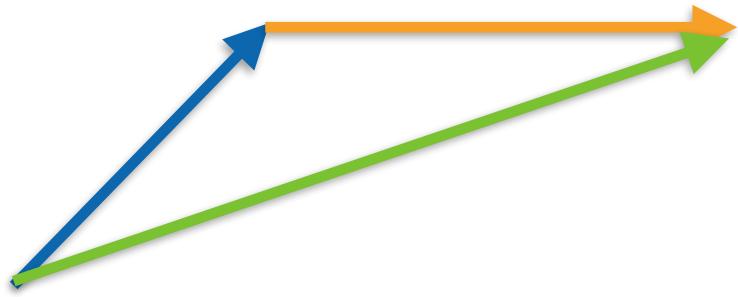
$$\|a\| \geq 0 \text{ for all } a$$

$$\|a + b\| \leq \|a\| + \|b\|$$

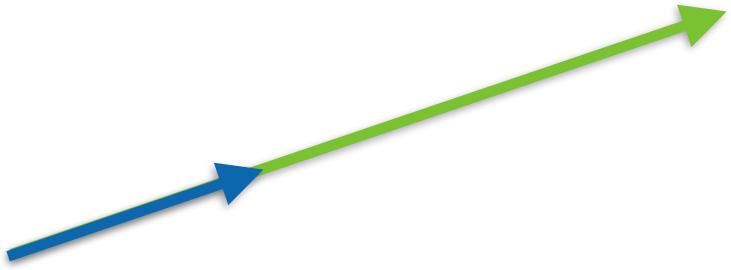
$$\|a \cdot b\| = |a| \cdot \|b\|$$



向量



$$c = a + b$$



$$c = \alpha \cdot b$$

数学家的 '**parallel for all do**'



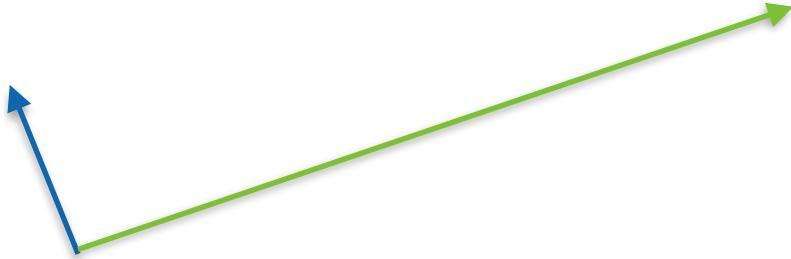
向量

- 点乘

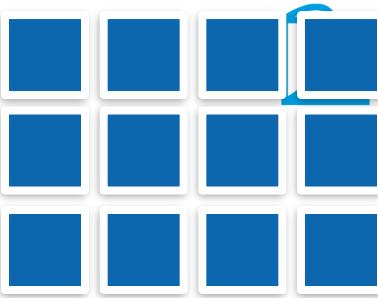
$$a^\top b = \sum_i a_i b_i$$

- 正交

$$a^\top b = \sum_i a_i b_i = 0$$



矩阵



- 简单操作

$$C = A + B$$

where $C_{ij} = A_{ij} + B_{ij}$

$$C = \alpha \cdot B$$

where $C_{ij} = \alpha B_{ij}$

$$C = \sin A$$

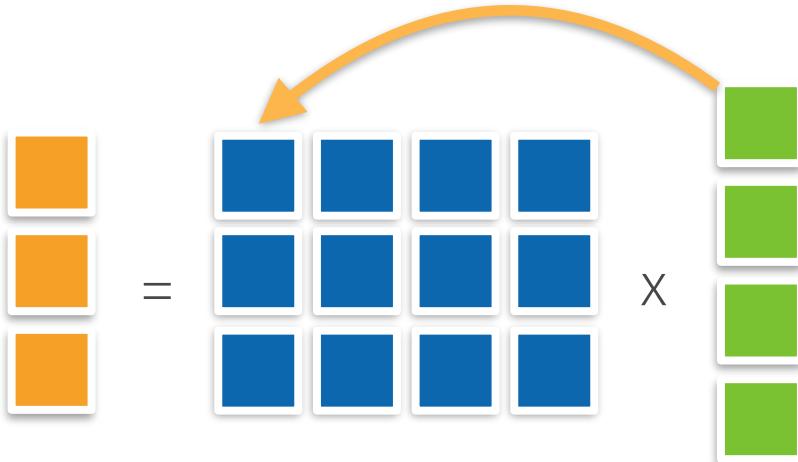
where $C_{ij} = \sin A_{ij}$



矩阵

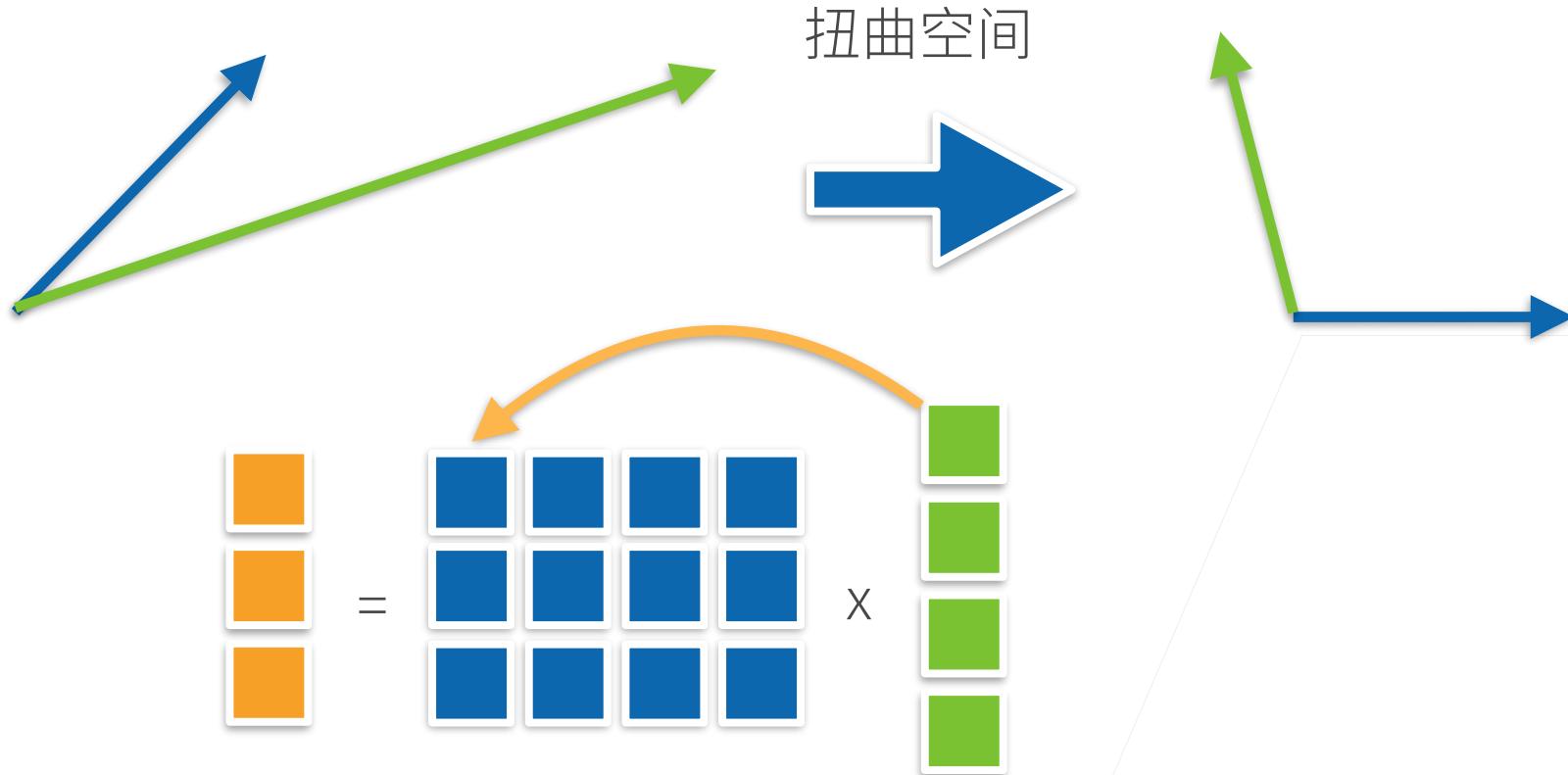
- 乘法 (矩阵乘以向量)

$$c = Ab \text{ where } c_i = \sum_j A_{ij} b_j$$





矩阵

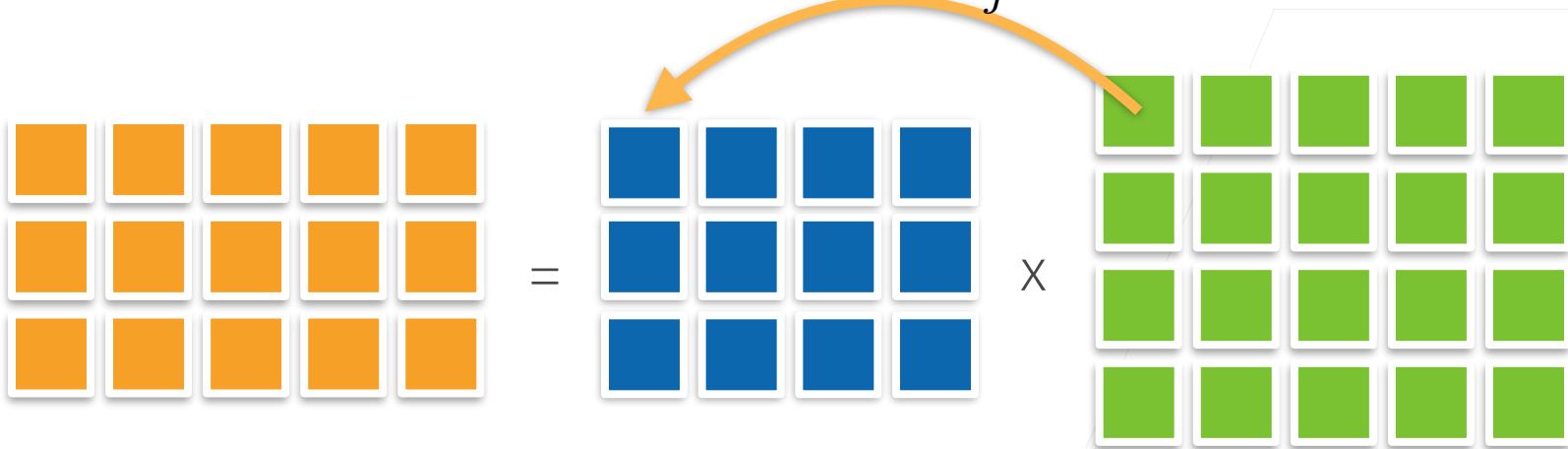




矩阵

- 乘法 (矩阵乘以矩阵)

$$C = AB \text{ where } C_{ik} = \sum_j A_{ij} B_{jk}$$





矩阵

- 范数

$$c = A \cdot b \text{ hence } \|c\| \leq \|A\| \cdot \|b\|$$

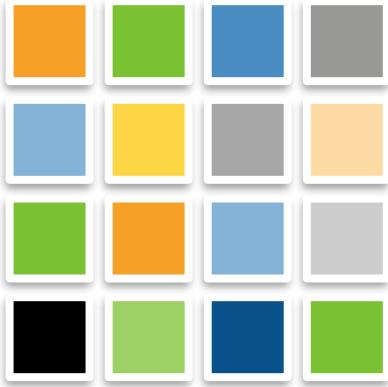
- 取决于如何衡量 b 和 c 的长度
- 常见范数
 - 矩阵范数：最小的满足的上面公式的值
 - Frobenius 范数

$$\|A\|_{\text{Frob}} = \left[\sum_{ij} A_{ij}^2 \right]^{\frac{1}{2}}$$

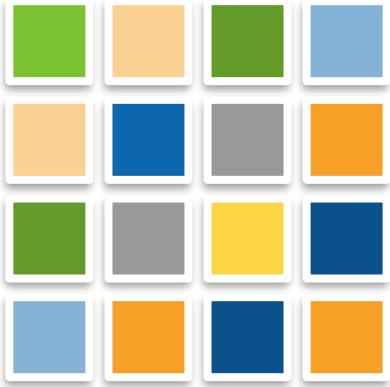


特殊矩阵

- 对称和反对称



$$A_{ij} = A_{ji} \text{ and } A_{ij} = -A_{ji}$$



- 正定

$$\|x\|^2 = x^\top x \geq 0 \text{ generalizes to } x^\top Ax \geq 0$$



特殊矩阵

- 正交矩阵
 - 所以行都相互正交
 - 所有行都有单位长度 U with $\sum_j U_{ij} U_{kj} = \delta_{ik}$
 - 可以写成 $UU^\top = \mathbf{1}$
- 置换矩阵

P where $P_{ij} = 1$ if and only if $j = \pi(i)$

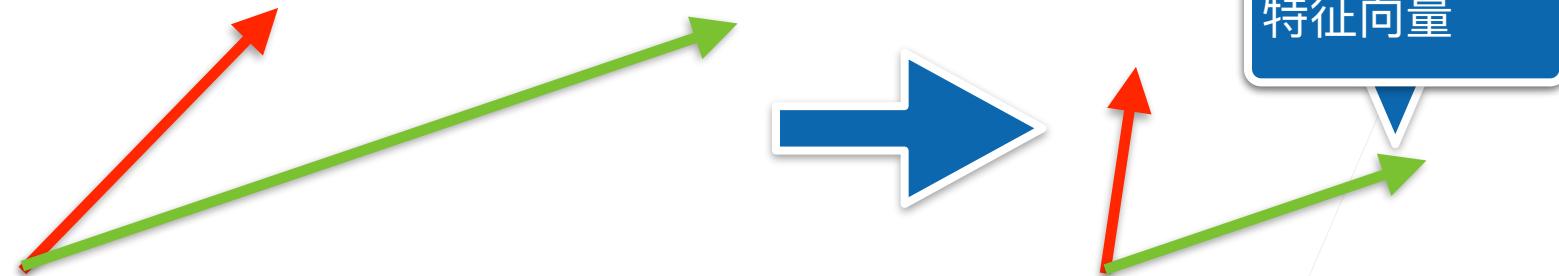
 - 置换矩阵是正交矩阵



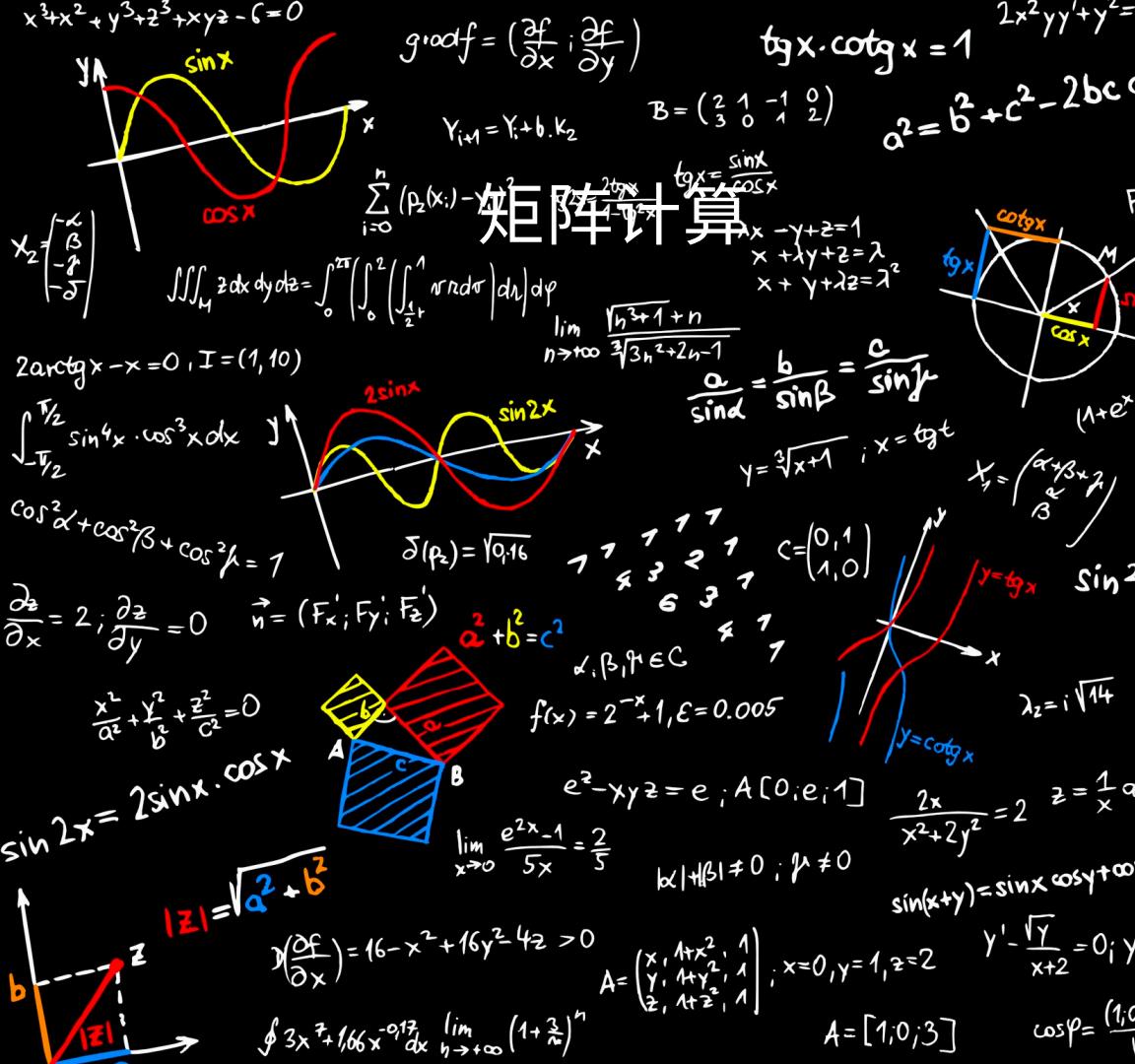
矩阵

- 特征向量和特征值
 - 不被矩阵改变方向的向量

$$Ax = \lambda x$$



- 对称矩阵总是可以找到特征向量



动手学深度学习 v2
李沐 · AWS





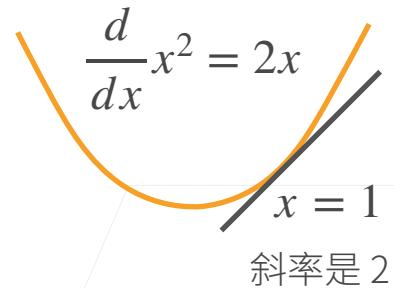
标量导数

$$(e^x)' = e^x.$$

y	a	x^n	$\exp(x)$	$\log(x)$	$\sin(x)$
$\frac{dy}{dx}$	0	nx^{n-1}	$\exp(x)$	$\frac{1}{x}$	$\cos(x)$

a 不是 x 的函数

导数是切线的斜率

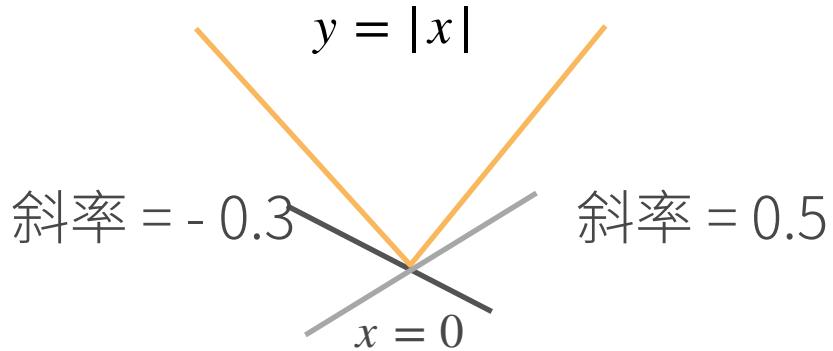


y	$u + v$	uv	$y = f(u), u = g(x)$
$\frac{dy}{dx}$	$\frac{du}{dx} + \frac{dv}{dx}$	$\frac{du}{dx}v + \frac{dv}{dx}u$	$\frac{dy}{du} \frac{du}{dx}$



亚导数

- 将导数拓展到不可微的函数



另一个例子

$$\frac{\partial}{\partial x} \max(x, 0) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ a & \text{if } x = 0, \quad a \in [0, 1] \end{cases}$$

$$\frac{\partial |x|}{\partial x} = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ a & \text{if } x = 0, \quad a \in [-1, 1] \end{cases}$$



梯度

- 将导数拓展到向量

向量		
标量		
	x	\mathbf{x}
标量	y	$\frac{\partial y}{\partial x}$
向量	\mathbf{y}	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$



$\partial y / \partial \mathbf{x}$

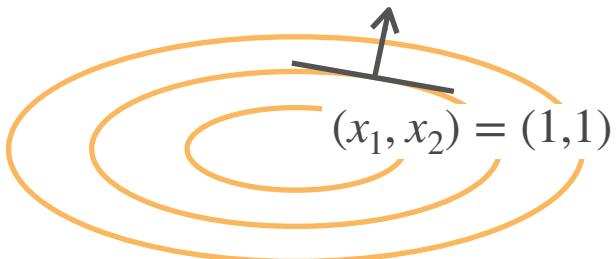
这里涉及到对向量求导，可以搜一下
<https://zhuanlan.zhihu.com/p/36448789>

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \frac{\partial y}{\partial \mathbf{x}} = \left[\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \dots, \frac{\partial y}{\partial x_n} \right]$$

y	$\frac{\partial y}{\partial x}$	$\frac{\partial y}{\partial \mathbf{x}}$
\mathbf{y}	$\frac{\partial \mathbf{y}}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$

$$\frac{\partial}{\partial \mathbf{x}} x_1^2 + 2x_2^2 = [2x_1, 4x_2]$$

方向 $(2, 4)$ 跟等高线正交





样例

y	a	au	$\text{sum}(\mathbf{x})$	$\ \mathbf{x}\ ^2$
$\frac{\partial y}{\partial \mathbf{x}}$	$\mathbf{0}^T$	$a \frac{\partial u}{\partial \mathbf{x}}$	$\mathbf{1}^T$	$2\mathbf{x}^T$

a is not a function of \mathbf{x}

$\mathbf{0}$ and $\mathbf{1}$ are vectors

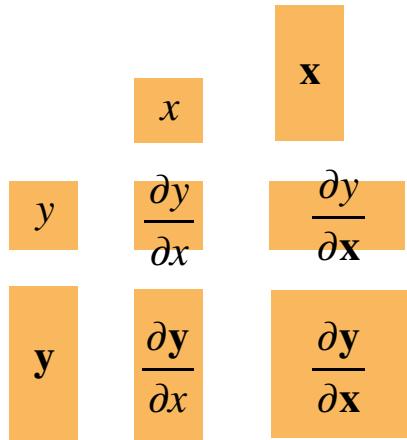
y	$u + v$	uv	$\langle \mathbf{u}, \mathbf{v} \rangle$
$\frac{\partial y}{\partial \mathbf{x}}$	$\frac{\partial u}{\partial \mathbf{x}} + \frac{\partial v}{\partial \mathbf{x}}$	$\frac{\partial u}{\partial \mathbf{x}}v + \frac{\partial v}{\partial \mathbf{x}}u$	$\mathbf{u}^T \frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \mathbf{v}^T \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$



$\partial \mathbf{y} / \partial x$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \vdots \\ \frac{\partial y_m}{\partial x} \end{bmatrix}$$



$\partial y / \partial \mathbf{x}$ 是行向量, $\partial \mathbf{y} / \partial x$ 是列向量

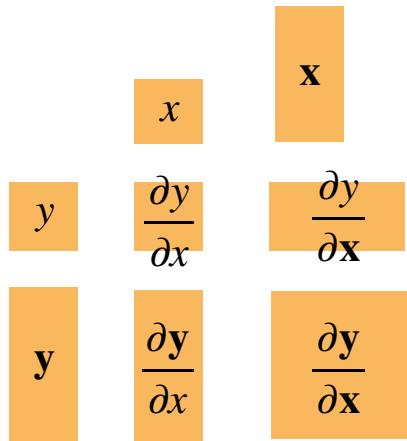
这个被称之为分子布局符号, 反过来的
版本叫分母布局符号



$\partial \mathbf{y} / \partial \mathbf{x}$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial \mathbf{x}} \\ \frac{\partial y_2}{\partial \mathbf{x}} \\ \vdots \\ \frac{\partial y_m}{\partial \mathbf{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1}, \frac{\partial y_1}{\partial x_2}, \dots, \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1}, \frac{\partial y_2}{\partial x_2}, \dots, \frac{\partial y_2}{\partial x_n} \\ \vdots \\ \frac{\partial y_m}{\partial x_1}, \frac{\partial y_m}{\partial x_2}, \dots, \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$





样例

\mathbf{y}	\mathbf{a}	\mathbf{x}	\mathbf{Ax}	$\mathbf{x}^T \mathbf{A}$
$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$\mathbf{0}$	\mathbf{I}	\mathbf{A}	\mathbf{A}^T

$$\mathbf{x} \in \mathbb{R}^n, \quad \mathbf{y} \in \mathbb{R}^m, \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathbb{R}^{m \times n}$$

a , \mathbf{a} and \mathbf{A} are not functions of \mathbf{x}

$\mathbf{0}$ and \mathbf{I} are matrices

\mathbf{y}	$a\mathbf{u}$	\mathbf{Au}	$\mathbf{u} + \mathbf{v}$
$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$a \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\mathbf{A} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$

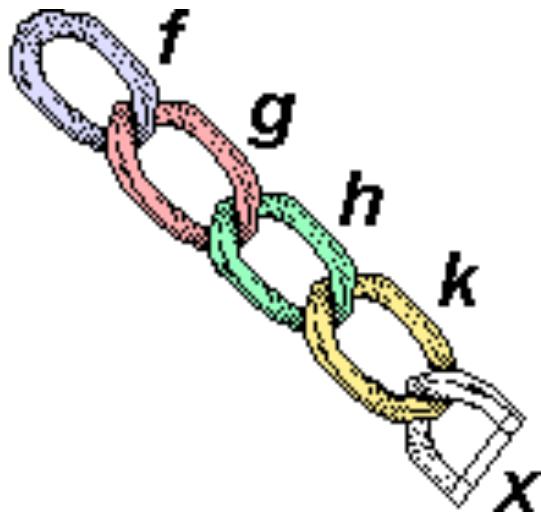


拓展到矩阵

	标量	向量	矩阵
标量	x (1,)	\mathbf{x} ($n, 1$)	\mathbf{X} (n, k)
向量	y (1,)	$\frac{\partial y}{\partial \mathbf{x}}$ (1,)	$\frac{\partial y}{\partial \mathbf{X}}$ (k, n)
矩阵	\mathbf{y} ($m, 1$)	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ ($m, 1$)	$\frac{\partial \mathbf{y}}{\partial \mathbf{X}}$ (m, k, n)
	\mathbf{Y} (m, l)	$\frac{\partial \mathbf{Y}}{\partial \mathbf{x}}$ (m, l)	$\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ (m, l, k, n)



链式法则和自动求导





向量链式法则

- 标量链式法则

$$y = f(u), \quad u = g(x) \quad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \frac{\partial u}{\partial x}$$

- 拓展到向量

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial u} \frac{\partial u}{\partial \mathbf{x}}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

(1,n) (1,) (1,n)

(1,n) (1,k) (k, n)

(m, n) (m, k) (k, n)

例子 1

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

假设

$$\mathbf{x}, \mathbf{w} \in \mathbb{R}^n, \quad y \in \mathbb{R}$$

$$z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$$

计算

$$\frac{\partial z}{\partial \mathbf{w}}$$

$$\begin{aligned}\frac{\partial z}{\partial \mathbf{w}} &= \frac{\partial z}{\partial b} \frac{\partial b}{\partial a} \frac{\partial a}{\partial \mathbf{w}} \\ &= \frac{\partial b^2}{\partial b} \frac{\partial a - y}{\partial a} \frac{\partial \langle \mathbf{x}, \mathbf{w} \rangle}{\partial \mathbf{w}} \\ &= 2b \cdot 1 \cdot \mathbf{x}^T \\ &= 2(\langle \mathbf{x}, \mathbf{w} \rangle - y) \mathbf{x}^T\end{aligned}$$



分解

$$b = a - y$$

$$z = b^2$$

例子 2

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

假设

$$\mathbf{X} \in \mathbb{R}^{m \times n}, \quad \mathbf{w} \in \mathbb{R}^n, \quad \mathbf{y} \in \mathbb{R}^m$$

$$z = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

计算

$$\frac{\partial z}{\partial \mathbf{w}}$$

$$\begin{aligned}\frac{\partial z}{\partial \mathbf{w}} &= \frac{\partial z}{\partial \mathbf{b}} \frac{\partial \mathbf{b}}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \\ &= \frac{\partial \|\mathbf{b}\|^2}{\partial \mathbf{b}} \frac{\partial \mathbf{a} - \mathbf{y}}{\partial \mathbf{a}} \frac{\partial \mathbf{X}\mathbf{w}}{\partial \mathbf{w}} \\ &= 2\mathbf{b}^T \times \mathbf{I} \times \mathbf{X} \\ &= 2(\mathbf{X}\mathbf{w} - \mathbf{y})^T \mathbf{X}\end{aligned}$$

分解

$$\mathbf{a} = \mathbf{X}\mathbf{w}$$



$$\mathbf{b} = \mathbf{a} - \mathbf{y}$$

$$z = \|\mathbf{b}\|^2$$



自动求导

- 自动求导计算一个函数在指定值上的导数
- 它有别于
 - 符号求导

```
In[1]:= D[4 x3 + x2 + 3, x]
```

```
Out[1]= 2 x + 12 x2
```

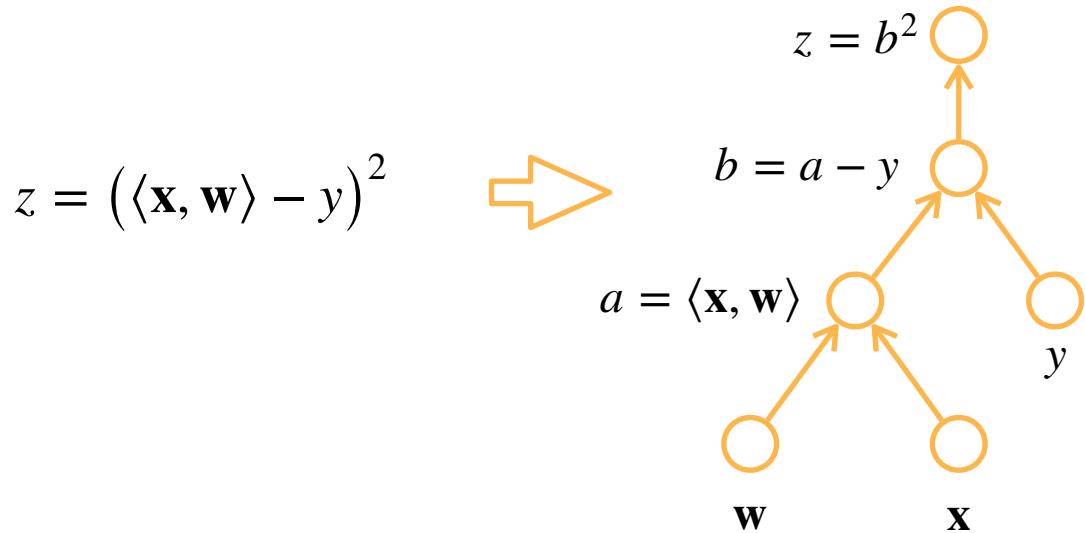
- 数值求导

$$\frac{\partial f(x)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$



计算图

- 将代码分解成操作子
- 将计算表示成一个无环图





计算图

- 将代码分解成操作子
- 将计算表示成一个无环图
- 显示构造

```
from mxnet import sym

a = sym.var()
b = sym.var()
c = 2 * a + b
# bind data into a and b later
```



计算图

- 将代码分解成操作子
- 将计算表示成一个无环图
- 显式构造
 - Tensorflow/Theano/MXNet
- 隐式构造
 - PyTorch/MXNet

```
from mxnet import autograd, nd  
  
with autograd.record():  
    a = nd.ones((2,1))  
    b = nd.ones((2,1))  
    c = 2 * a + b
```



自动求导的两种模式

- 链式法则： $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_n} \frac{\partial u_n}{\partial u_{n-1}} \dots \frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x}$
- 正向累积 $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_n} \left(\frac{\partial u_n}{\partial u_{n-1}} \left(\dots \left(\frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x} \right) \right) \right)$
- 反向累积、又称反向传递

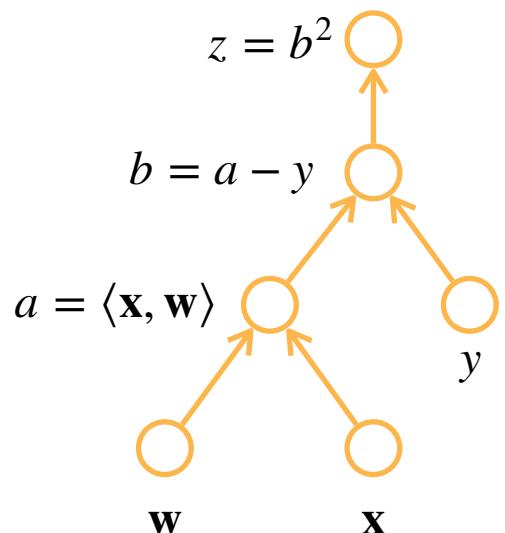
$$\frac{\partial y}{\partial x} = \left(\left(\left(\frac{\partial y}{\partial u_n} \frac{\partial u_n}{\partial u_{n-1}} \right) \dots \right) \frac{\partial u_2}{\partial u_1} \right) \frac{\partial u_1}{\partial x}$$



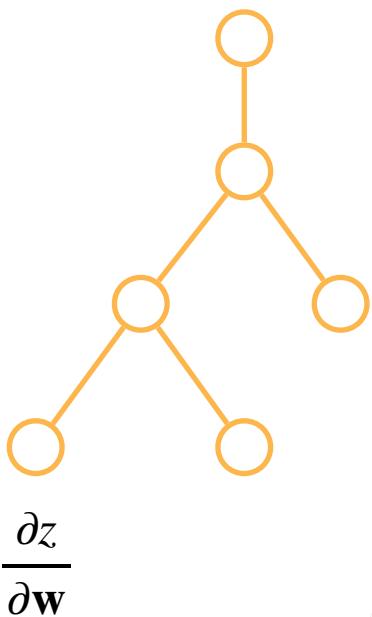
反向累积

$$z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$$

正向



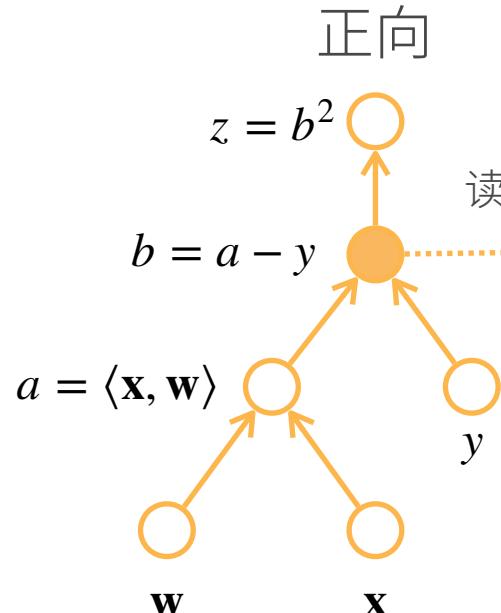
反向



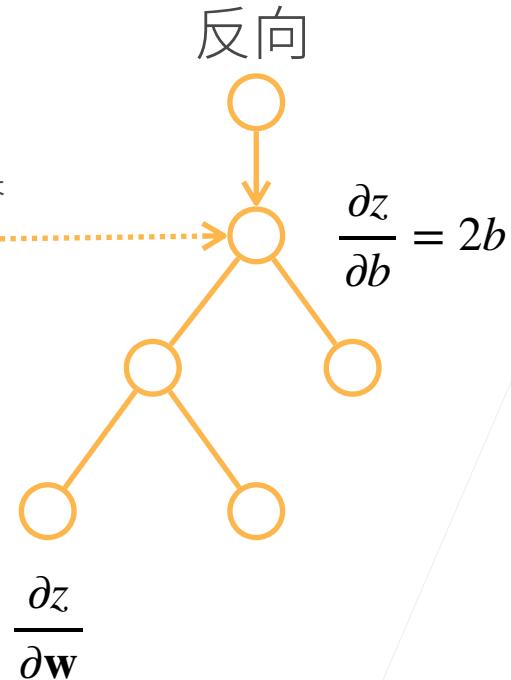


反向累积

$$z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$$



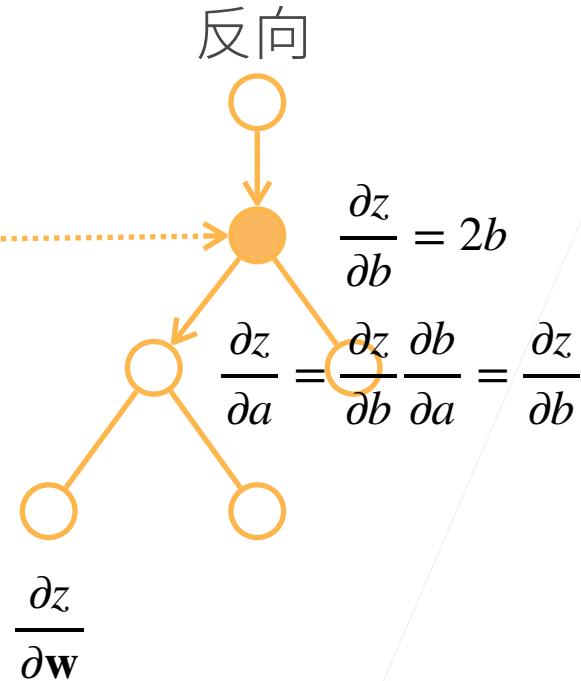
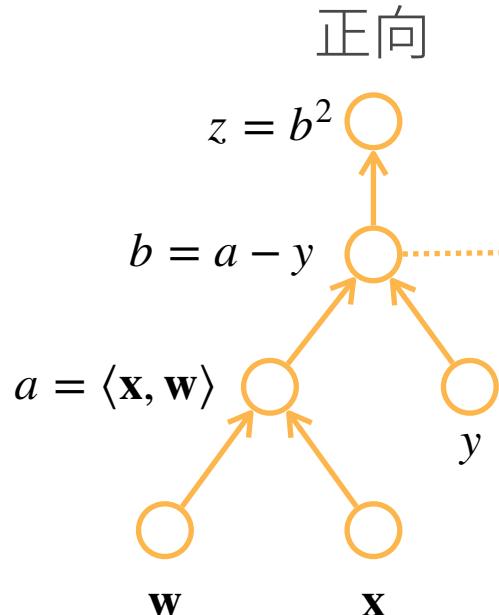
读取之前结果





反向累积

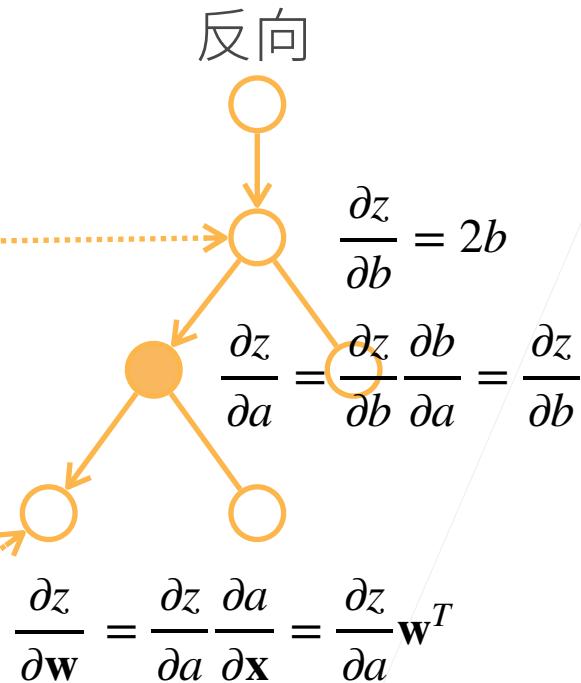
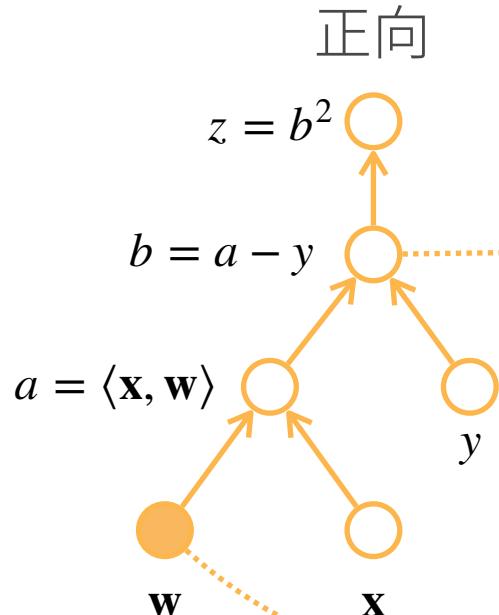
$$z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$$





反向累积

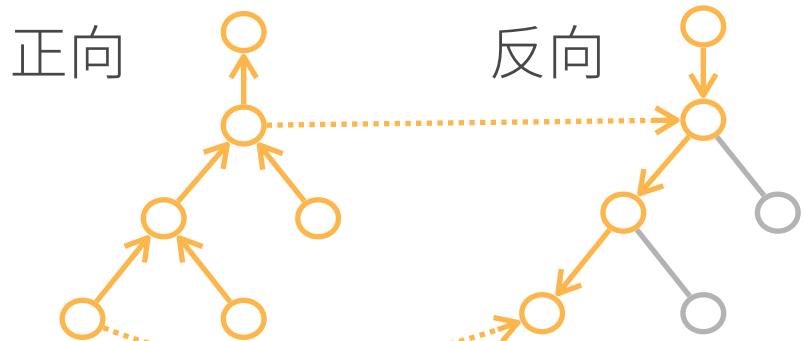
$$z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$$





反向累积总结

- 构造计算图
- 前向：执行图，存储中间结果
- 反向：从相反方向执行图
 - 去除不需要的枝



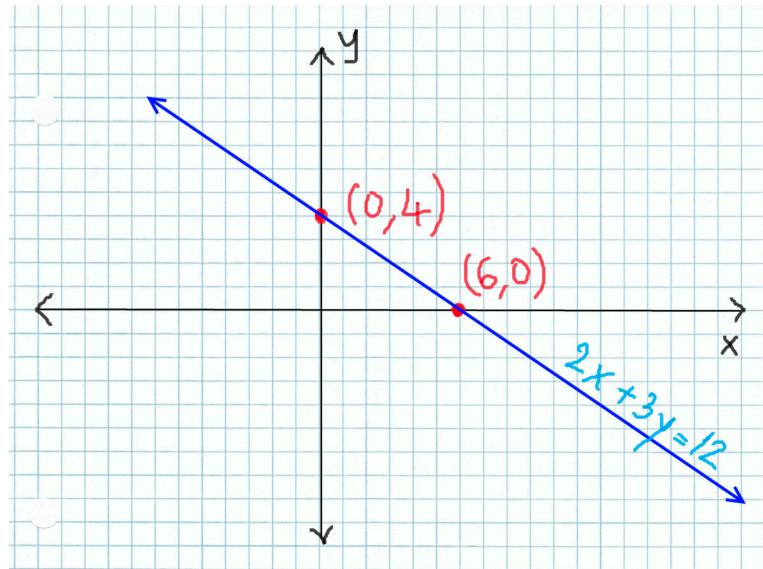


复杂度

- 计算复杂度: $O(n)$, n 是操作子个数
 - 通常正向和反向的代价类似
- 内存复杂度: $O(n)$, 因为需要存储正向的所有中间结果
- 跟正向累积对比:
 - $O(n)$ 计算复杂度用来计算一个变量的梯度
 - $O(1)$ 内存复杂度



线性回归





如何在美国买房

- 看中一个房，参观了解
- 估计一个价格，出价



标价

\$5,498,000 | 7 | 5 | 4,865 Sq. Ft.
Price | Beds | Baths | \$1130 / Sq. Ft.

Redfin Estimate: \$5,390,037 On Redfin: 15 days

预计价格

Virtual Tour

- [Branded Virtual Tour](#)
- [Virtual Tour \(External Link\)](#)

Parking Information

- Garage (Minimum): 2
- Garage (Maximum): 2
- Parking Description: Attached Garage, On Street
- Garage Spaces: 2

Interior Features

Bedroom Information

- # of Bedrooms (Minimum): 7
- # of Bedrooms (Maximum): 7

Multi-Unit Information

- # of Stories: 2

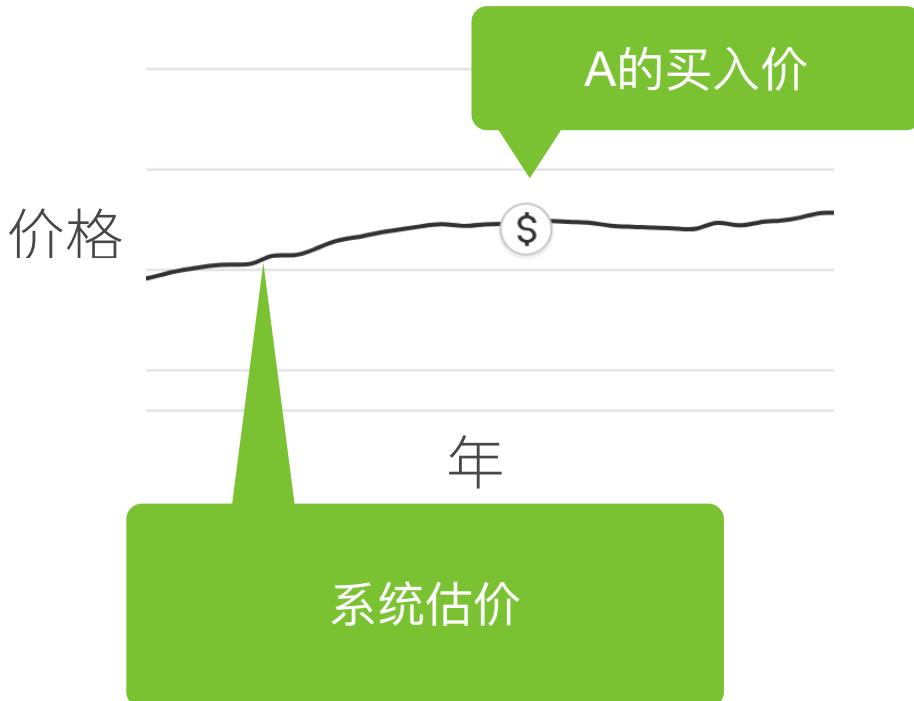
School Information

- Elementary School: El Carmelo
- Elementary School District: Palo Alto Unified
- Middle School: Jane Lathrop Sta
- High School: Palo Alto High
- High School District: Palo Alto Unified



房价预测

很重要，这是真钱



\$100K+ 差价





一个简化模型



- 假设 1：影响房价的关键因素是卧室个数，卫生间个数，和居住面积，记为 x_1, x_2, x_3
- 假设 2：成交价是关键因素的加权和

$$y = w_1x_1 + w_2x_2 + w_3x_3 + b$$

权重和偏差的实际值在后面决定



线性模型

- 给定 n 维输入 $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$
- 线性模型有一个 n 维权重和一个标量偏差

$$\mathbf{w} = [w_1, w_2, \dots, w_n]^T, \quad b$$

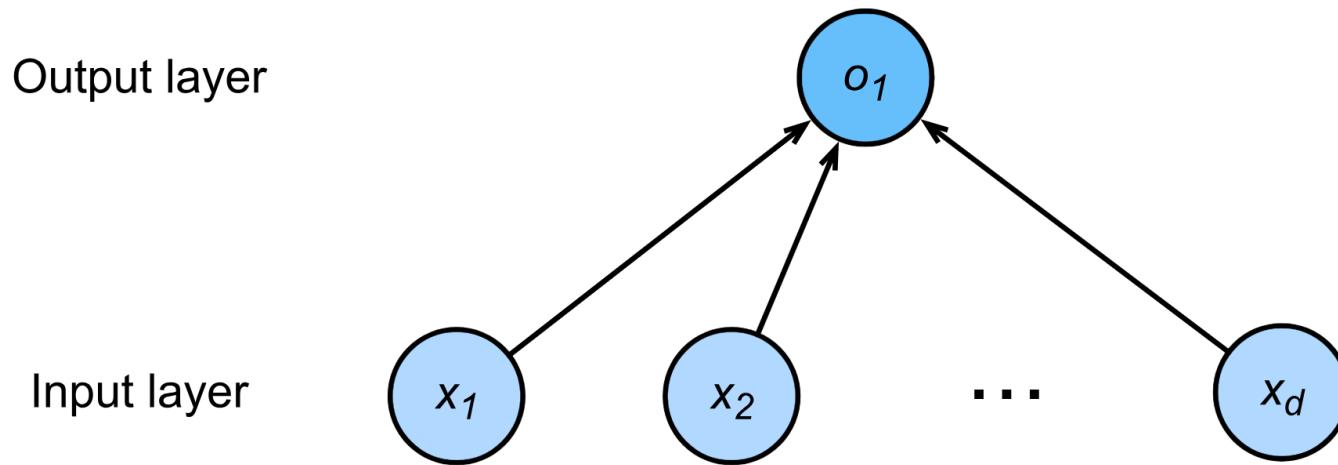
- 输出是输入的加权和

$$y = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

向量版本: $y = \langle \mathbf{w}, \mathbf{x} \rangle + b$



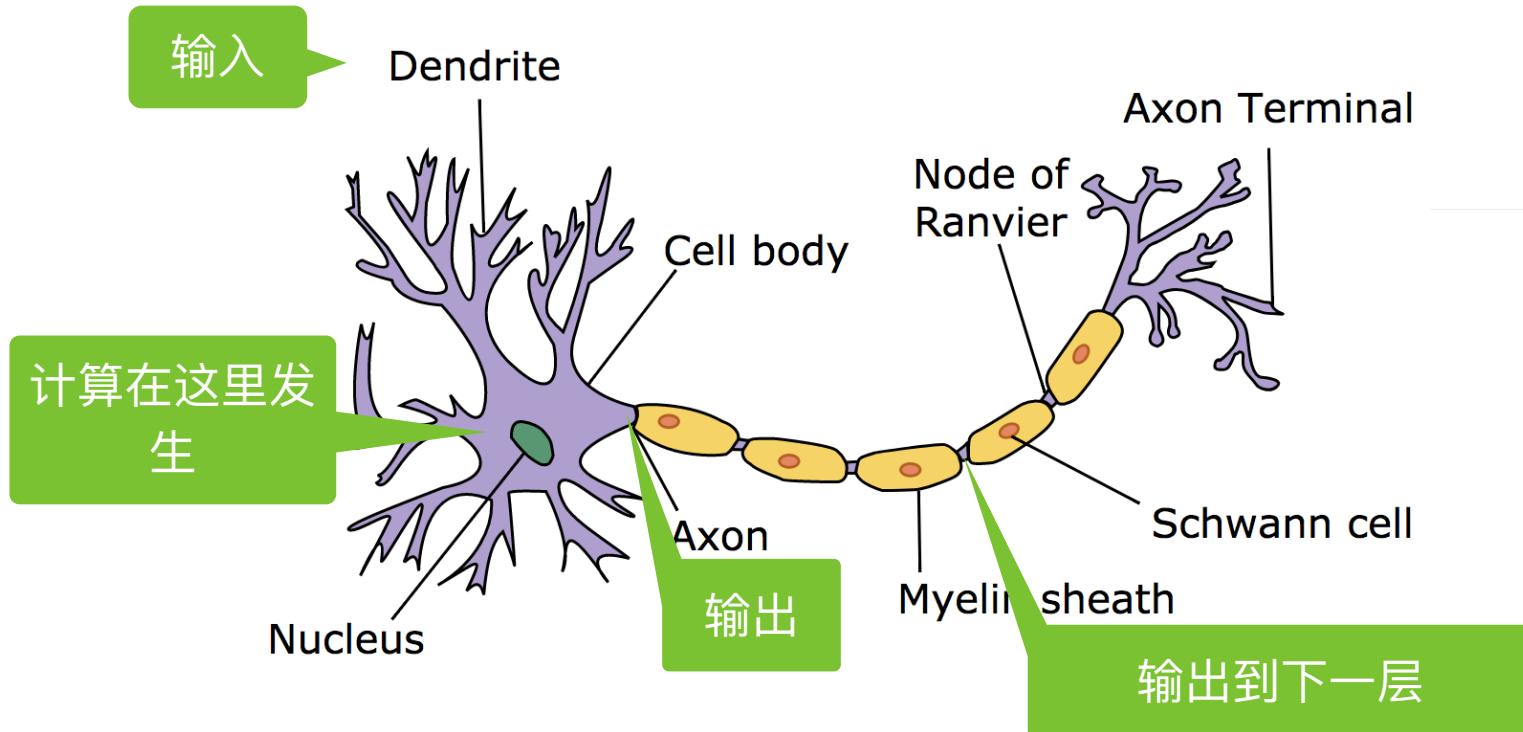
线性模型可以看做是单层神经网络





神经网络源于神经科学

真实的神经元





衡量预估质量

- 比较真实值和预估值，例如房屋售价和估价
- 假设 y 是真实值， \hat{y} 是估计值，我们可以比较

$$\ell(y, \hat{y}) = \frac{1}{2} (y - \hat{y})^2$$

这个叫做平方损失



训练数据

- 收集一些数据点来决定参数值（权重和偏差），例如过去6个月卖的房子
- 这被称之为训练数据
- 通常越多越好
- 假设我们有 n 个样本，记

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^T \quad \mathbf{y} = [y_1, y_2, \dots, y_n]^T$$



参数学习

- 训练损失

$$\ell(\mathbf{X}, \mathbf{y}, \mathbf{w}, b) = \frac{1}{2n} \sum_{i=1}^n (y_i - \langle \mathbf{x}_i, \mathbf{w} \rangle - b)^2 = \frac{1}{2n} \| \mathbf{y} - \mathbf{X}\mathbf{w} - b \| ^2$$

- 最小化损失来学习参数

$$\mathbf{w}^*, \mathbf{b}^* = \arg \min_{\mathbf{w}, b} \ell(\mathbf{X}, \mathbf{y}, \mathbf{w}, b)$$



显示解

- 将偏差加入权重 $\mathbf{X} \leftarrow [\mathbf{X}, \mathbf{1}] \quad \mathbf{w} \leftarrow \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}$

$$\ell(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{1}{2n} \parallel \mathbf{y} - \mathbf{X}\mathbf{w} \parallel^2 \quad \frac{\partial}{\partial \mathbf{w}} \ell(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{1}{n} (\mathbf{y} - \mathbf{X}\mathbf{w})^T \mathbf{X}$$

- 损失是凸函数，所以最优解满足

$$\frac{\partial}{\partial \mathbf{w}} \ell(\mathbf{X}, \mathbf{y}, \mathbf{w}) = 0$$

$$\Leftrightarrow \frac{1}{n} (\mathbf{y} - \mathbf{X}\mathbf{w})^T \mathbf{X} = 0$$

$$\Leftrightarrow \mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

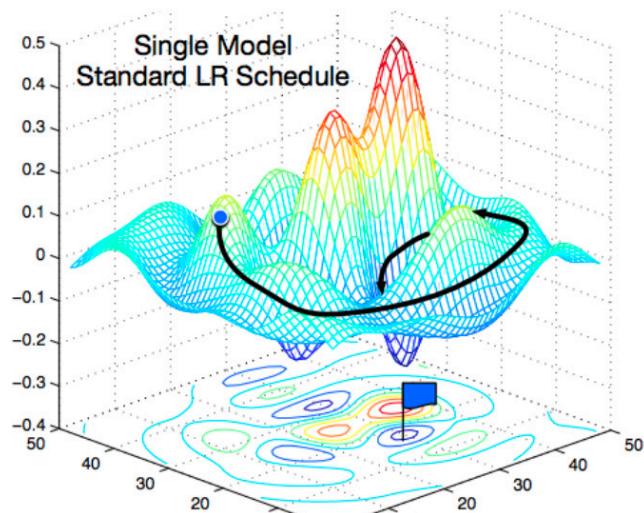


总结

- 线性回归是对n维输入的加权，外加偏差
- 使用平方损失来衡量预测值和真实值的差异
- 线性回归有显示解
- 线性回归可以看做是单层神经网络



基础优化方法



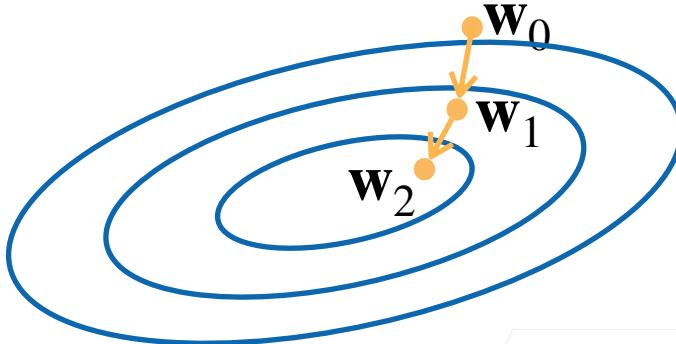


梯度下降

- 挑选一个初始值 \mathbf{w}_0
- 重复迭代参数 $t=1,2,3$

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \frac{\partial \ell}{\partial \mathbf{w}_{t-1}}$$

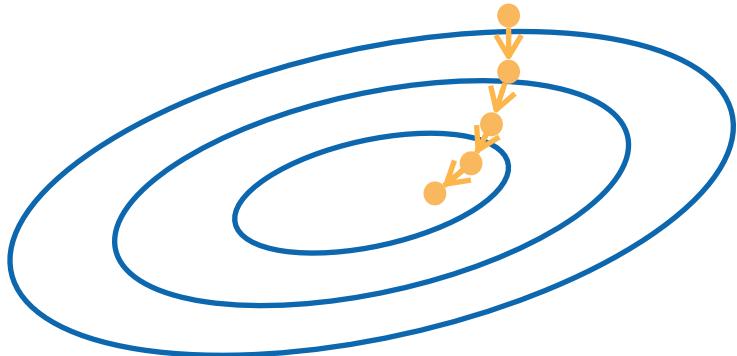
- 沿梯度方向将增加损失函数值
- 学习率：步长的超参数



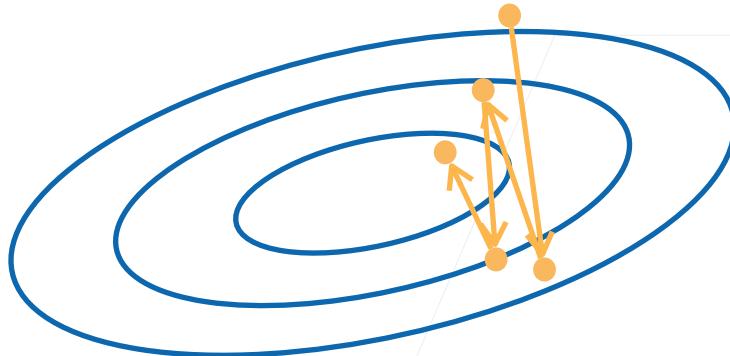


选择学习率

不能太小



也不能太大





小批量随机梯度下降

- 在整个训练集上算梯度太贵
 - 一个深度神经网络模型可能需要数分钟至数小时
- 我们可以随机采样 b 个样本 i_1, i_2, \dots, i_b 来近似损失

$$\frac{1}{b} \sum_{i \in I_b} \ell(\mathbf{x}_i, y_i, \mathbf{w})$$

- b 是批量大小，另一个重要的超参数



选择批量大小

不能太小

每次计算量太小，不适合并行来最大利用计算资源

不能太大

内存消耗增加
浪费计算，例如如果所有样本都是相同的

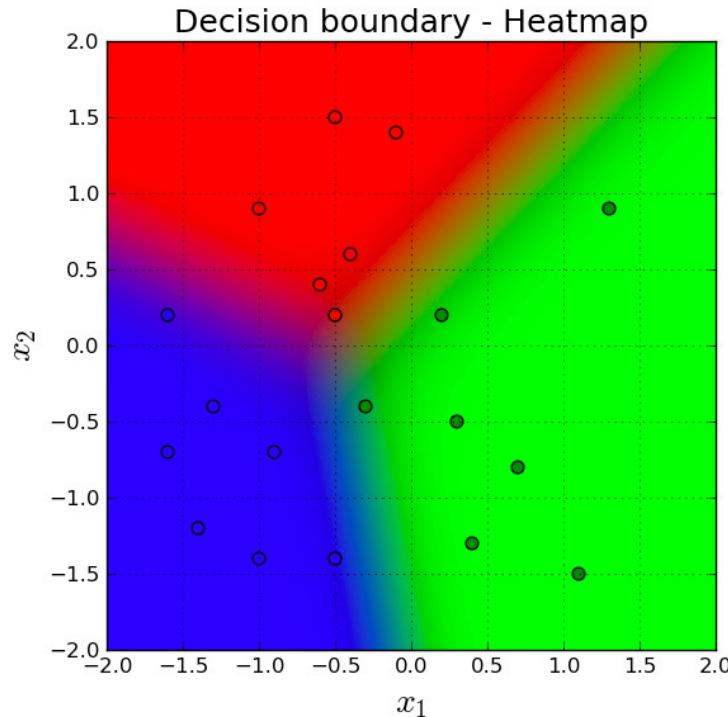


总结

- 梯度下降通过不断沿着反梯度方向更新参数求解
- 小批量随机梯度下降是深度学习默认的求解算法
- 两个重要的超参数是批量大小和学习率



Softmax 回归





回归 vs 分类

- 回归估计一个连续值
- 分类预测一个离散类别

MNIST: 手写数字识别 (10类)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	8	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9

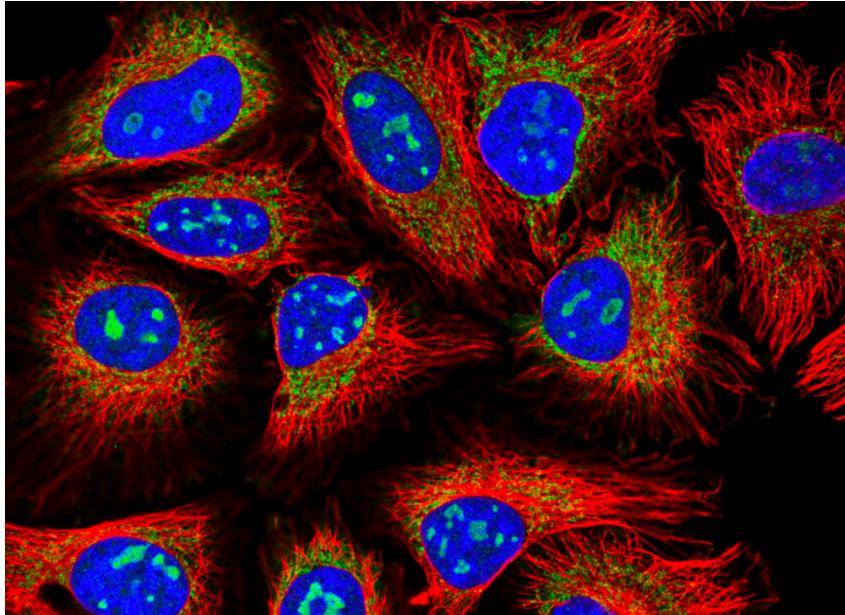
ImageNet: 自然物体分类 (1000类)





Kaggle 上的分类问题

将人类蛋白质显微镜图片分成28类



- 0. Nucleoplasm
- 1. Nuclear membrane
- 2. Nucleoli
- 3. Nucleoli fibrillar
- 4. Nuclear speckles
- 5. Nuclear bodies
- 6. Endoplasmic reticu
- 7. Golgi apparatus
- 8. Peroxisomes
- 9. Endosomes
- 10. Lysosomes
- 11. Intermediate fila
- 12. Actin filaments
- 13. Focal adhesion si
- 14. Microtubules
- 15. Microtubule ends
- 16. Cytokinetic bridg

<https://www.kaggle.com/c/human-protein-atlas-image-classification>



Kaggle 上的分类问题

将恶意软件分成9个类别



<https://www.kaggle.com/c/malware-classification>

Kaggle 上的分类问题



将恶意的 Wikipedia 评论分成 7 类

comment_text	toxic	severe_toxic	obscene
Explanation\nWhy the edits made under my user...	0	0	0
D'aww! He matches this background colour I'm s...	0	0	0
Hey man, I'm really not trying to edit war. It...	0	0	0
"\nMore\nI can't make any real suggestions on ...	0	0	0
You, sir, are my hero. Any chance you remember...	0	0	0

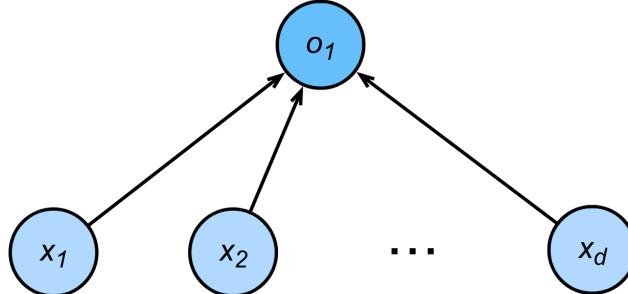
<https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge>



从回归到多类分类

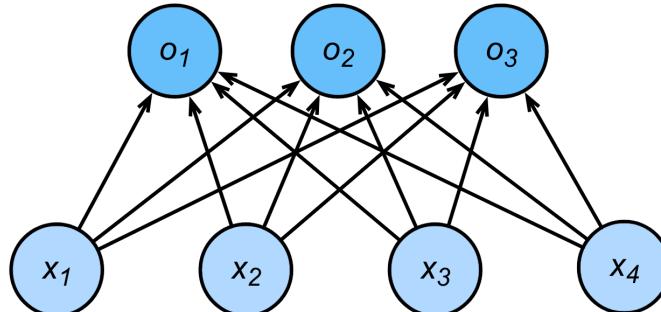
回归

- 单连续数值输出
- 自然区间 \mathbb{R}
- 跟真实值的区别作为损失



分类

- 通常多个输出
- 输出 i 是预测为第 i 类的置信度





从回归到多类分类 – 均方损失

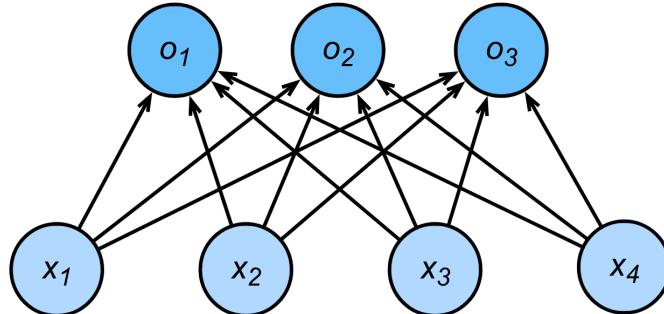
- 对类别进行一位有效编码

$$\mathbf{y} = [y_1, y_2, \dots, y_n]^\top$$

$$y_i = \begin{cases} 1 & \text{if } i = y \\ 0 & \text{otherwise} \end{cases}$$

- 使用均方损失训练
- 最大值最为预测

$$\hat{y} = \operatorname{argmax}_i o_i$$





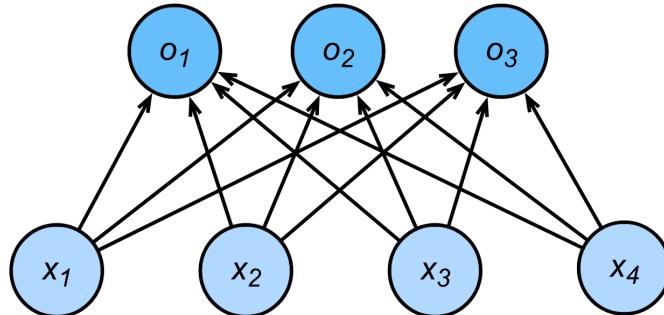
从回归到多类分类 – 无校验比例

- 对类别进行一位有效编码
- 最大值最为预测

$$\hat{y} = \operatorname{argmax}_i o_i$$

- 需要更置信的识别正确类 (大余量)

$$o_y - o_i \geq \Delta(y, i)$$





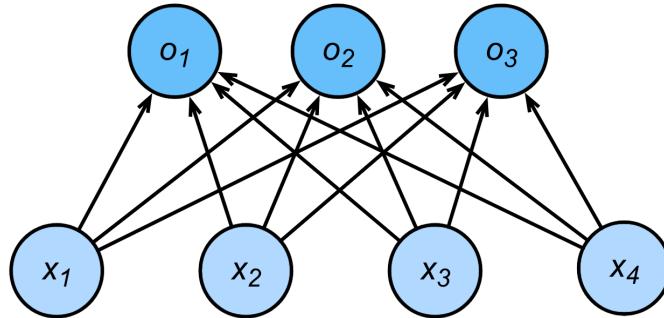
从回归到多类分类 – 校验比例

- 输出匹配概率（非负，和为 1）

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{o})$$

$$\hat{y}_i = \frac{\exp(o_i)}{\sum_k \exp(o_k)}$$

- 概率 \mathbf{y} 和 $\hat{\mathbf{y}}$ 的区别作为损失





Softmax 和交叉熵损失

- 交叉熵常用来衡量两个概率的区别 $H(\mathbf{p}, \mathbf{q}) = \sum_i -p_i \log(q_i)$
- 将它作为损失

$$l(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_i y_i \log \hat{y}_i = - \log \hat{y}_y$$

- 其梯度是真实概率和预测概率的区别

$$\partial_{o_i} l(\mathbf{y}, \hat{\mathbf{y}}) = \text{softmax}(\mathbf{o})_i - y_i$$

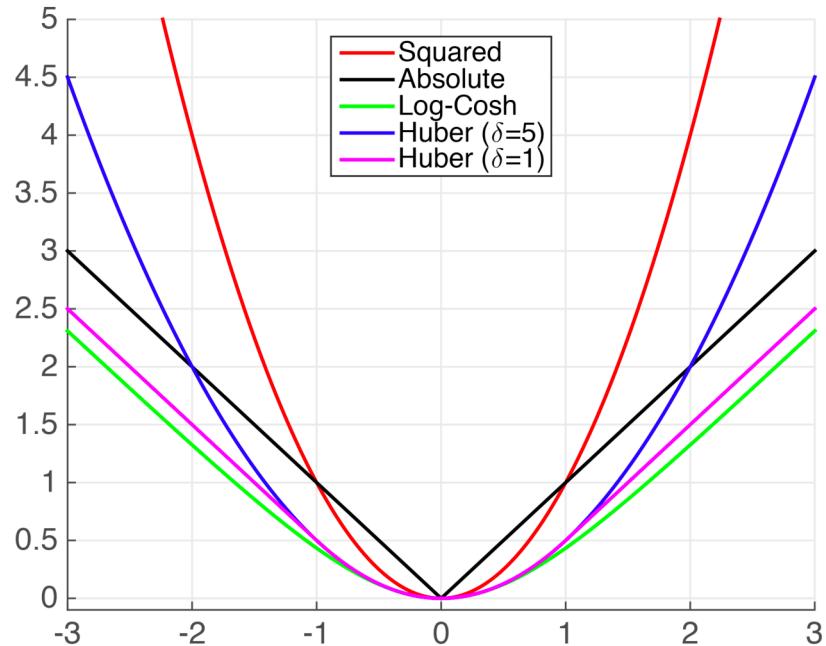


总结

- Softmax 回归是一个多类分类模型
- 使用 Softmax 操作子得到每个类的预测置信度
- 使用交叉熵来衡量预测和标号的区别

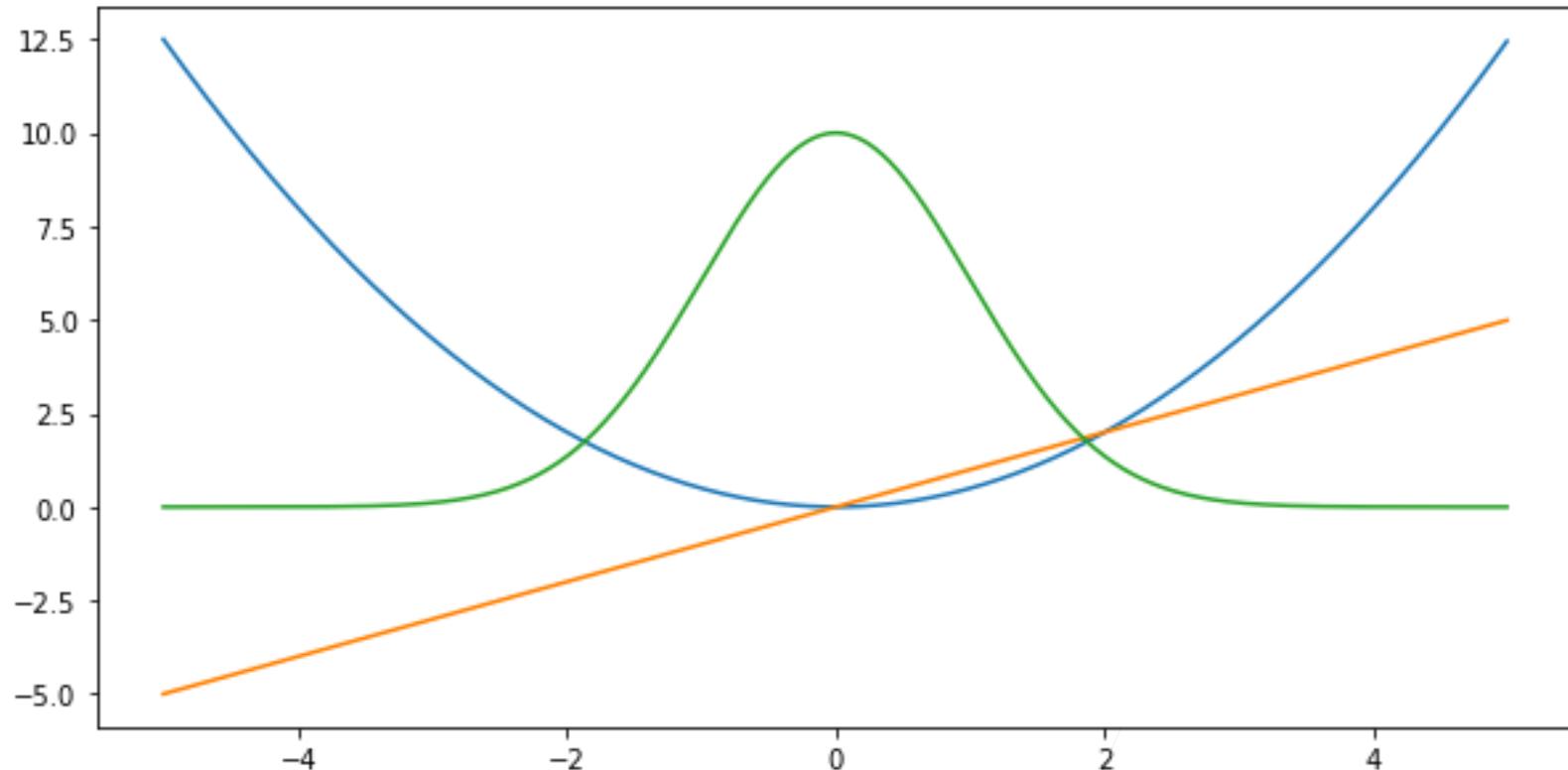


损失函数



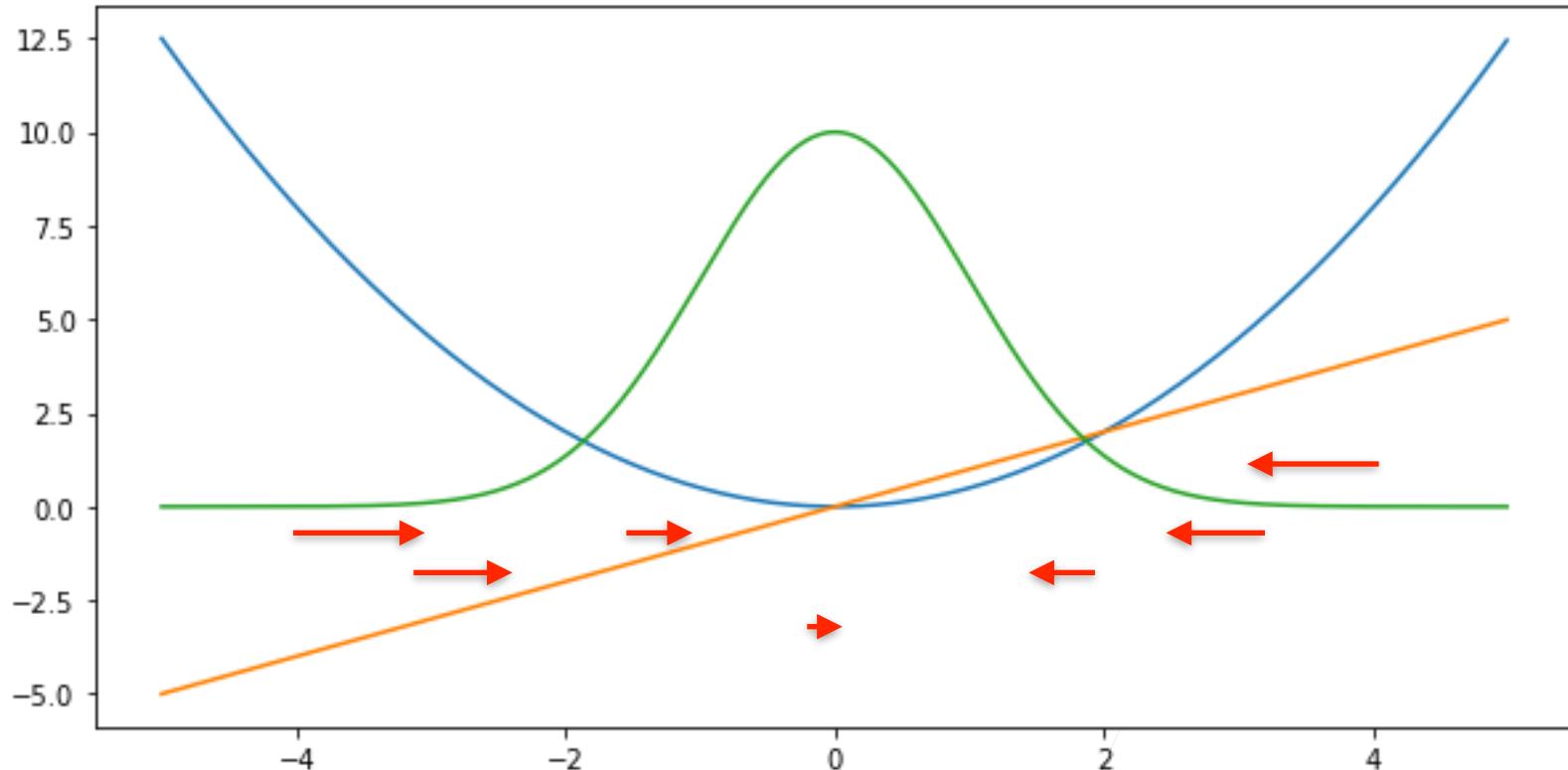
L2 Loss

$$l(y, y') = \frac{1}{2}(y - y')^2$$



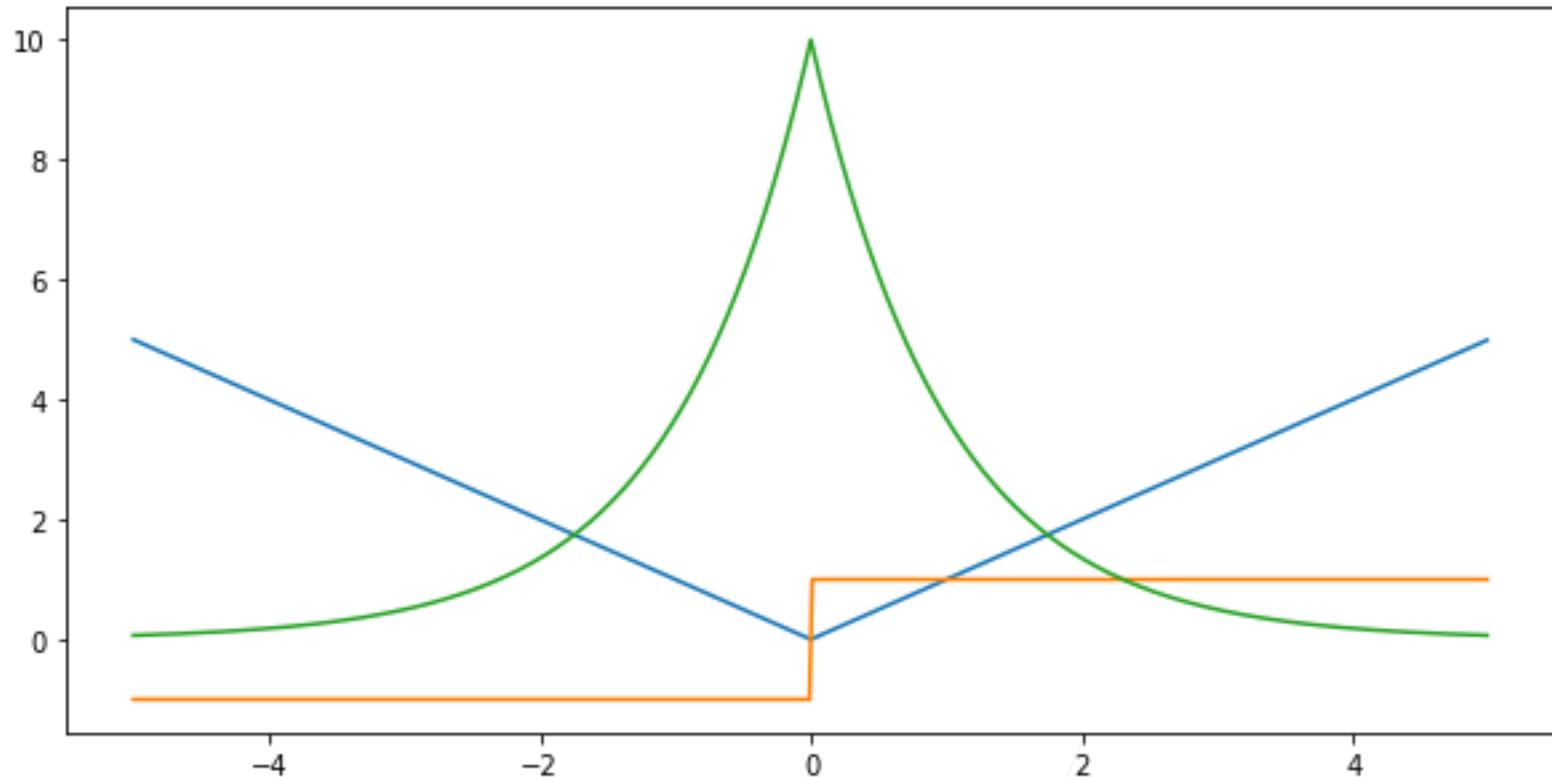
L2 Loss

$$l(y, y') = \frac{1}{2}(y - y')^2$$



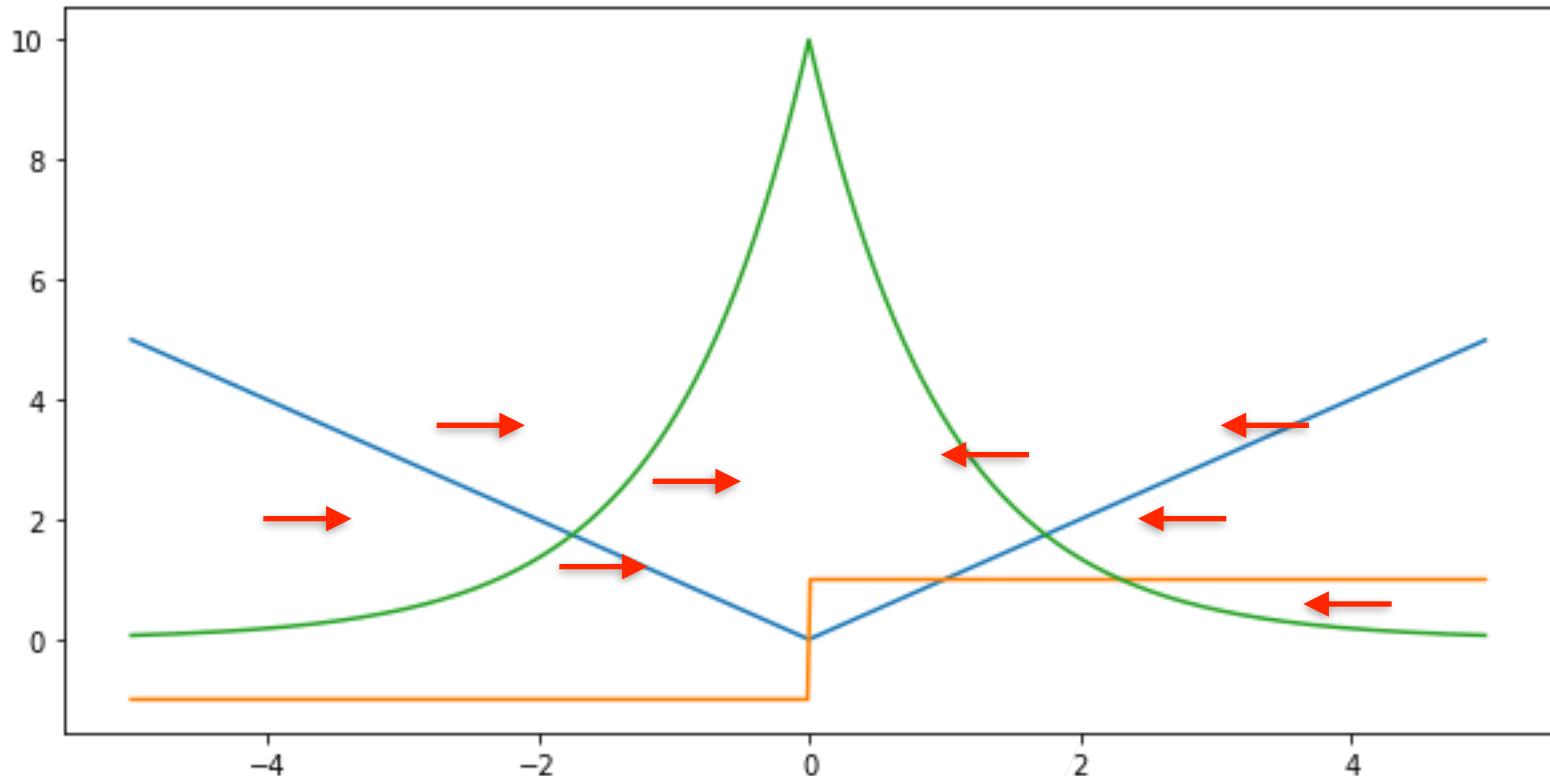
L1 Loss

$$l(y, y') = |y - y'|$$



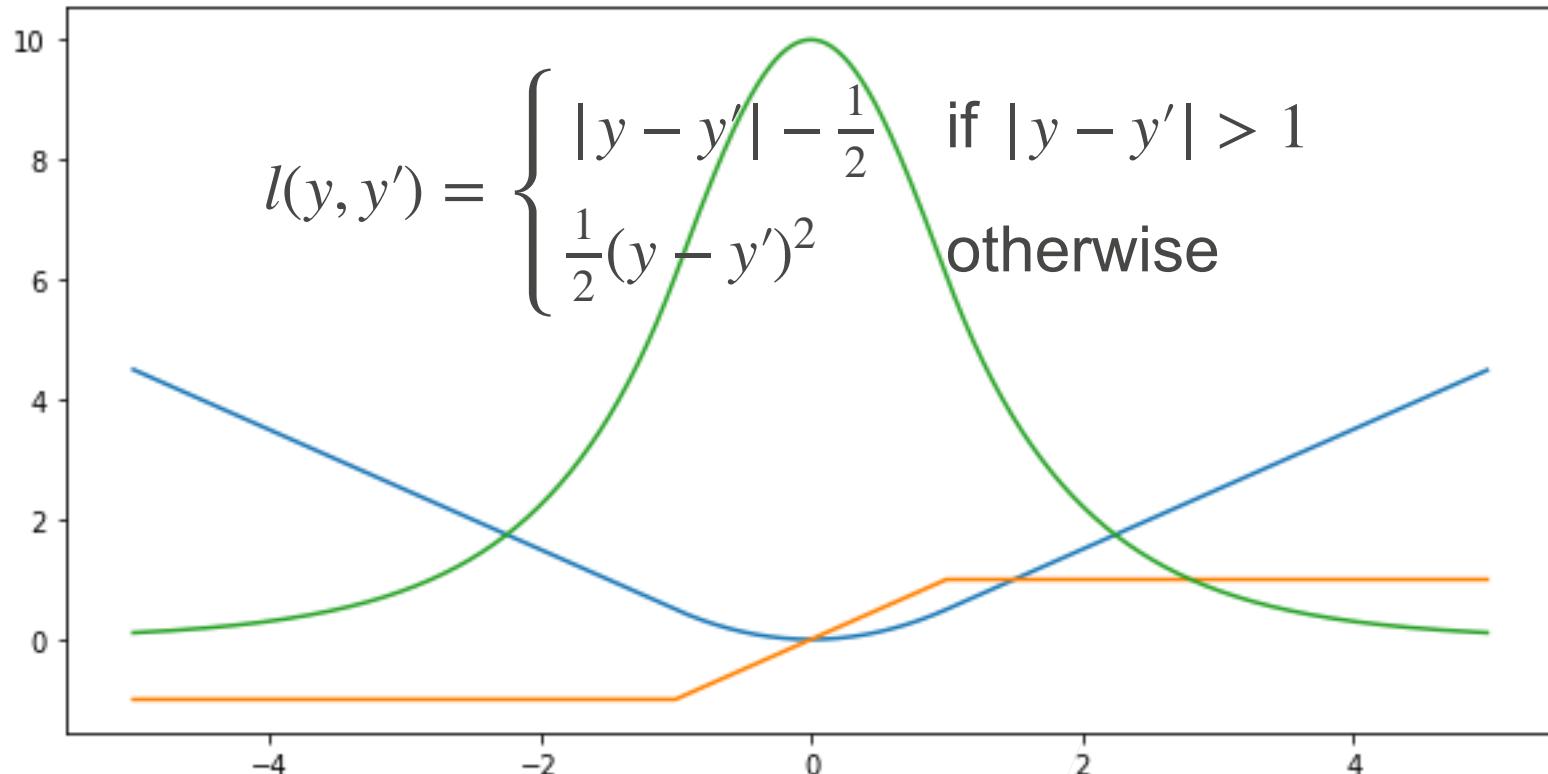
L1 Loss

$$l(y, y') = |y - y'|$$



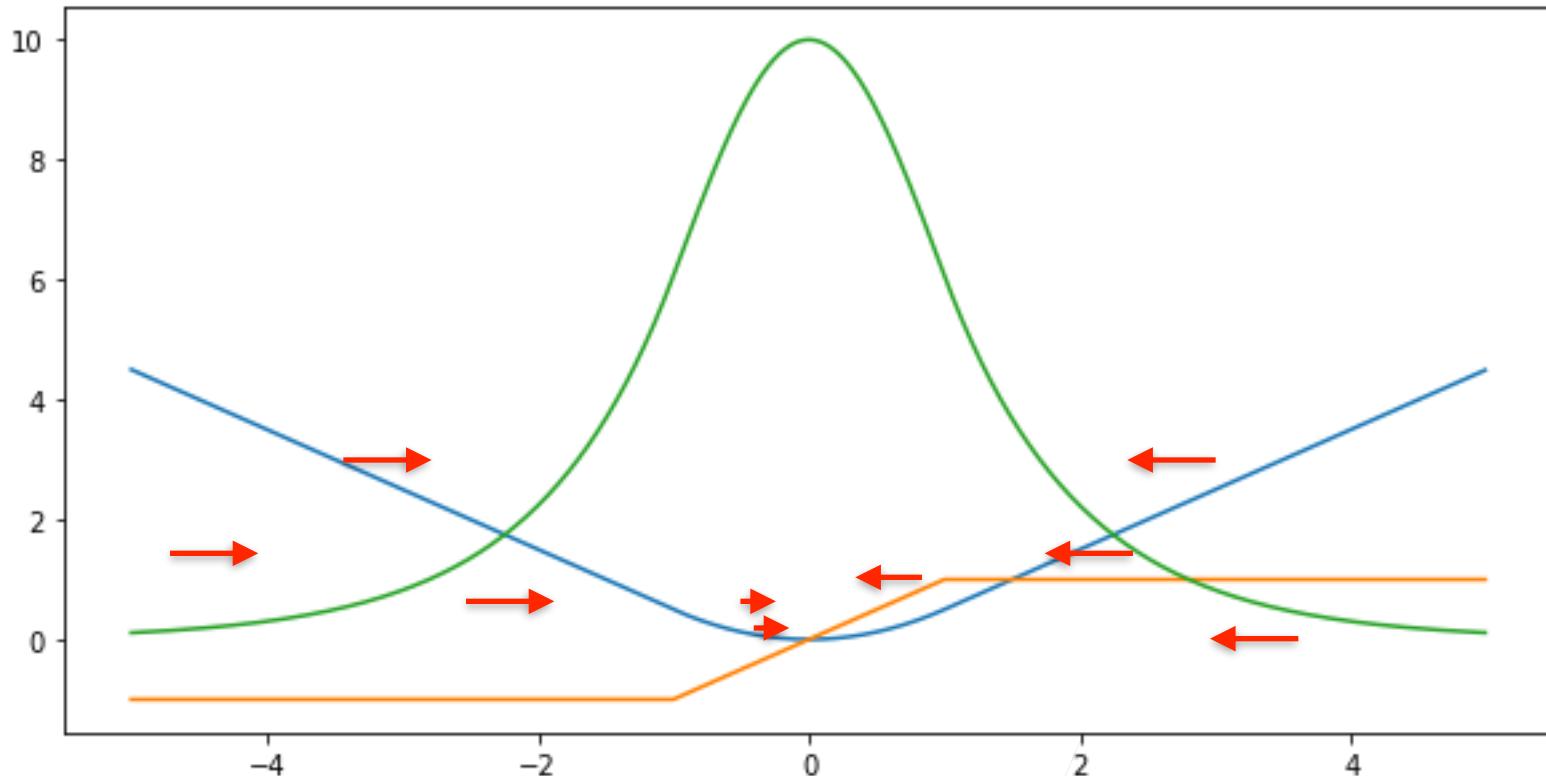


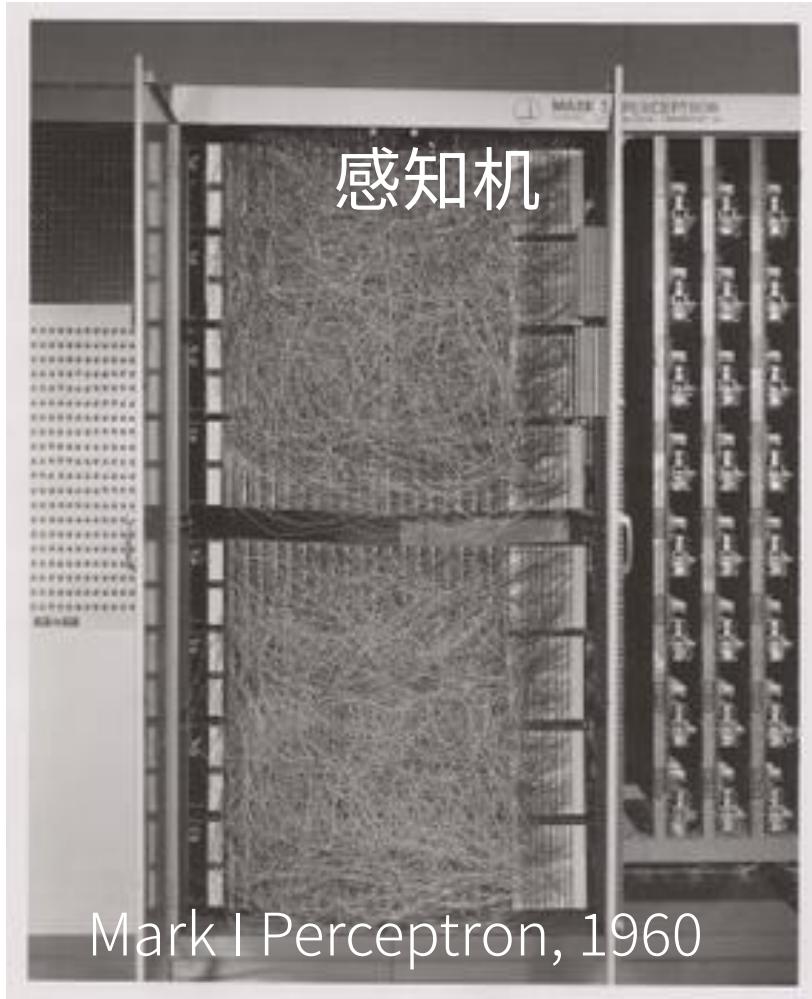
Huber's Robust Loss





Huber's Robust Loss





动手学深度学习 v2

李沐 · AWS

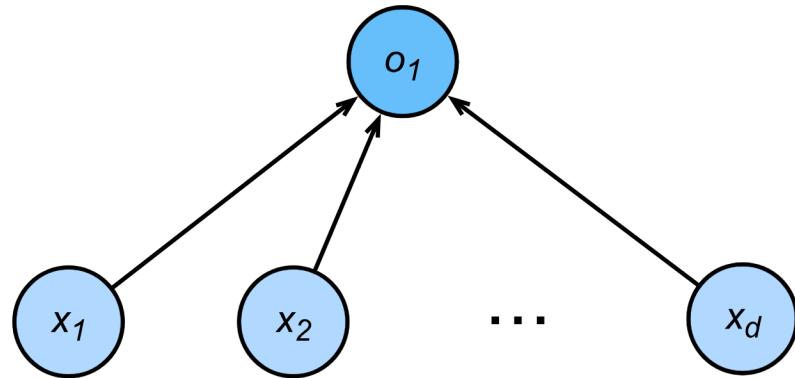




感知机

- 给定输入 \mathbf{x} , 权重 \mathbf{w} , 和偏移 b , 感知机输出:

$$o = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b) \quad \sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$





感知机

- 给定输入 \mathbf{x} , 权重 \mathbf{w} , 和偏移 b , 感知机输出:

$$o = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b) \quad \sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{otherwise} \end{cases}$$

- 二分类: -1 或 1
 - Vs. 回归输出实数
 - Vs. Softmax 回归输出概率



训练感知机

initialize $w = 0$ and $b = 0$

repeat

if $y_i [\langle w, x_i \rangle + b] \leq 0$ **then**

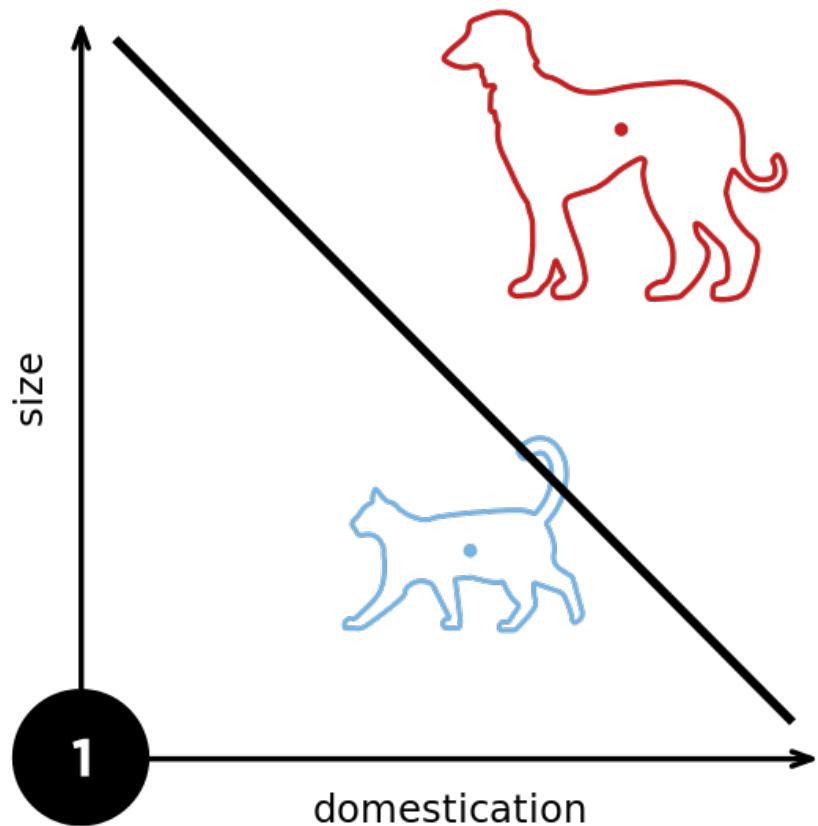
$w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$

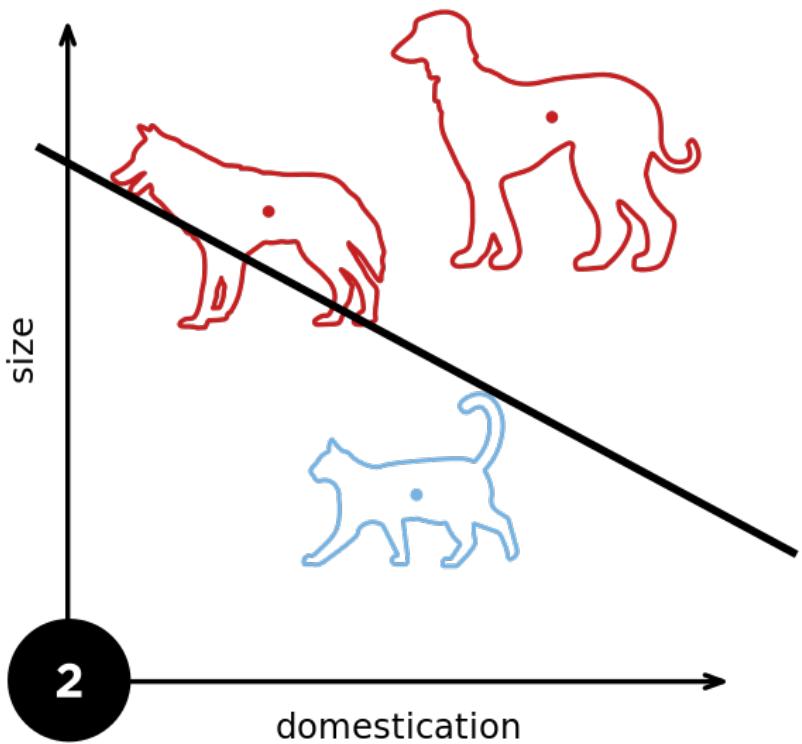
end if

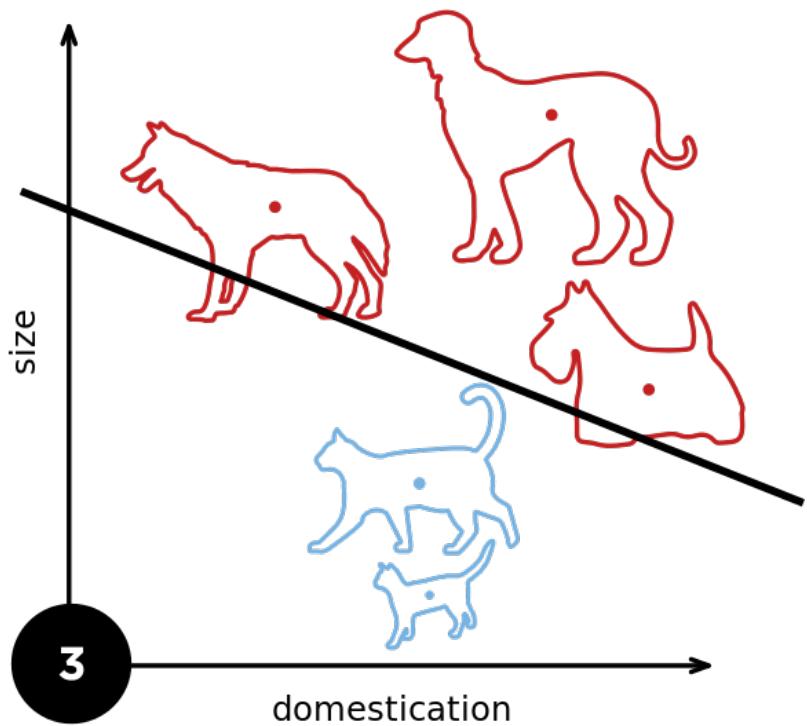
until all classified correctly

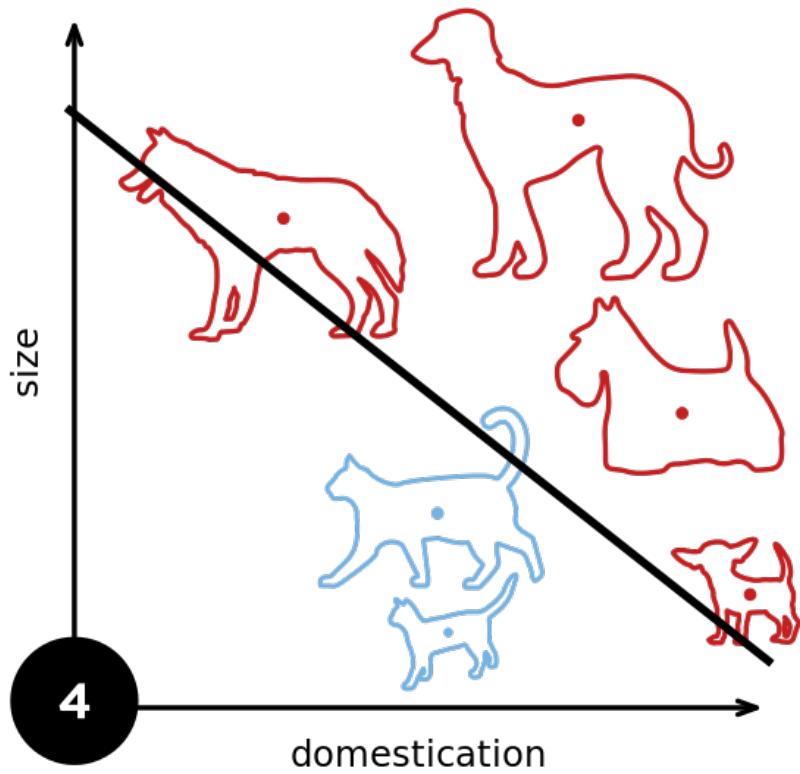
等价于使用批量大小为1的梯度下降，
并使用如下的损失函数

$$\ell(y, \mathbf{x}, \mathbf{w}) = \max(0, -y\langle \mathbf{w}, \mathbf{x} \rangle)$$











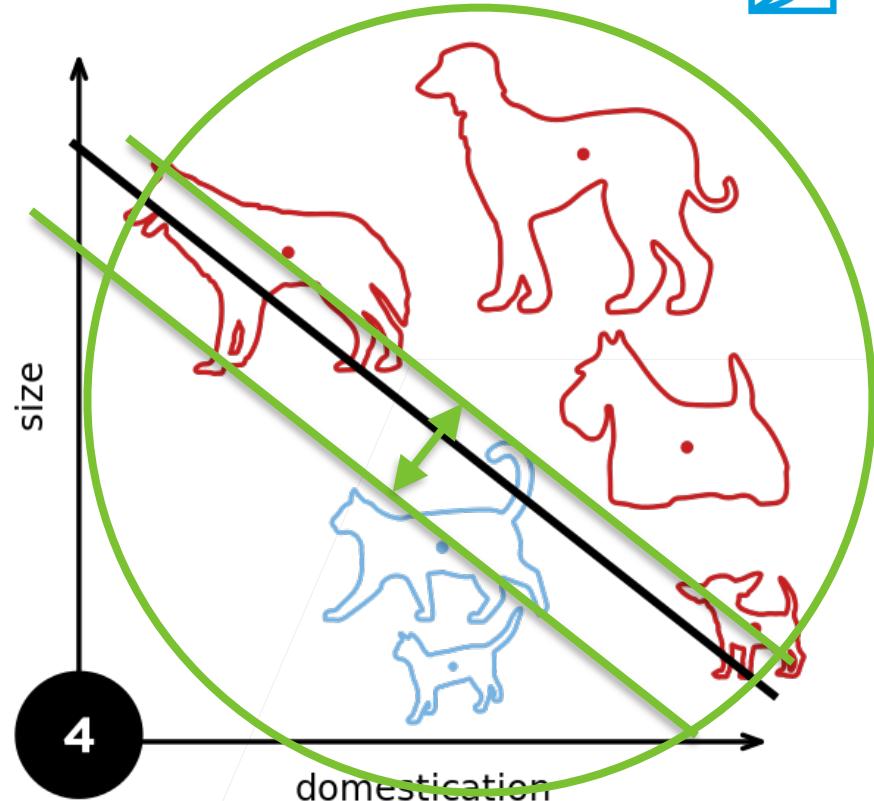
收敛定理

- 数据在半径 r 内
- 余量 ρ 分类两类

$$y(\mathbf{x}^\top \mathbf{w} + b) \geq \rho$$

对于 $\|\mathbf{w}\|^2 + b^2 \leq 1$

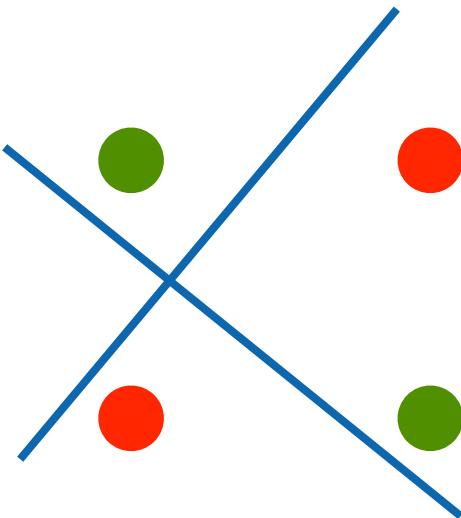
• 感知机保证在 $\frac{r^2 + 1}{\rho^2}$ 步后收敛





XOR 问题 (Minsky & Papert, 1969)

感知机不能拟合 XOR 函数，它只能产生线性分割面



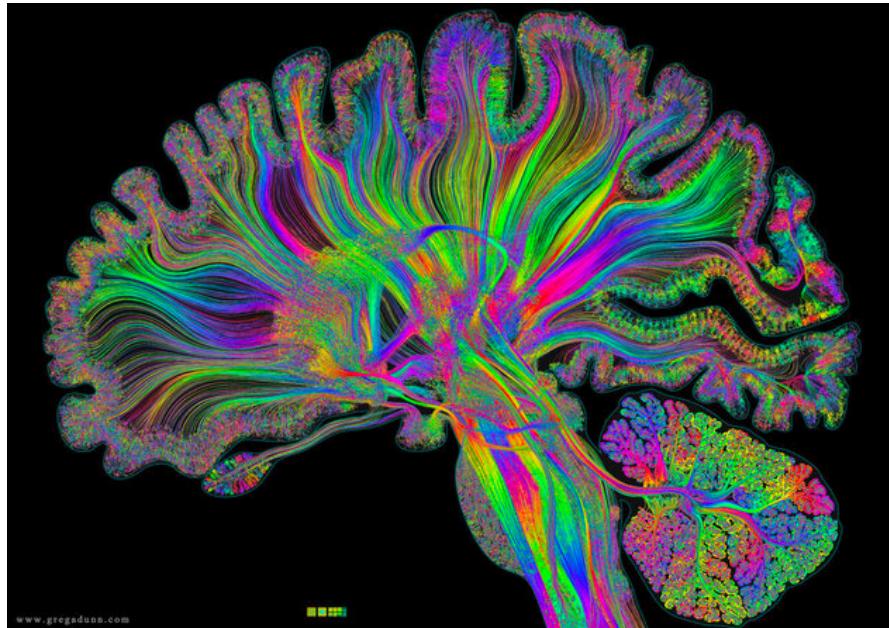


总结

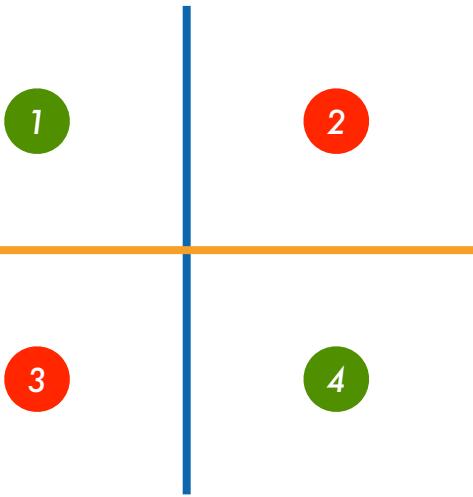
- 感知机是一个二分类模型，是最早的AI模型之一
- 它的求解算法等价于使用批量大小为1的梯度下降
- 它不能拟合 XOR 函数，导致的第一次 AI 寒冬



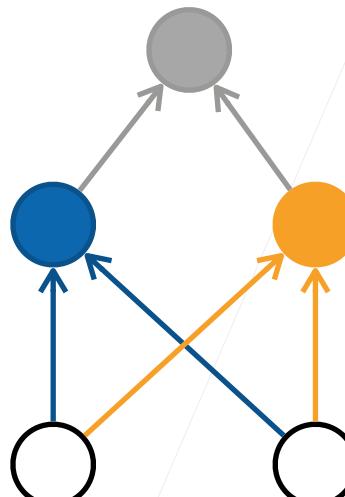
多层感知机



学习 XOR

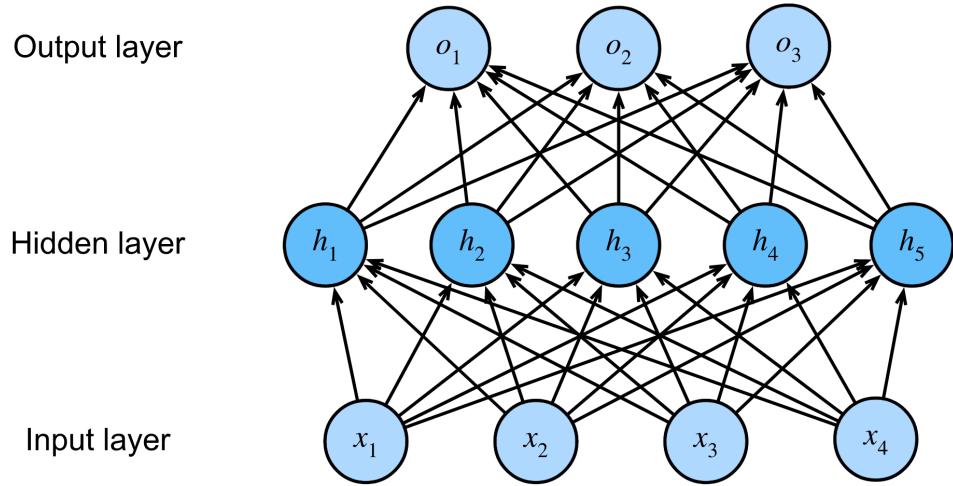


	1	2	3	4
	+	-	+	-
	+	+	-	-
product	+	-	-	+





单隐藏层



隐藏层大小是超参数



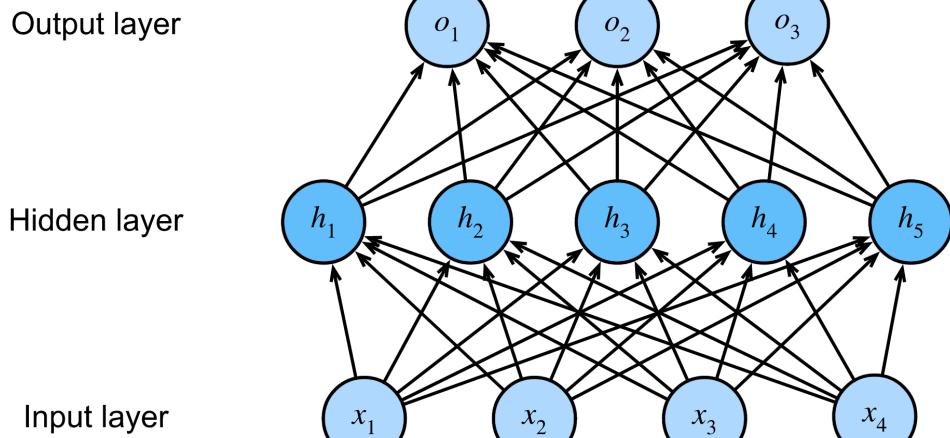
单隐藏层 – 单分类

- 输入 $\mathbf{x} \in \mathbb{R}^n$
- 隐藏层 $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$
- 输出层 $\mathbf{w}_2 \in \mathbb{R}^m, b_2 \in \mathbb{R}$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$o = \mathbf{w}_2^T \mathbf{h} + b_2$$

σ 是按元素的激活函数





单隐藏层

- 输入 $\mathbf{x} \in \mathbb{R}^n$
- 隐藏层 $\mathbf{W}_1 \in \mathbb{R}^{n \times m}, \mathbf{b}_1 \in \mathbb{R}^m$
- 输出层 $\mathbf{w}_2 \in \mathbb{R}^m, b_2 \in \mathbb{R}$

为什么需要非线性
激活函数?

layer

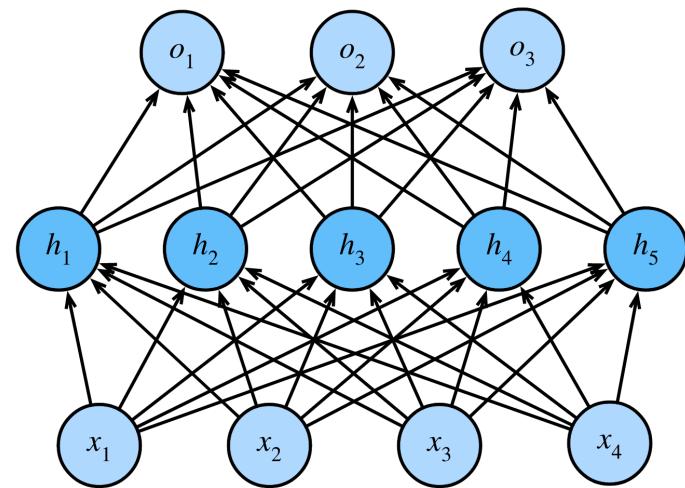
Hidden layer

Input layer

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$o = \mathbf{w}_2^T \mathbf{h} + b_2$$

σ 是按元素的激活函数





单隐藏层

- 输入 $\mathbf{x} \in \mathbb{R}^n$
- 隐藏层 $\mathbf{W}_1 \in \mathbb{R}^{n \times m}, \mathbf{b}_1 \in \mathbb{R}^m$
- 输出层 $\mathbf{w}_2 \in \mathbb{R}^m, b_2 \in \mathbb{R}$

为什么需要非线性
激活函数?

layer

Hidden layer

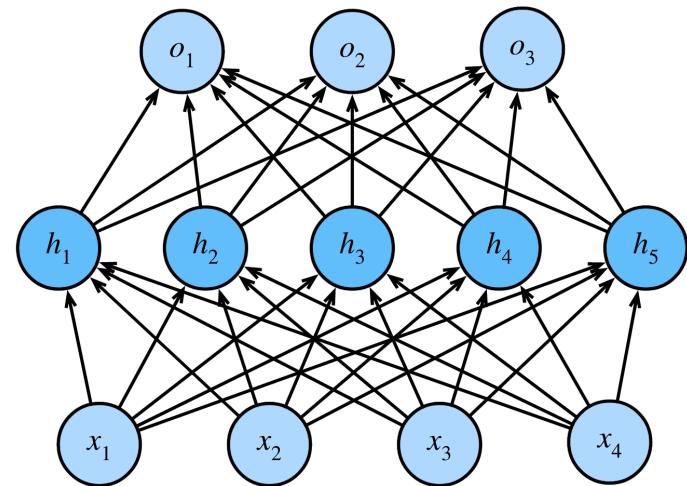
Input layer

$$\mathbf{h} = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1$$

$$o = \mathbf{w}_2^T \mathbf{h} + b_2$$

$$\text{hence } o = \mathbf{w}_2^T \mathbf{W}_1 \mathbf{x} + b'$$

仍然是线性



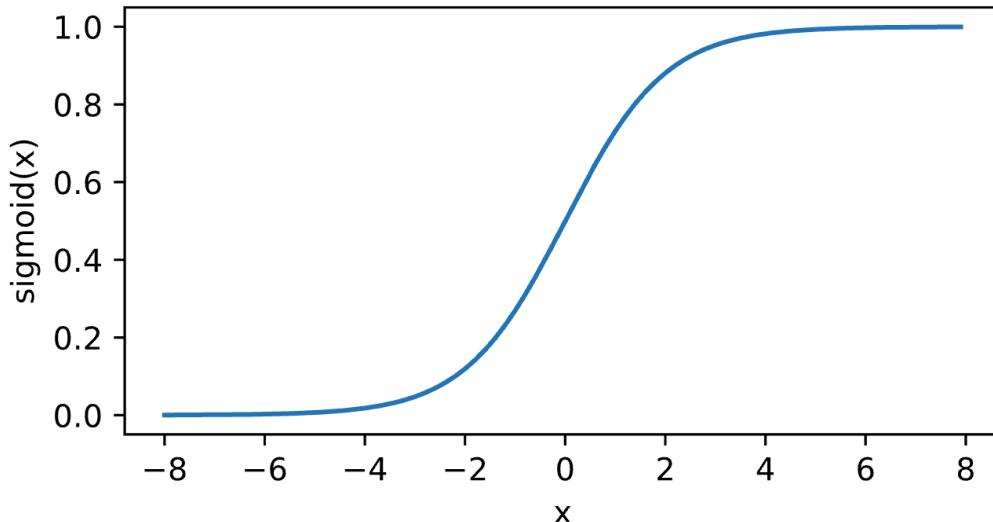


Sigmoid 激活函数

将输入投影到 $(0, 1)$, 是一个软的

$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$

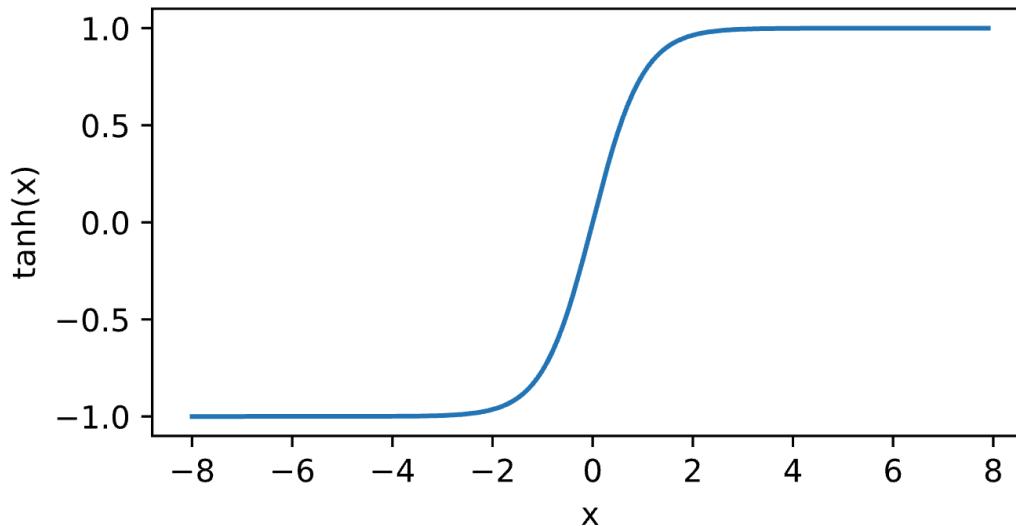




Tanh 激活函数

将输入投影到 (-1, 1)

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$

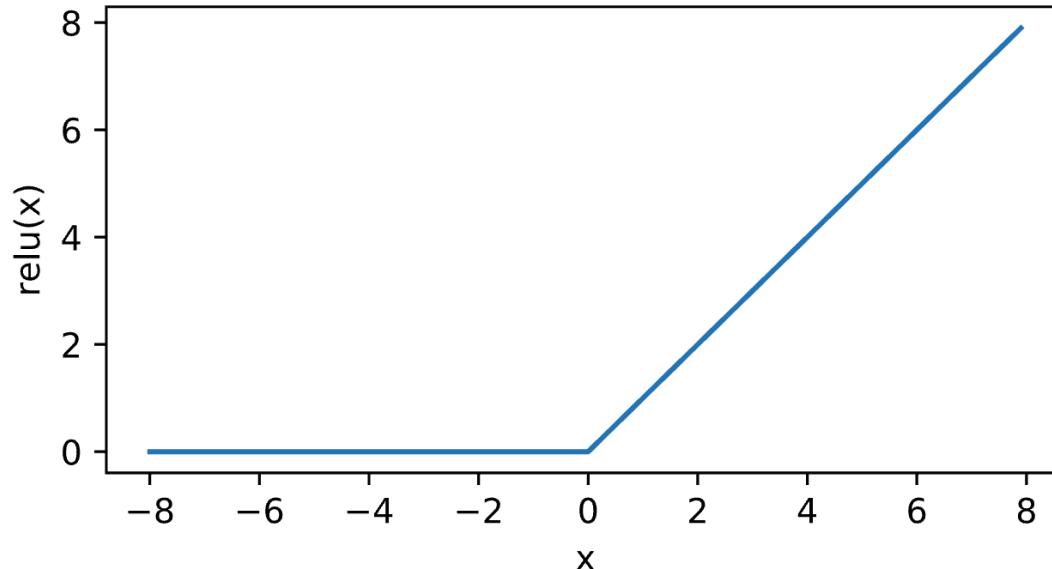




ReLU 激活函数

ReLU: rectified linear unit

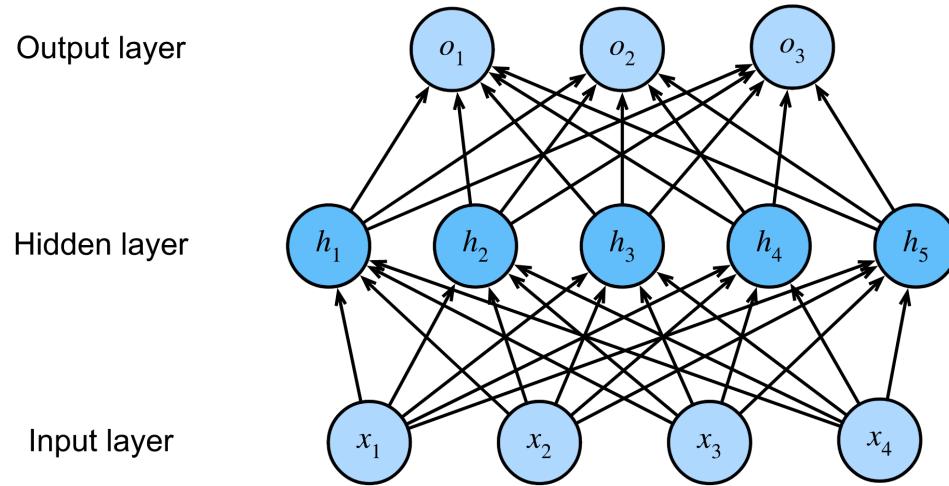
$$\text{ReLU}(x) = \max(x, 0)$$





多类分类

$$y_1, y_2, \dots, y_k = \text{softmax}(o_1, o_2, \dots, o_k)$$





多类分类

- 输入 $\mathbf{x} \in \mathbb{R}^n$
- 隐藏层 $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$
- 输出层 $\mathbf{W}_2 \in \mathbb{R}^{m \times k}, \mathbf{b}_2 \in \mathbb{R}^k$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

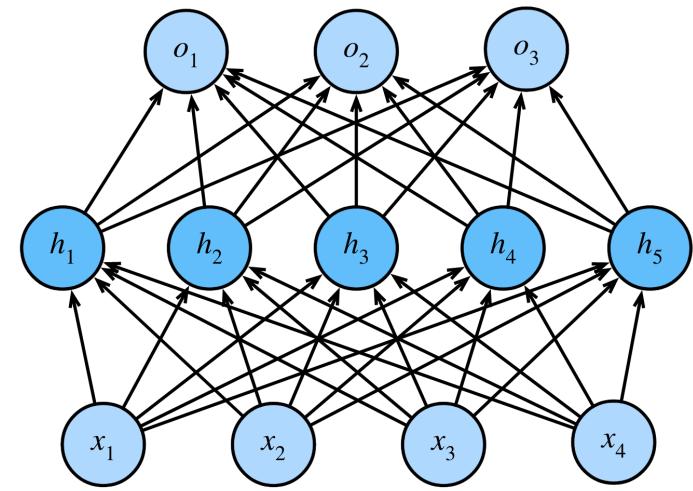
$$\mathbf{o} = \mathbf{W}_2^T \mathbf{h} + \mathbf{b}_2$$

$$\mathbf{y} = \text{softmax}(\mathbf{o})$$

Output layer

Hidden layer

Input layer





多隐藏层

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}_2 = \sigma(\mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2)$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}_3 \mathbf{h}_2 + \mathbf{b}_3)$$

$$\mathbf{o} = \mathbf{W}_4 \mathbf{h}_3 + \mathbf{b}_4$$

超参数

- 隐藏层数
- 每层隐藏层的大小

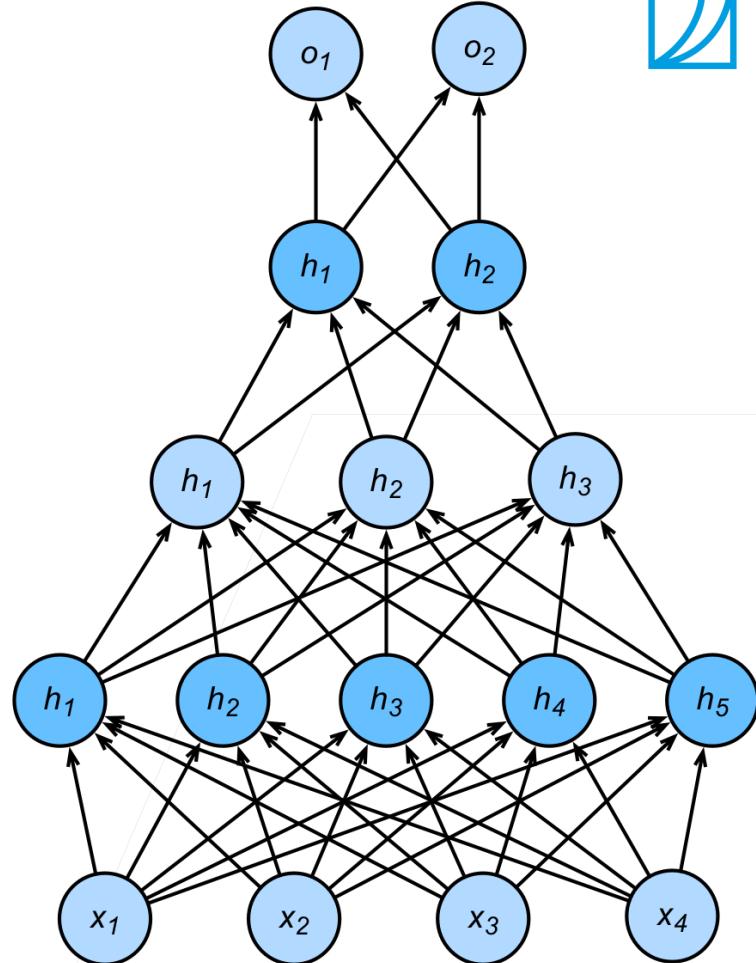
Output layer

Hidden layer

Hidden layer

Hidden layer

Input layer



总结



- 多层感知机使用隐藏层和激活函数来得到非线性模型
- 常用激活函数是Sigmoid, Tanh, ReLU
- 使用 Softmax 来处理多类分类
- 超参数为隐藏层数, 和各个隐藏层大小



模型选择





预测谁会偿还贷款

- 银行雇你来调查谁会偿还贷款
 - 你得到了 100 个申请人的信息
 - 其中五个人在 3 年内违约了





惊讶的发现

- 你发现所有的 5 个人在面试的时候都穿了蓝色衬衫
- 你的模型也发现了这个强信号
- 这会有什么问题？





训练误差和泛化误差

- 训练误差：模型在训练数据上的误差
- 泛化误差：模型在新数据上的误差
- 例子：根据摸考成绩来预测未来考试分数
 - 在过去的考试中表现很好（训练误差）不代表未来考试一定会好（泛化误差）
 - 学生 A 通过背书在摸考中拿到很好成绩
 - 学生 B 知道答案后面的原因



验证数据集和测试数据集

- 验证数据集：一个用来评估模型好坏的数据集
 - 例如拿出 50% 的训练数据
 - 不要跟训练数据混在一起（常犯错误）
- 测试数据集：只用一次的数据集。例如
 - 未来的考试
 - 我出价的房子的实际成交价
 - 用在 Kaggle 私有排行榜中的数据集



K-则交叉验证

- 在没有足够多数据时使用 (这是常态)
- 算法：
 - 将训练数据分割成 K 块
 - For $i = 1, \dots, K$
 - 使用第 i 块作为验证数据集，其余的作为训练数据集
 - 报告 K 个验证集误差的平均
 - 常用： $K = 5$ 或 10

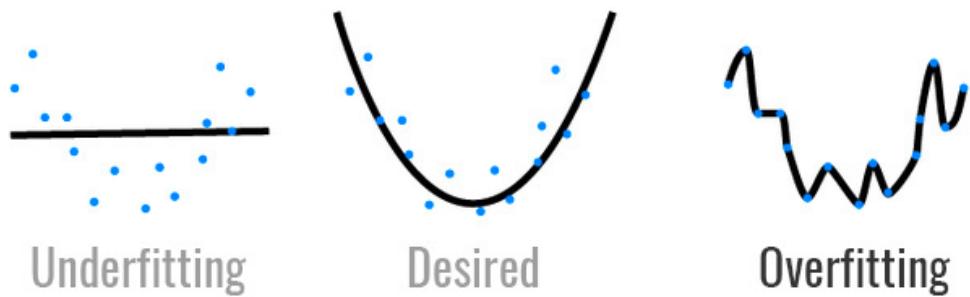


总结

- 训练数据集：训练模型参数
- 验证数据集：选择模型超参数
- 非大数据集上通常使用 k-折交叉验证



过拟合和欠拟合





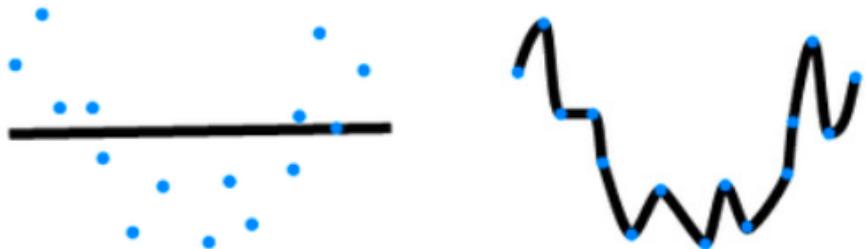
过拟合和欠拟合

		数据	
		简单	复杂
模型容量	低	正常	欠拟合
	高	过拟合	正常



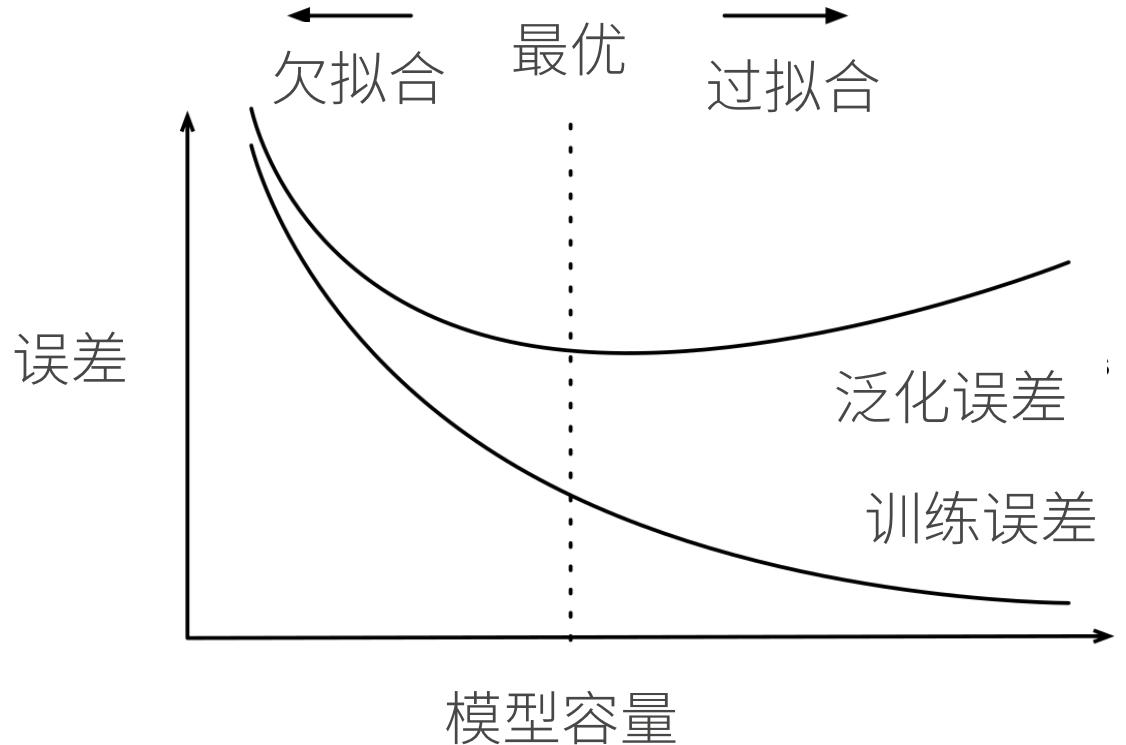
模型容量

- 拟合各种函数的能力
- 低容量的模型难以拟合训练数据
- 高容量的模型可以记住所有的训练数据





模型容量的影响

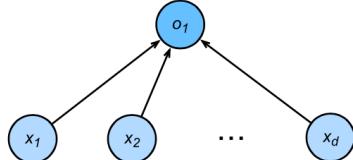




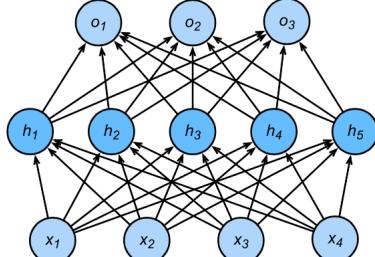
估计模型容量

- 难以在不同的种类算法之间比较
 - 例如数模型和神经网络
- 给定一个模型种类，将有两个主要因素
 - 参数的个数
 - 参数值的选择范围

$$d + 1$$



$$(d + 1)m + (m + 1)k$$



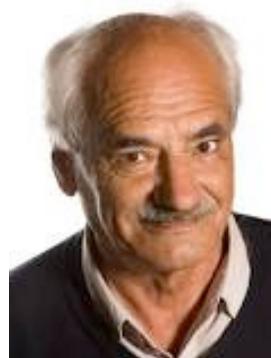


- 统计学习理论的一个核心思想
- 对于一个分类模型，VC等于一个最大的数据集的大小，不管如何给定标号，都存在一个模型来对它进行完美分类

Vladimir Vapnik



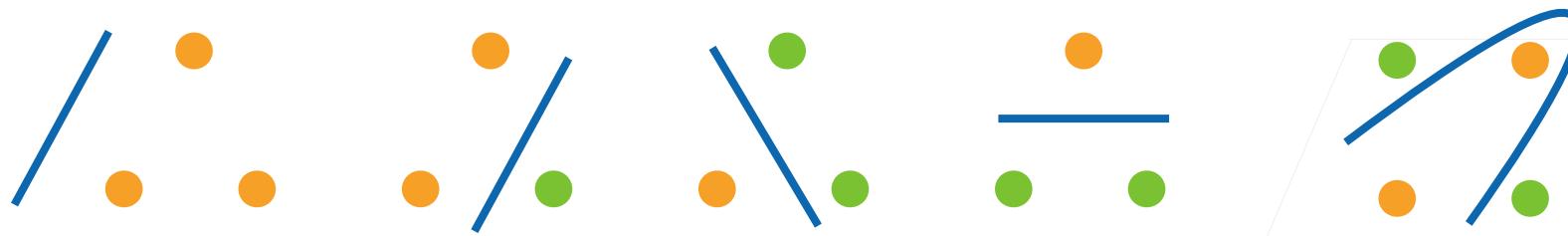
Alexey Chervonenkis





线性分类器的 VC 维

- 2 维输入的感知机，VC 维 = 3
 - 能够分类任何三个点，但不是 4 个 (xor)



- 支持 N 维输入的感知机的 VC 维是 $N + 1$
- 一些多层感知机的 VC 维 $O(N \log_2 N)$



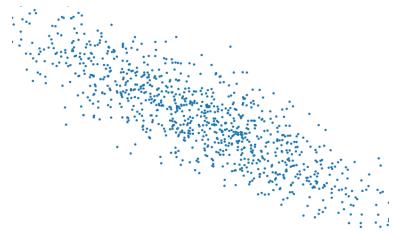
VC 维的用处

- 提供为什么一个模型好的理论依据
 - 它可以衡量训练误差和泛化误差之间的间隔
- 但深度学习中很少使用
 - 衡量不是很准确
 - 计算深度学习模型的 VC 维很困难



数据复杂度

- 多个重要因素
 - 样本个数
 - 每个样本的元素个数
 - 时间、空间结构
 - 多样性



VS



总结

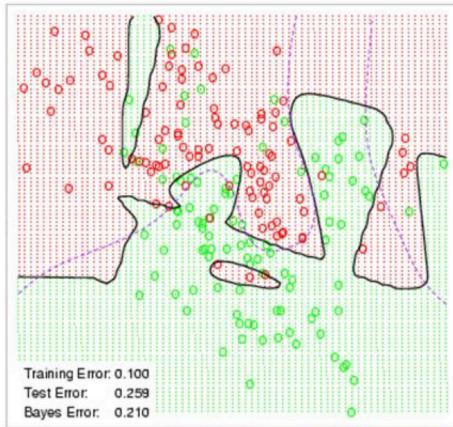


- 模型容量需要匹配数据复杂度，否则可能导致欠拟合和过拟合
- 统计机器学习提供数学工具来衡量模型复杂度
- 实际中一般靠观察训练误差和验证误差

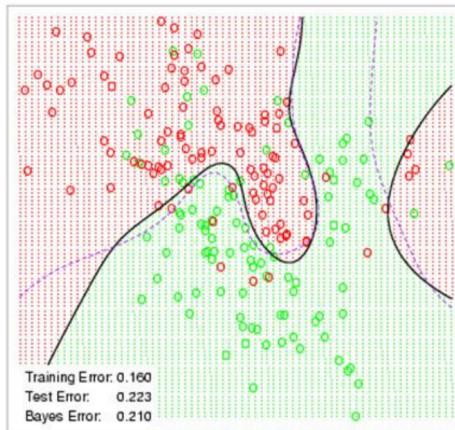


权重衰退

Neural Network - 10 Units, No Weight Decay



Neural Network - 10 Units, Weight Decay=0.02

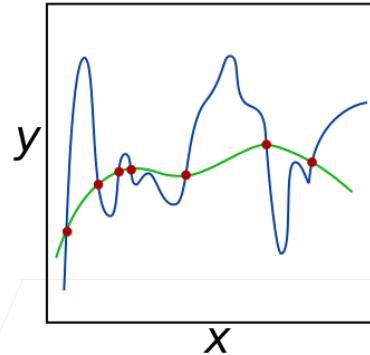




使用均方范数作为硬性限制

- 通过限制参数值的选择范围来控制模型容量

$$\min \ell(\mathbf{w}, b) \text{ subject to } \|\mathbf{w}\|^2 \leq \theta$$



- 通常不限制偏移 b (限不限制都差不多)
- 小的 θ 意味着更强的正则项



使用均方范数作为柔性限制

- 对每个 θ , 都可以找到 λ 使得之前的目标函数等价于下面

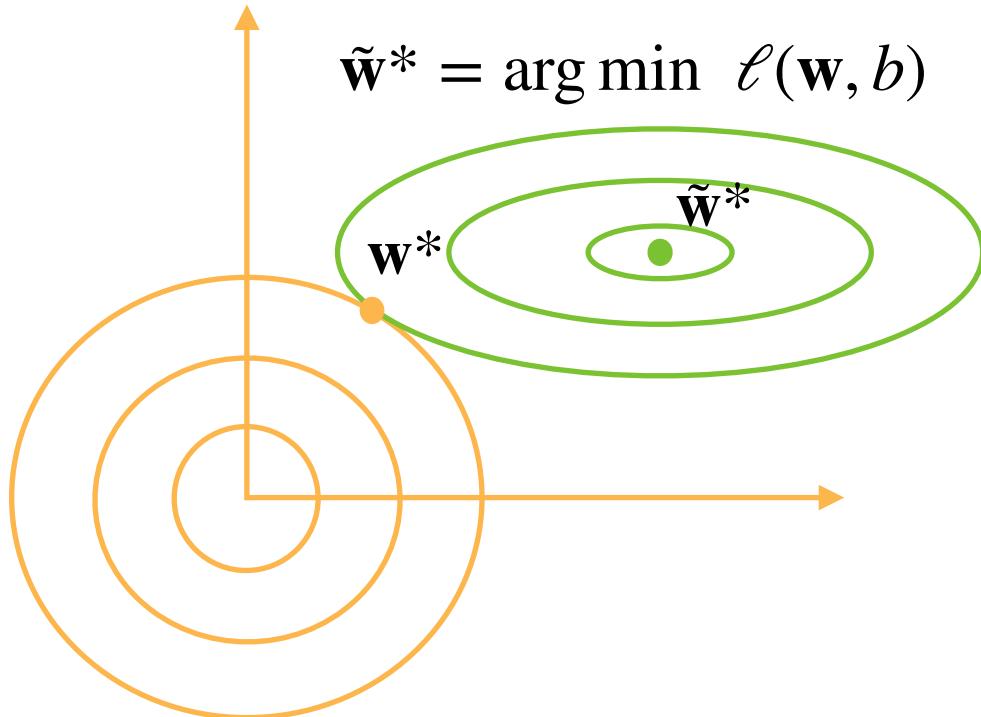
$$\min \ell(\mathbf{w}, b) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

- 可以通过拉格朗日乘子来证明
- 超参数 λ 控制了正则项的重要程度
 - $\lambda = 0$: 无作用
 - $\lambda \rightarrow \infty, \mathbf{w}^* \rightarrow \mathbf{0}$



演示对最优解的影响

$$\mathbf{w}^* = \arg \min \ell(\mathbf{w}, b) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$
$$\tilde{\mathbf{w}}^* = \arg \min \ell(\mathbf{w}, b)$$





参数更新法则

- 计算梯度

$$\frac{\partial}{\partial \mathbf{w}} \left(\ell(\mathbf{w}, b) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \right) = \frac{\partial \ell(\mathbf{w}, b)}{\partial \mathbf{w}} + \lambda \mathbf{w}$$

- 时间 t 更新参数

$$\mathbf{w}_{t+1} = (1 - \eta \lambda) \mathbf{w}_t - \eta \frac{\partial \ell(\mathbf{w}_t, b_t)}{\partial \mathbf{w}_t}$$

- 通常 $\eta \lambda < 1$, 在深度学习中通常叫做权重衰退



总结

- 权重衰退通过 L2 正则项使得模型参数不会过大，从而控制模型复杂度
- 正则项权重是控制模型复杂度的超参数



丢弃法





动机

- 一个好的模型需要对输入数据的扰动鲁棒
 - 使用有噪音的数据等价于 Tikhonov 正则
 - 丢弃法：在层之间加入噪音





无偏差的加入噪音

- 对 \mathbf{x} 加入噪音得到 \mathbf{x}' , 我们希望

$$\mathbf{E}[\mathbf{x}'] = \mathbf{x}$$

- 丢弃法对每个元素进行如下扰动

$$x'_i = \begin{cases} 0 & \text{with probability } p \\ \frac{x_i}{1-p} & \text{otherwise} \end{cases}$$



使用丢弃法

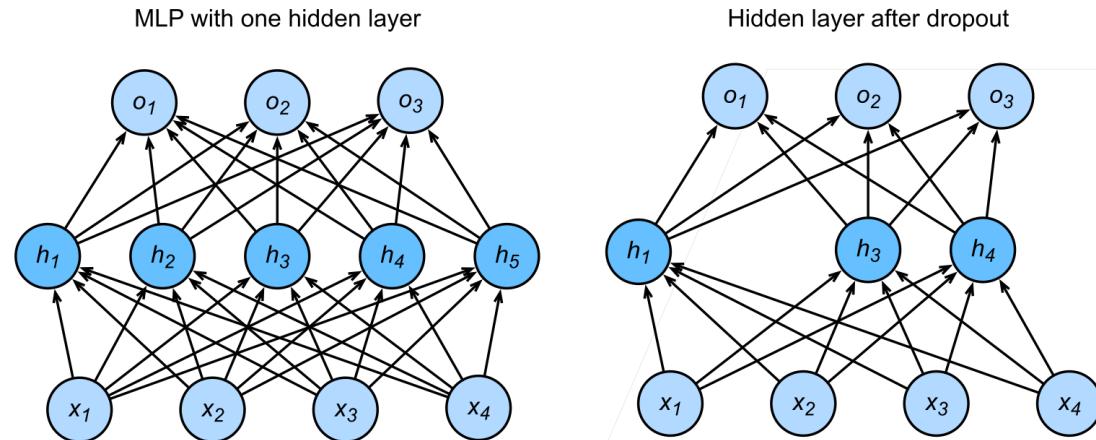
- 通常将丢弃法作用在隐藏全连接层的输出上

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

$$\mathbf{o} = \mathbf{W}_2 \mathbf{h}' + \mathbf{b}_2$$

$$\mathbf{y} = \text{softmax}(\mathbf{o})$$





推理中的丢弃法

- 正则项只在训练中使用：他们影响模型参数的更新
- 在推理过程中，丢弃法直接返回输入

$$\mathbf{h} = \text{dropout}(\mathbf{h})$$

- 这样也能保证确定性的输出



总结

- 丢弃法将一些输出项随机置0来控制模型复杂度
- 常作用在多层感知机的隐藏层输出上
- 丢弃概率是控制模型复杂度的超参数

动手学深度学习 v2

李沐 · AWS



数值稳定性





神经网络的梯度

- 考虑如下有 d 层的神经网络

$$\mathbf{h}^t = f_t(\mathbf{h}^{t-1}) \quad \text{and} \quad y = \ell \circ f_d \circ \dots \circ f_1(\mathbf{x})$$

- 计算损失 ℓ 关于参数 \mathbf{W}_t 的梯度

$$\frac{\partial \ell}{\partial \mathbf{W}^t} = \frac{\partial \ell}{\partial \mathbf{h}^d} \underbrace{\frac{\partial \mathbf{h}^d}{\partial \mathbf{h}^{d-1}} \cdots \frac{\partial \mathbf{h}^{t+1}}{\partial \mathbf{h}^t}}_{\text{d-t 次矩阵乘法}} \frac{\partial \mathbf{h}^t}{\partial \mathbf{W}^t}$$

数值稳定性的常见两个问题

$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i}$$

梯度爆炸



梯度消失



$$1.5^{100} \approx 4 \times 10^{17}$$

$$0.8^{100} \approx 2 \times 10^{-10}$$



例子：MLP

- 加入如下 MLP (为了简单省略了偏移)

$$f_t(\mathbf{h}^{t-1}) = \sigma(\mathbf{W}^t \mathbf{h}^{t-1}) \quad \sigma \text{ 是激活函数}$$

$$\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = \text{diag} (\sigma'(\mathbf{W}^t \mathbf{h}^{t-1})) (\mathbf{W}^t)^T \quad \sigma' \text{ 是 } \sigma \text{ 的导数函数}$$

$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag} (\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})) (\mathbf{W}^i)^T$$



梯度爆炸

- 使用 ReLU 作为激活函数

$$\sigma(x) = \max(0, x) \quad \text{and} \quad \sigma'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

- $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})) (\mathbf{W}^i)^T$ 的一些元素会来自于 $\prod_{i=t}^{d-1} (\mathbf{W}^i)^T$
 - 如果 $d-t$ 很大，值将会很大



梯度爆炸的问题

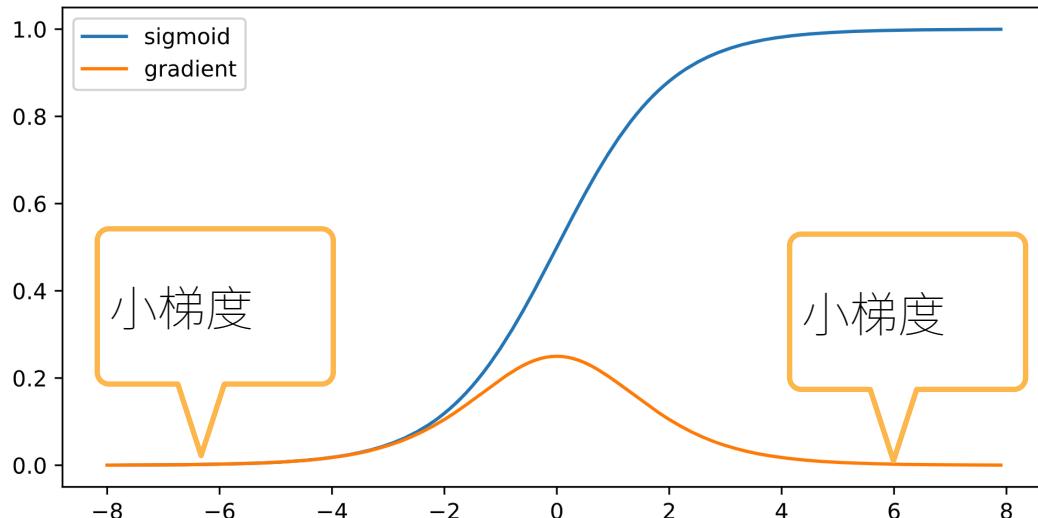
- 值超出值域 (infinity)
 - 对于 16位浮点数尤为严重 (数值区间 $6e-5 - 6e4$)
- 对学习率敏感
 - 如果学习率太大 -> 大参数值 -> 更大的梯度
 - 如果学习率太小 -> 训练无进展
 - 我们可能需要在训练过程不断调整学习率



梯度消失

- 使用 sigmoid 作为激活函数

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$





梯度消失

- 使用 sigmoid 作为激活函数

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$

- $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})) (\mathbf{W}^i)^T$ 的元素值是 $d-t$ 个大小数值的乘积

$$0.8^{100} \approx 2 \times 10^{-10}$$



梯度消失的问题

- 梯度值变成 0
 - 对 16 位浮点数尤为严重
- 训练没有进展
 - 不管如何选择学习率
- 对于底部层尤为严重
 - 仅仅顶部层训练的较好
 - 无法让神经网络更深



总结

- 当数值过大或者过小时会导致数值问题
- 常发生在深度模型中，因为其会对 n 个数累乘

动手学深度学习 v2

李沐 · AWS



让训练更加稳定





让训练更加稳定

- 目标：让梯度值在合理的范围内
 - 例如 $[1e-6, 1e3]$
- 将乘法变加法
 - ResNet, LSTM
- 归一化
 - 梯度归一化， 梯度裁剪
- 合理的权重初始和激活函数



让每层的方差是一个常数

- 将每层的输出和梯度都看做随机变量
- 让它们的均值和方差都保持一致

正向

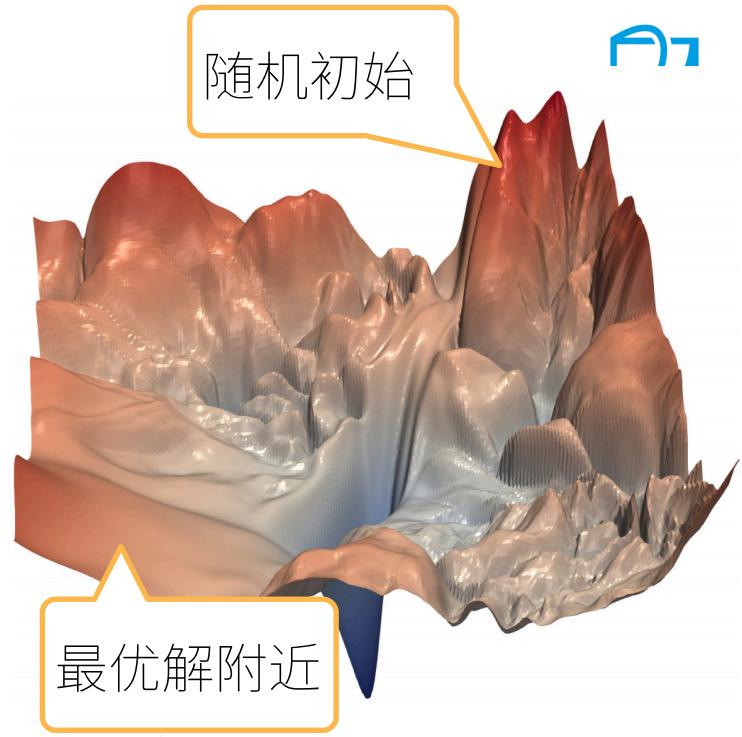
反向

$$\begin{aligned} \mathbb{E}[h_i^t] &= 0 & \mathbb{E} \left[\frac{\partial \ell}{\partial h_i^t} \right] &= 0 & \text{Var} \left[\frac{\partial \ell}{\partial h_i^t} \right] &= b & \forall i, t \\ \text{Var}[h_i^t] &= a \end{aligned}$$

a 和 b 都是常数

权重初始化

- 在合理值区间里随机初始参数
- 训练开始的时候更容易有数值不稳定
 - 远离最优解的地方损失函数表面可能很复杂
 - 最优解附近表面会比较平
- 使用 $\mathcal{N}(0, 0.01)$ 来初始可能对小网络没问题，但不能保证深度神经网络





例子：MLP

- 假设
 - $w_{i,j}^t$ 是 i.i.d，那么 $\mathbb{E}[w_{i,j}^t] = 0$, $\text{Var}[w_{i,j}^t] = \gamma_t$
 - h_i^{t-1} 独立于 $w_{i,j}^t$
- 假设没有激活函数 $\mathbf{h}^t = \mathbf{W}^t \mathbf{h}^{t-1}$, 这里 $\mathbf{W}^t \in \mathbb{R}^{n_t \times n_{t-1}}$

$$\mathbb{E}[h_i^t] = \mathbb{E} \left[\sum_j w_{i,j}^t h_j^{t-1} \right] = \sum_j \mathbb{E}[w_{i,j}^t] \mathbb{E}[h_j^{t-1}] = 0$$



正向方差

$$\begin{aligned}\text{Var}[h_i^t] &= \mathbb{E}[(h_i^t)^2] - \mathbb{E}[h_i^t]^2 = \mathbb{E} \left[\left(\sum_j w_{i,j}^t h_j^{t-1} \right)^2 \right] \\ &= \mathbb{E} \left[\sum_j \left(w_{i,j}^t \right)^2 \left(h_j^{t-1} \right)^2 + \sum_{j \neq k} w_{i,j}^t w_{i,k}^t h_j^{t-1} h_k^{t-1} \right] \\ &= \sum_j \mathbb{E} \left[\left(w_{i,j}^t \right)^2 \right] \mathbb{E} \left[\left(h_j^{t-1} \right)^2 \right] \\ &= \sum_j \text{Var}[w_{i,j}^t] \text{Var}[h_j^{t-1}] = n_{t-1} \gamma_t \text{Var}[h_j^{t-1}]\end{aligned}$$



$$n_{t-1} \gamma_t = 1$$



反向均值和方差

- 跟正向情况类似

$$\frac{\partial \ell}{\partial \mathbf{h}^{t-1}} = \frac{\partial \ell}{\partial \mathbf{h}^t} \mathbf{W}^t \quad \Rightarrow \quad \left(\frac{\partial \ell}{\partial \mathbf{h}^{t-1}} \right)^T = (\mathbf{W}^t)^T \left(\frac{\partial \ell}{\partial \mathbf{h}^t} \right)^T$$

$$\mathbb{E} \left[\frac{\partial \ell}{\partial h_i^{t-1}} \right] = 0$$

$$\text{Var} \left[\frac{\partial \ell}{\partial h_i^{t-1}} \right] = n_t \gamma_t \text{Var} \left[\frac{\partial \ell}{\partial h_j^t} \right] \quad \Rightarrow \quad n_t \gamma_t = 1$$



Xavier 初始

- 难以需要满足 $n_{t-1}\gamma_t = 1$ 和 $n_t\gamma_t = 1$
- Xavier 使得 $\gamma_t(n_{t-1} + n_t)/2 = 1 \rightarrow \gamma_t = 2/(n_{t-1} + n_t)$
 - 正态分布 $\mathcal{N}\left(0, \sqrt{2/(n_{t-1} + n_t)}\right)$
 - 均匀分布 $\mathcal{U}\left(-\sqrt{6/(n_{t-1} + n_t)}, \sqrt{6/(n_{t-1} + n_t)}\right)$
 - 分布 $\mathcal{U}[-a, a]$ 和方差是 $a^2/3$
- 适配权重形状变换，特别是 n_t



假设线性的激活函数

- 假设 $\sigma(x) = \alpha x + \beta$

$$\mathbf{h}' = \mathbf{W}^t \mathbf{h}^{t-1} \quad \text{and} \quad \mathbf{h}^t = \sigma(\mathbf{h}')$$

$$\mathbb{E}[h_i^t] = \mathbb{E} [\alpha h'_i + \beta] = \beta \qquad \Rightarrow \qquad \beta = 0$$

$$\begin{aligned}\text{Var}[h_i^t] &= \mathbb{E}[(h_i^t)^2] - \mathbb{E}[h_i^t]^2 \\ &= \mathbb{E}[(\alpha h'_i + \beta)^2] - \beta^2 \qquad \Rightarrow \qquad \alpha = 1 \\ &= \mathbb{E}[\alpha^2(h'_i)^2 + 2\alpha\beta h'_i + \beta^2] - \beta^2 \\ &= \alpha^2 \text{Var}[h'_i]\end{aligned}$$



反向

- 假设 $\sigma(x) = \alpha x + \beta$

$$\frac{\partial \ell}{\partial \mathbf{h}'} = \frac{\partial \ell}{\partial \mathbf{h}^t} (\mathbf{W}^t)^T \quad \text{and} \quad \frac{\partial \ell}{\partial \mathbf{h}^{t-1}} = \alpha \frac{\partial \ell}{\partial \mathbf{h}'}$$

$$\mathbb{E} \left[\frac{\partial \ell}{\partial h_i^{t-1}} \right] = 0 \qquad \Rightarrow \qquad \beta = 0$$

$$\text{Var} \left[\frac{\partial \ell}{\partial h_i^{t-1}} \right] = \alpha^2 \text{Var} \left[\frac{\partial \ell}{\partial h_j'} \right] \quad \Rightarrow \quad \alpha = 1$$



检查常用激活函数

- 使用泰勒展开

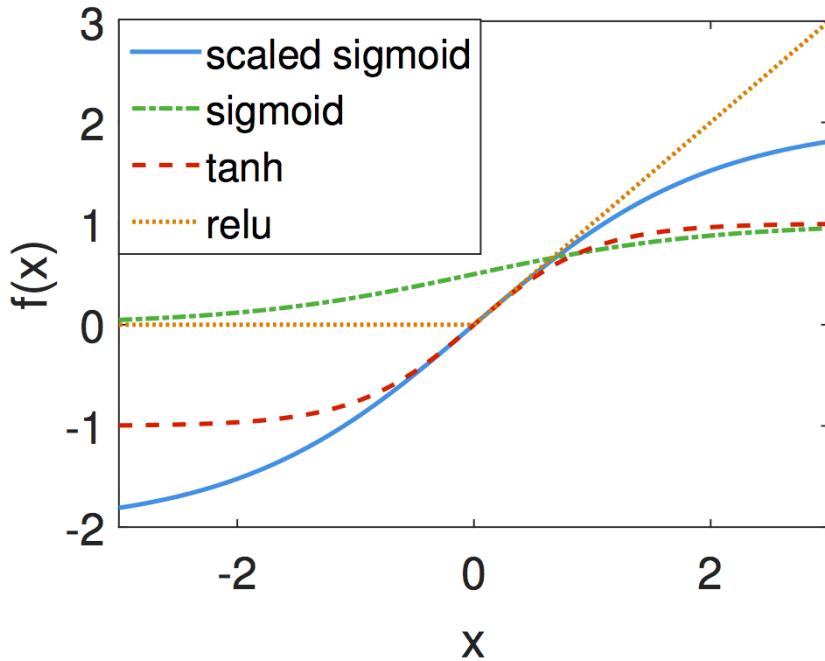
$$\text{sigmoid}(x) = \frac{1}{2} + \frac{x}{4} - \frac{x^3}{48} + O(x^5)$$

$$\tanh(x) = 0 + x - \frac{x^3}{3} + O(x^5)$$

$$\text{relu}(x) = 0 + x \quad \text{for } x \geq 0$$

- 调整 sigmoid:

$$4 \times \text{sigmoid}(x) - 2$$





总结

- 合理的权重初始值和激活函数的选取可以提升数值稳定性



竞赛：房价预测





规则

- 网址: <https://www.kaggle.com/c/california-housing-prices/overview>
- 前5个人（团队）将获赠作者签名版《动手学深度学习》第二版
- 欢迎分享经验（学习第一，结果第二）



竞赛：房价预测总结





结果

- 172只队伍， 2288次提交
- 前5位同学将获得奖品

#	△pub	Team Name	Notebook	Team Members	Score ⓘ	Entries	Last
1	—	fxzero			0.11922	5	5d
2	▲ 1	sxjscience			0.12063	2	2mo
3	▲ 1	wuwawa			0.12283	77	5d
4	▲ 5	Leon			0.12283	163	5d
5	▲ 6	jack			0.12455	21	5d
6	▲ 2	totoro			0.12485	6	5d
7	▲ 8	automl (baseline)			0.12502	1	2mo
8	▼ 2	pulpullyes			0.12518	123	5d
9	▲ 5	Shaoqing			0.12559	84	5d



方法总结

- 第二和第七：autogluon

<https://www.bilibili.com/video/BV1rh411m7Hb/>

- 第三：h2o

<https://www.kaggle.com/wuwawa/automl-using-h2o>

- 第四：随机森林

<https://www.kaggle.com/jackzh/the-4th-place-approach-random-forest>



一些分析

- 已知的排名靠前的4个成绩均使用了集成学习
- 目前不知道是否有使用书中的mlp取得好成绩
 - 特征预处理和超参数是取得好成绩的基础
- 这个数据的一些难点
 - 数值较大
 - 有文本特征（地址，介绍）
 - 训练数据是前6个月，公榜是后3个月，私榜是再后三个月



关于automl

- 数据科学家80%时间在处理数据， 20%调模型
- Automl现在能处理一些基础的情况
 - 目前节省10%时间， 未来节省20%时间
- 为什么还要学习模型
 - (我初中时说：会加减乘除就可以买菜了，为什么还要学三角函数)
 - 当人人都会玩游戏时， 你需要成为玩得最好的，或者甚至去制作游戏

卷积

李沐 · AWS





分类猫和狗的图片

- 使用一个还不错的相机采集图片 (12M 像素)
- RGB 图片有 36M 元素
- 使用 100 大小的单隐藏层 MLP，模型有 3.6B 元素
 - 远多于世界上所有猫和狗总数 (900M 狗, 600M 猫)



VS





回顾：单隐藏层 MLP

100 神经元

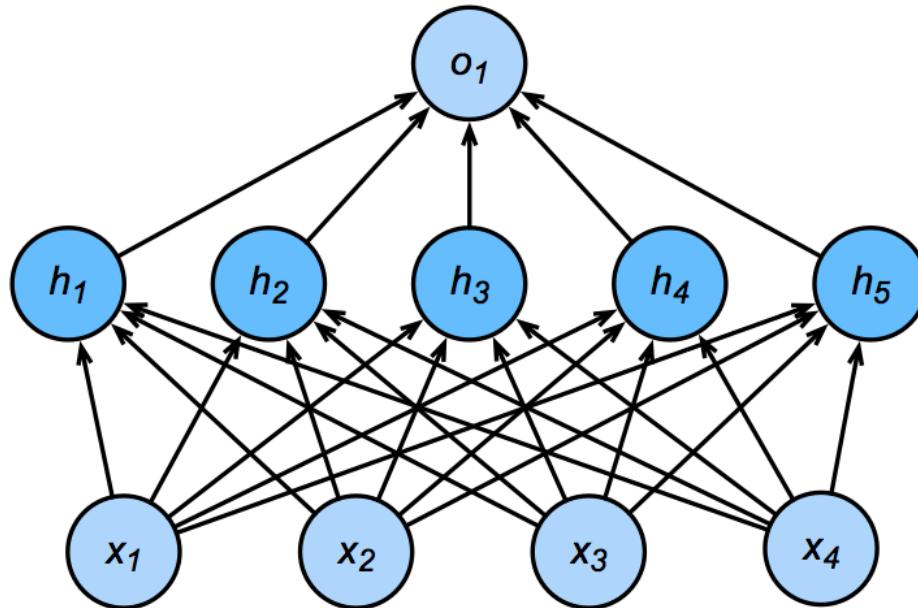
Output layer

Hidden layer

3.6B 参数 = 14GB

36M 特征

Input layer



$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Waldo 在
哪里？



两个原则

- 平移不变性
- 局部性





重新考察全连接层

- 将输入和输出变形为矩阵（宽度，高度）
- 将权重变形为4-D张量 (h, w) 到 (h', w')

$$h_{i,j} = \sum_{k,l} w_{i,j,k,l} x_{k,l} = \sum_{a,b} v_{i,j,a,b} x_{i+a, j+b}$$

- V 是 W 的重新索引 $v_{i,j,a,b} = w_{i,j,i+a,j+b}$



原则 #1 - 平移不变性

- x 的平移导致 h 的平移 $h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a,j+b}$
- v 不应该依赖于 (i, j)
- 解决方案: $v_{i,j,a,b} = v_{a,b}$

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a,j+b}$$

这就是 2 维卷积 交叉相关



原则 #2 - 局部性

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a, j+b}$$

- 当评估 $h_{i,j}$ 时，我们不应该用远离 $x_{i,j}$ 的参数
- 解决方案：当 $|a|, |b| > \Delta$ 时，使得 $v_{a,b} = 0$

$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a, j+b}$$



总结

- 对全连接层使用平移不变性和局部性得到卷积层

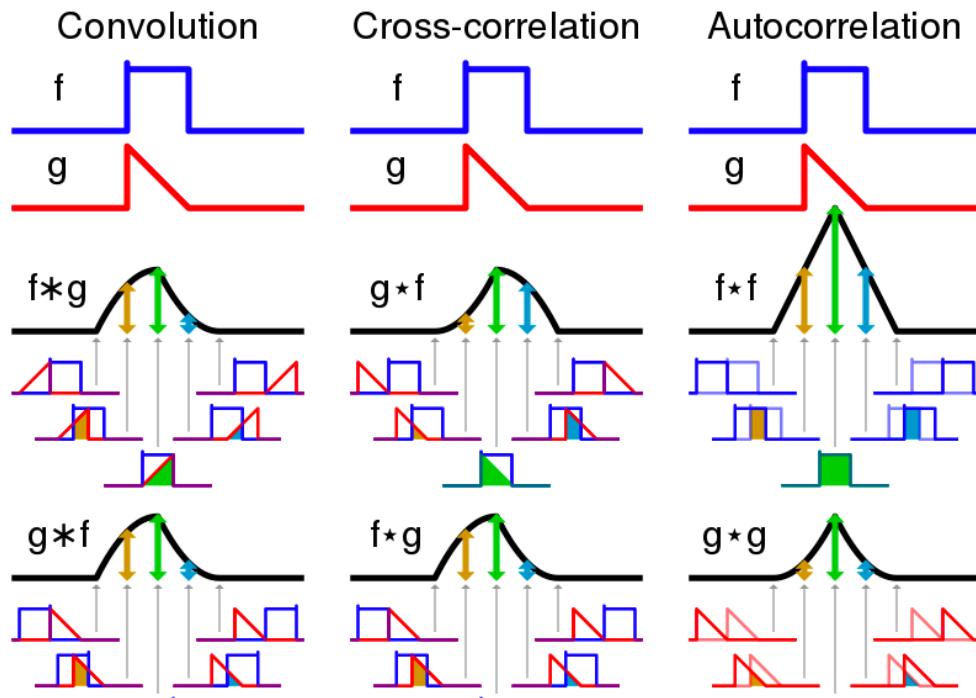
$$h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a,j+b}$$



$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a,j+b}$$



卷积层





二维交叉相关

Input Kernel Output

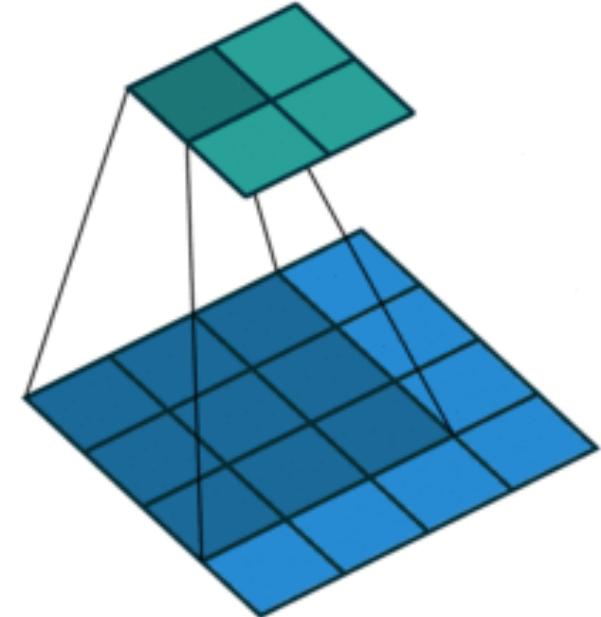
$$\begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 0 & 1 \\ \hline 2 & 3 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 19 & 25 \\ \hline 37 & 43 \\ \hline \end{array}$$

$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$

$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$

$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$

$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$



(vdumoulin@ Github)



二维卷积层

$$\begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 0 & 1 \\ \hline 2 & 3 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 19 & 25 \\ \hline 37 & 43 \\ \hline \end{array}$$

- 输入 \mathbf{X} : $n_h \times n_w$
- 核 \mathbf{W} : $k_h \times k_w$
- 偏差 $b \in \mathbb{R}$
- 输出 \mathbf{Y} : $(n_h - k_h + 1) \times (n_w - k_w + 1)$

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

- \mathbf{W} 和 b 是可学习的参数



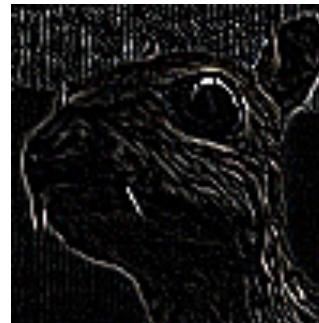
例子

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



(wikipedia)

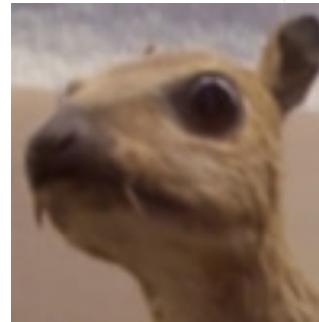
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



边缘检测



锐化



高斯模糊

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



交叉相关 vs 卷积

- 二维交叉相关

$$y_{i,j} = \sum_{a=1}^h \sum_{b=1}^w w_{a,b} x_{i+a, j+b}$$

- 二维卷积

$$y_{i,j} = \sum_{a=1}^h \sum_{b=1}^w w_{-a,-b} x_{i+a, j+b}$$

- 由于对称性，在实际使用中没有区别



一维和三维交叉相关

- 一维

$$y_i = \sum_{a=1}^h w_a x_{i+a}$$

- 三维

$$y_{i,j,k} = \sum_{a=1}^h \sum_{b=1}^w \sum_{c=1}^d w_{a,b,c} x_{i+a, j+b, k+c}$$

- 文本
- 语言
- 时序序列

- 视频
- 医学图像
- 气象地图

总结



- 卷积层将输入和核矩阵进行交叉相关，加上偏移后得到输出
- 核矩阵和偏移是可学习的参数
- 核矩阵的大小是超参数



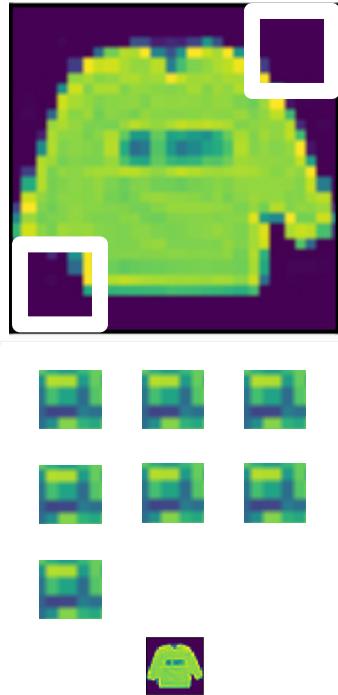
李沐 · AWS

填充和步幅



填充

- 给定 (32×32) 输入图像
- 应用 5×5 大小的 卷积核
 - 第1层得到输出大小 28×28
 - 第7层得到输出大小 4×4
- 更大的卷积核可以更快地减小输出大小
 - 形状从 $n_h \times n_w$ 减少到
$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$



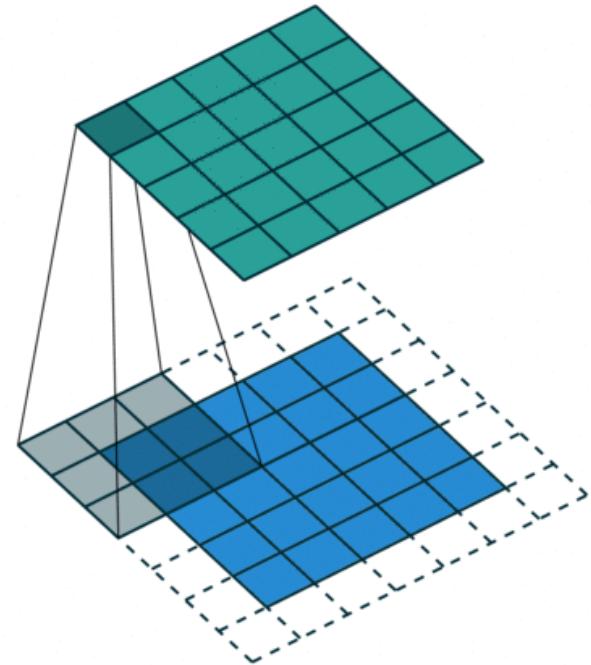


填充

在输入周围添加额外的行 / 列

Input					Kernel		Output					
0	0	0	0	0	*	0	1	=	0	3	8	4
0	0	1	2	0		2	3		9	19	25	10
0	3	4	5	0					21	37	43	16
0	6	7	8	0					6	7	8	0
0	0	0	0	0								

$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$





填充

- 填充 p_h 行和 p_w 列，输出形状为

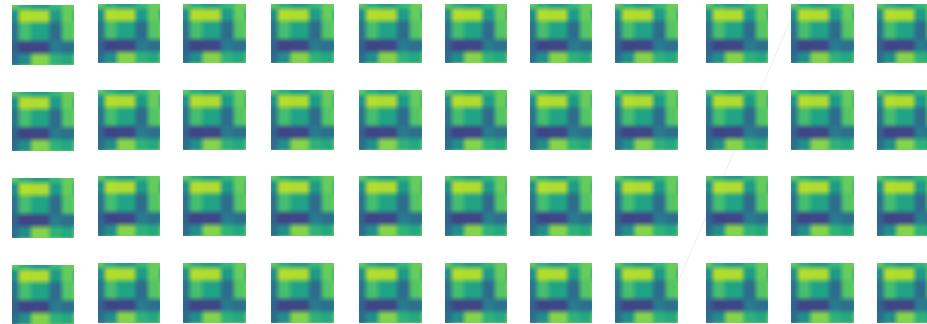
$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

- 通常取 $p_h = k_h - 1$, $p_w = k_w - 1$
 - 当 k_h 为奇数：在上下两侧填充 $p_h/2$
 - 当 k_h 为偶数：在上侧填充 $\lceil p_h/2 \rceil$ ，在下侧填充 $\lfloor p_h/2 \rfloor$



步幅

- 填充减小的输出大小与层数线性相关
 - 给定输入大小 224×224 ，在使用 5×5 卷积核的情况下，需要 44 层将输出降低到 4×4
 - 需要大量计算才能得到较小输出





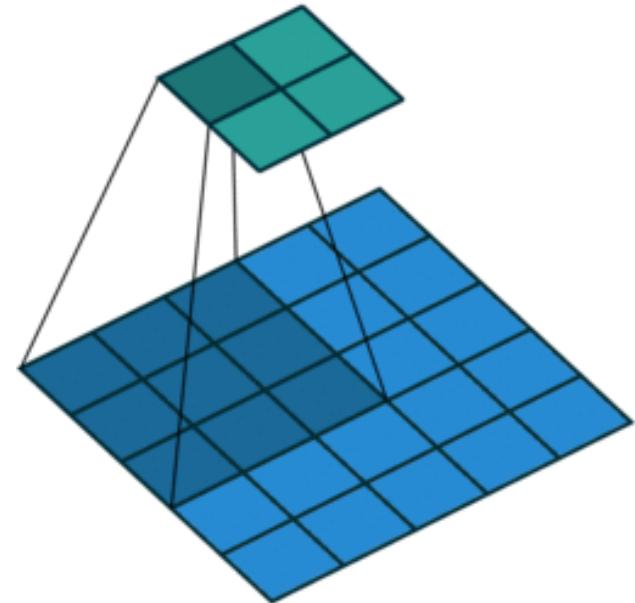
步幅

- 步幅是指行/列的滑动步长
 - 例：高度3 宽度2 的步幅

Input	Kernel	Output																													
<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>2</td><td>0</td></tr><tr><td>0</td><td>3</td><td>4</td><td>5</td><td>0</td></tr><tr><td>0</td><td>6</td><td>7</td><td>8</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	1	2	0	0	3	4	5	0	0	6	7	8	0	0	0	0	0	0	$*$	<table border="1"><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr></table>	0	1	2	3
0	0	0	0	0																											
0	0	1	2	0																											
0	3	4	5	0																											
0	6	7	8	0																											
0	0	0	0	0																											
0	1																														
2	3																														
	$=$	<table border="1"><tr><td>0</td><td>8</td></tr><tr><td>6</td><td>8</td></tr></table>	0	8	6	8																									
0	8																														
6	8																														

$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$

$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$





步幅

- 给定高度 s_h 和宽度 s_w 的步幅，输出形状是

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

- 如果 $p_h = k_h - 1$, $p_w = k_w - 1$

$$\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$$

- 如果输入高度和宽度可以被步幅整除

$$(n_h/s_h) \times (n_w/s_w)$$



总结

- 填充和步幅是卷积层的超参数
- 填充在输入周围添加额外的行 / 列，来控制输出形状的减少量
- 步幅是每次滑动核窗口时的行/列的步长，可以成倍的减少输出形状



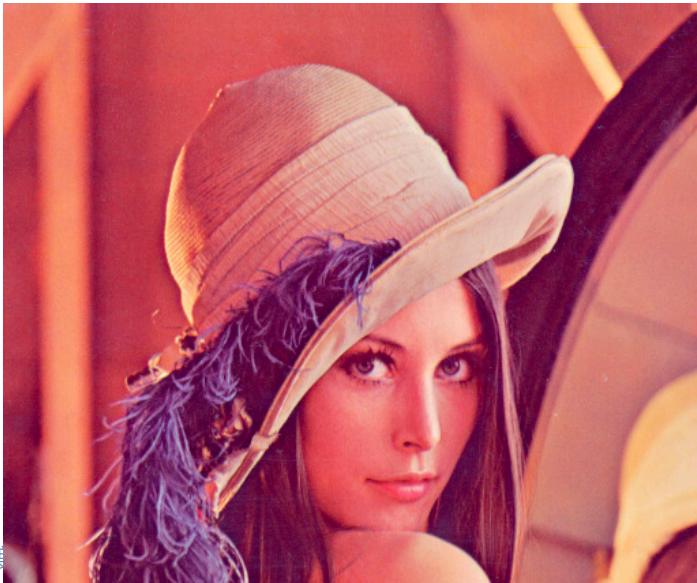
李沐 · AWS

多个输入和
输出通道



多个输入通道

- 彩色图像可能有 RGB 三个通道
- 转换为灰度会丢失信息





多个输入通道

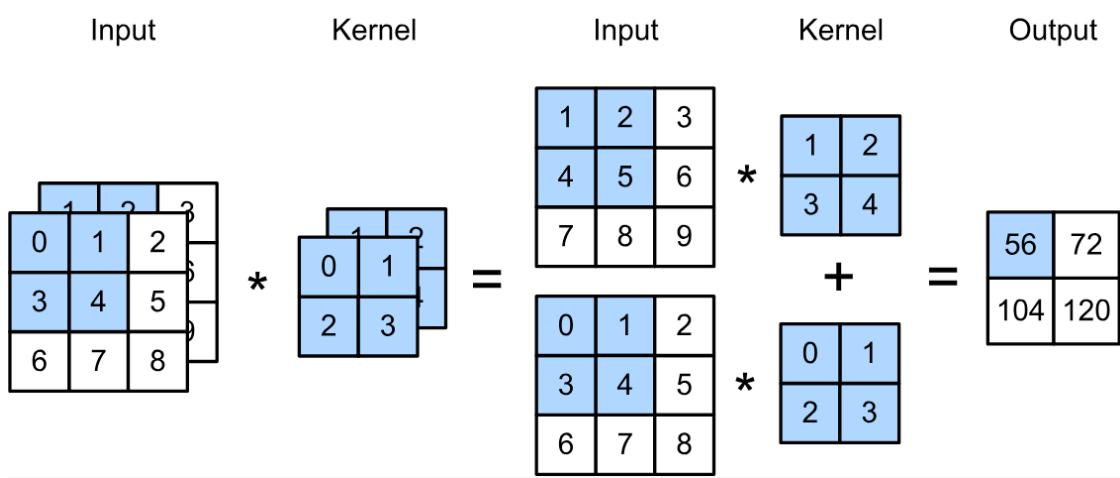
- 彩色图像可能有 RGB 三个通道
- 转换为灰度会丢失信息





多个输入通道

- 每个通道都有一个卷积核，结果是所有通道卷积结果的和



$$(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4)$$

$$+(0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3) = 56$$



多个输入通道

- 输入 \mathbf{X} : $c_i \times n_h \times n_w$
- 核 \mathbf{W} : $c_i \times k_h \times k_w$
- 输出 \mathbf{Y} : $m_h \times m_w$

$$\mathbf{Y} = \sum_{i=0}^{c_i} \mathbf{X}_{i,:,:} \star \mathbf{W}_{i,:,:}$$



多个输出通道

- 无论有多少输入通道，到目前为止我们只用到单输出通道
- 我们可以有多个三维卷积核，每个核生成一个输出通道
- 输入 $\mathbf{X} : c_i \times n_h \times n_w$
- 核 $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
- 输出 $\mathbf{Y} : c_o \times m_h \times m_w$

$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:} \quad \text{for } i = 1, \dots, c_o$$



多个输入和输出通道

- 每个输出通道可以识别特定模式

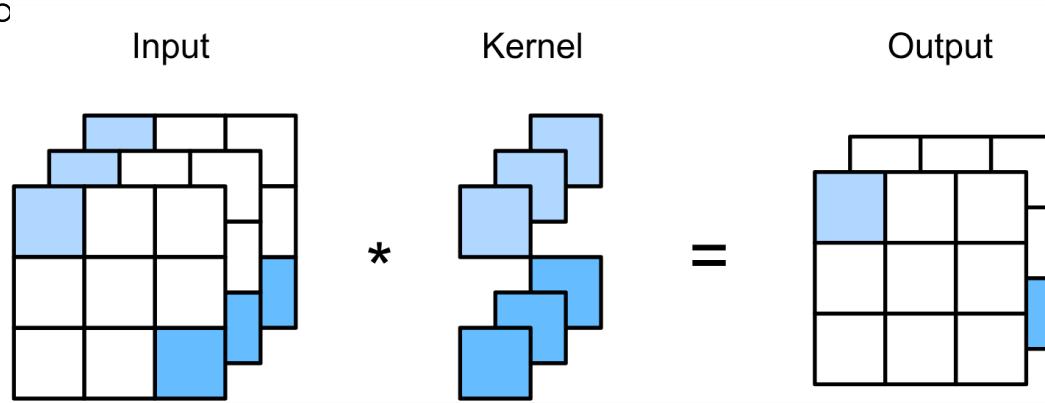


- 输入通道核识别并组合输入中的模式



1 x 1 卷积层

$k_h = k_w = 1$ 是一个受欢迎的选择。它不识别空间模式，只是融合通道。



相当于输入形状为 $n_h n_w \times c_i$, 权重为 $c_o \times c_i$ 的全连接层



二维卷积层

- 输入 $\mathbf{X} : c_i \times n_h \times n_w$
- 核 $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
- 偏差 $\mathbf{B} : c_o \times c_i$
- 输出 $\mathbf{Y} : c_o \times m_h \times m_w$
- 计算复杂度 (浮点计算数 FLOP) $O(c_i c_o k_h k_w m_h m_w)$

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + \mathbf{B}$$

$$c_i = c_o = 100$$

$$k_h = h_w = 5$$

$$m_h = m_w = 64$$



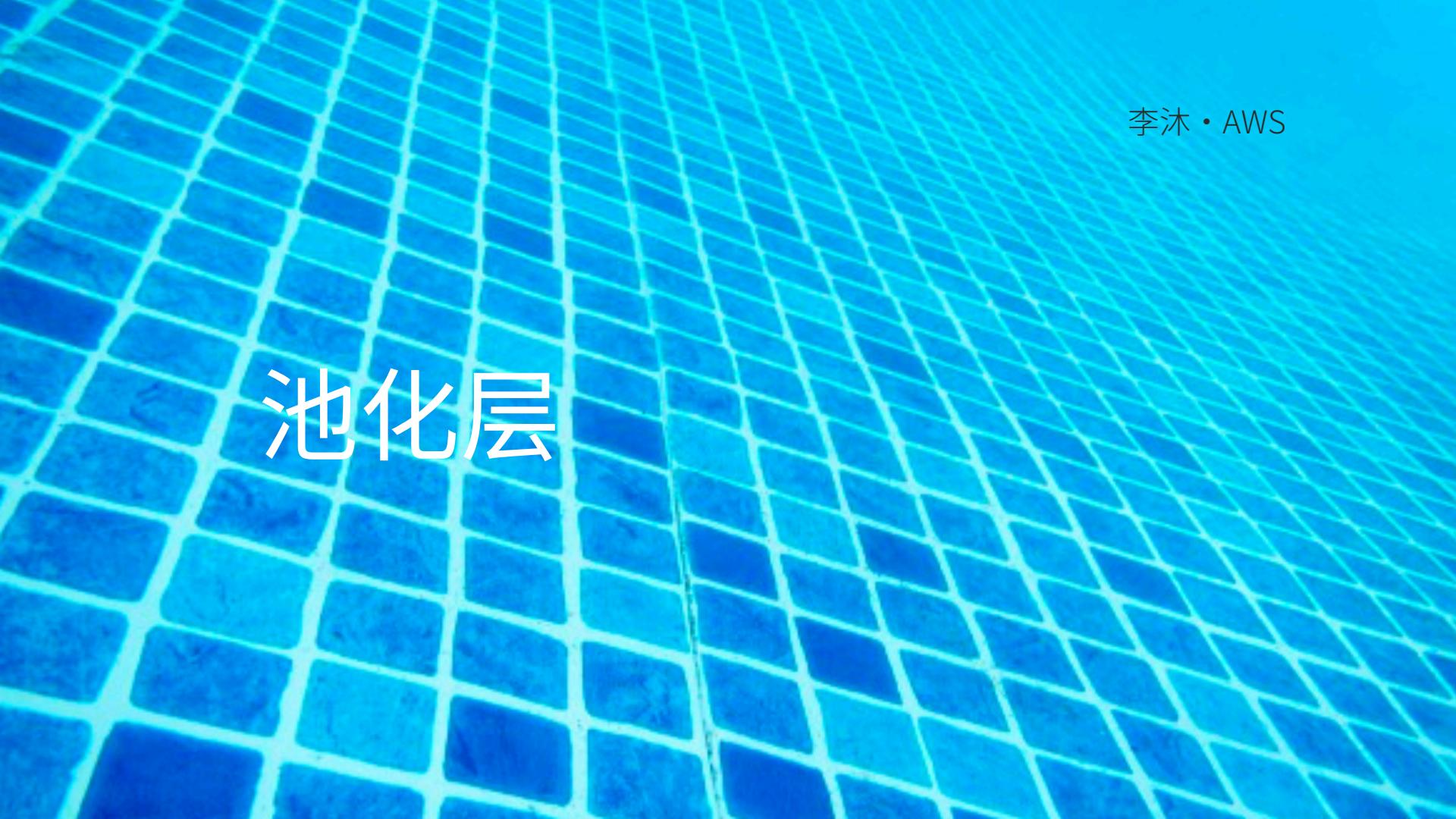
1GFLOP

- 10 层, 1M 样本, 10 PFlops
(CPU: 0.15 TF = 18h, GPU: 12 TF = 14min)



总结

- 输出通道数是卷积层的超参数
- 每个输入通道有独立的二维卷积核，所有通道结果相加得到一个输出通道结果
- 每个输出通道有独立的三维卷积核



李沐 · AWS

池化层



池化层

- 卷积对位置敏感
 - 检测垂直边缘

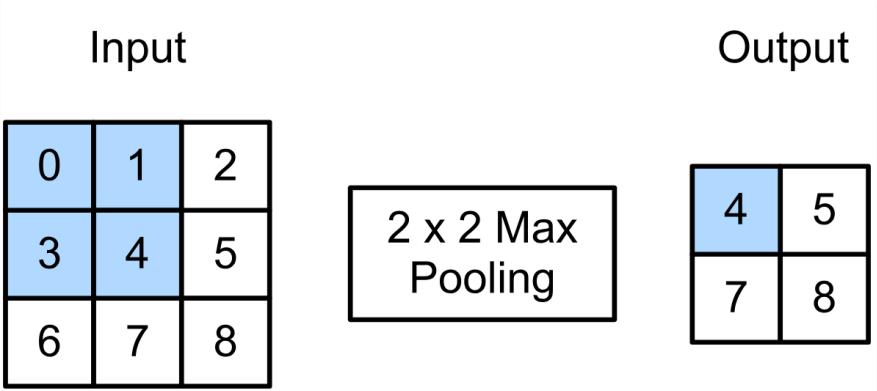
1像素移位 导致 0 输出

$$\begin{array}{c} X \\ \times \end{array} \quad \begin{bmatrix} [1. & 1. & 0. & 0. & 0. \\ 1. & 1. & 0. & 0. & 0. \\ 1. & 1. & 0. & 0. & 0. \\ 1. & 1. & 0. & 0. & 0. \end{bmatrix} \quad Y \quad \begin{bmatrix} [0. & 1. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 1. & 0. & 0. \end{bmatrix}$$

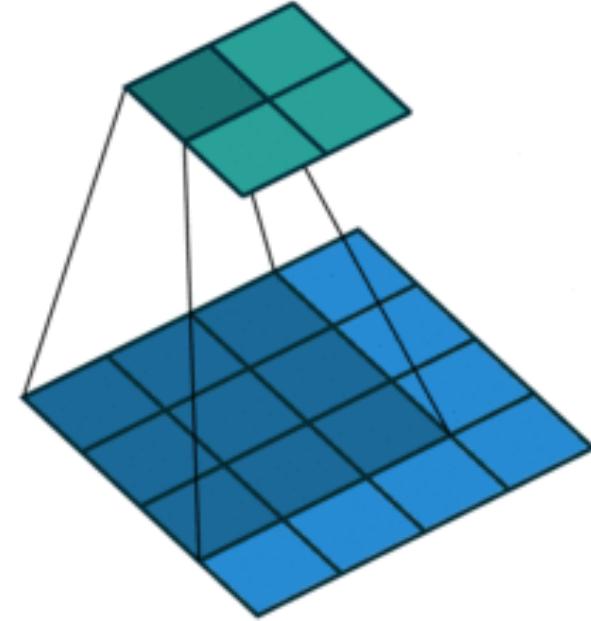
- 需要一定程度的平移不变性
 - 照明，物体位置，比例，外观等等因图像而异

二维最大池化

- 返回滑动窗口中的最大值



$$\max(0,1,3,4) = 4$$





二维最大池化

- 返回滑动窗口中的最大值

垂直边缘检测

```
[[1. 1. 0. 0. 0.  
 [1. 1. 0. 0. 0.  
 [1. 1. 0. 0. 0.  
 [1. 1. 0. 0. 0.
```

卷积输出

```
[[ 0. 1.  
 [ 0. 1.  
 [ 0. 1.  
 [ 0. 1.
```

2 x 2 最大池化

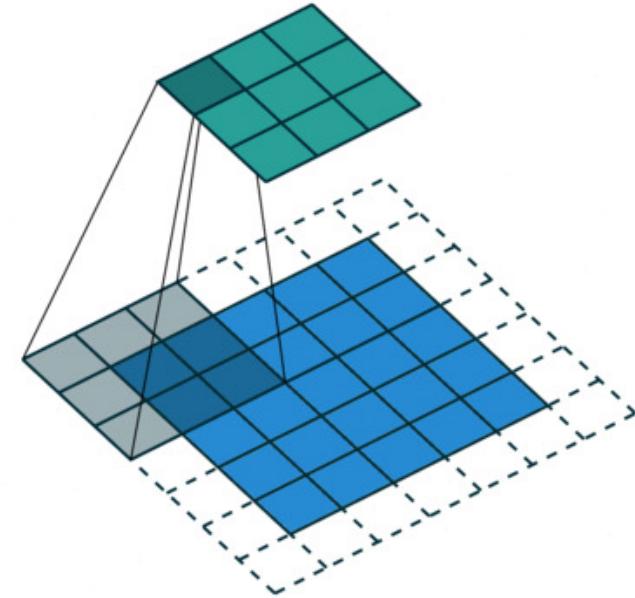
```
[[ 1. 1.  
 [ 1. 1.  
 [ 1. 1.  
 [ 1. 1.
```

可容1像素移位



填充，步幅和多个通道

- 池化层与卷积层类似，都具有填充和步幅
- 没有可学习的参数
- 在每个输入通道应用池化层以获得相应的输出通道
- 输出通道数 = 输入通道数

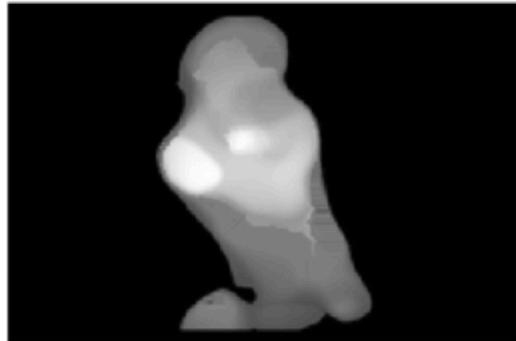




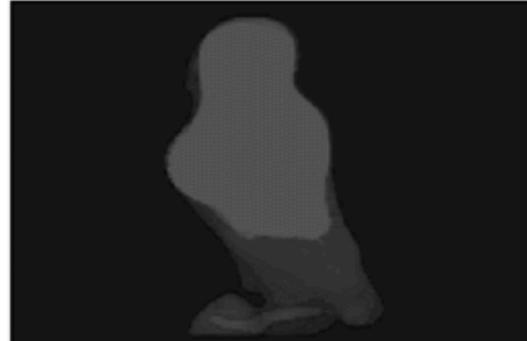
平均池化层

- 最大池化层：每个窗口中最强的模式信号
- 平均池化层：将最大池化层中的“最大”操作替换为“平均”

最大池化层



平均池化层



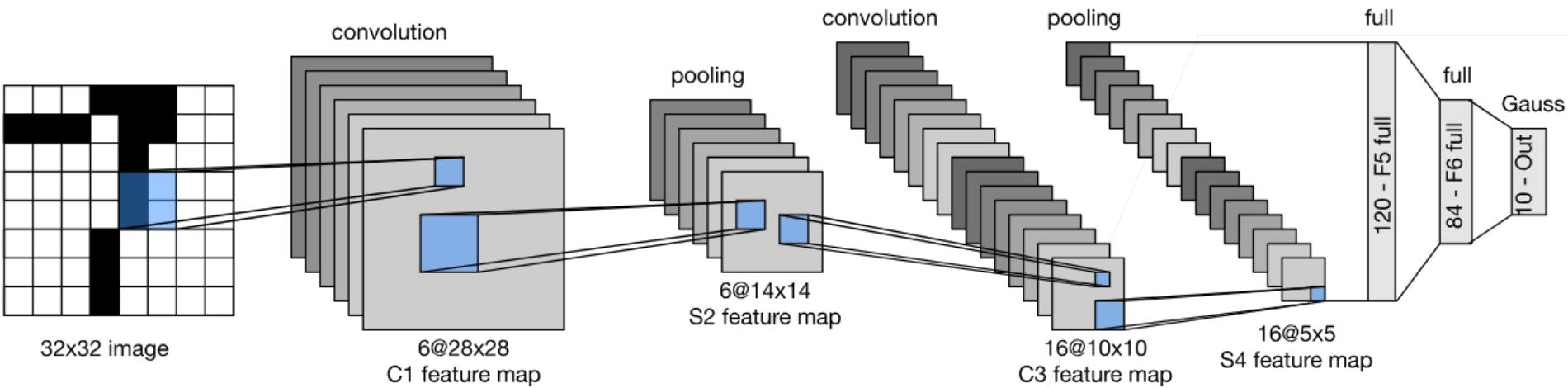


总结

- 池化层返回窗口中最大或平均值
- 缓解卷积层会位置的敏感性
- 同样有窗口大小、填充、和步幅作为超参数

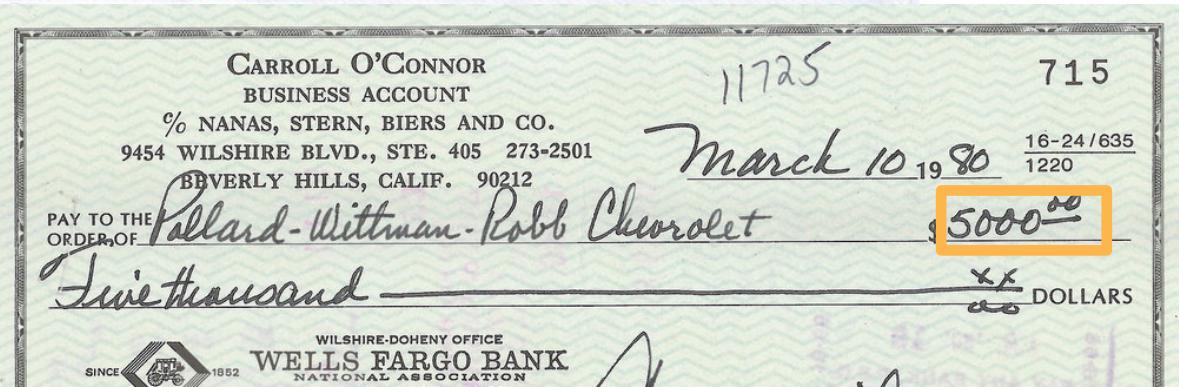
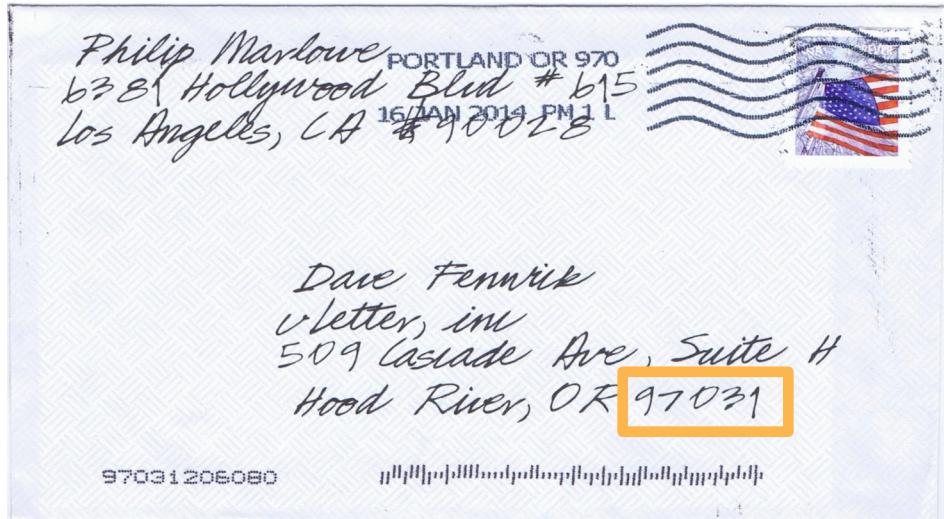


LeNet





手写的数字识别



MNIST

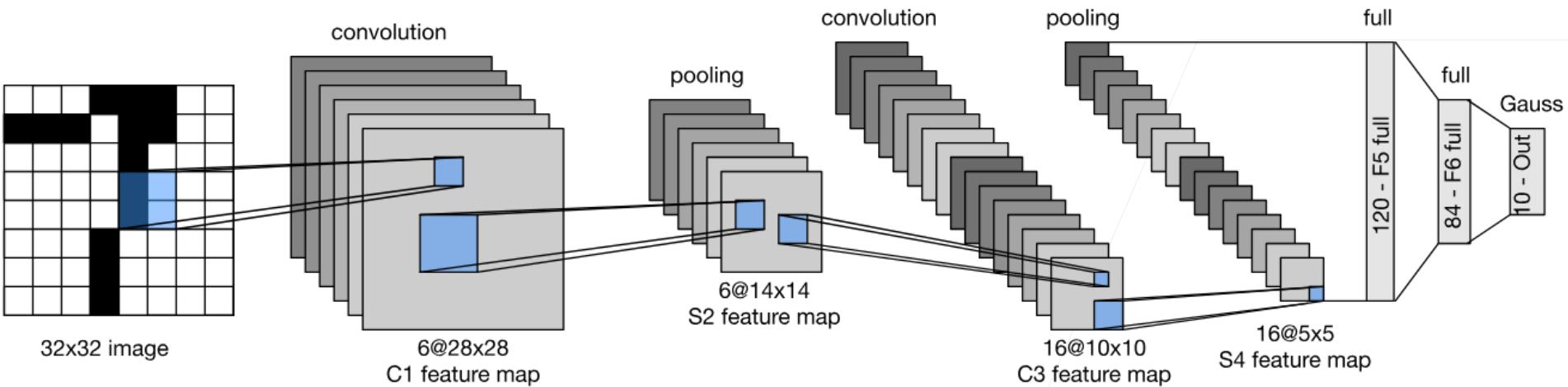
- 50,000 个训练数据
- 10,000 个测试数据
- 图像大小 28×28
- 10类





Y. LeCun, L.
Bottou, Y. Bengio,
P. Haffner, 1998
Gradient-based
learning applied to
document
recognition

95



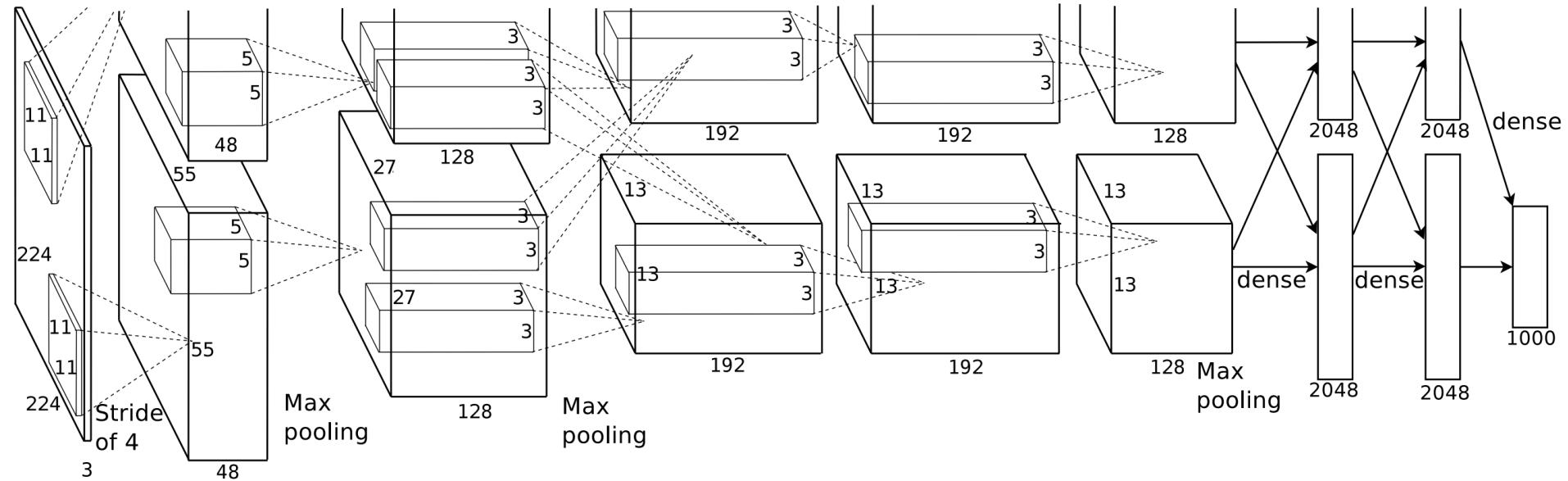


总结

- LeNet是早期成功的神经网络
- 先使用卷积层来学习图片空间信息
- 然后使用全连接层来转换到类别空间



AlexNet





2001

Learning with Kernels

Support Vector Machines, Regularization,
Optimization, and Beyond

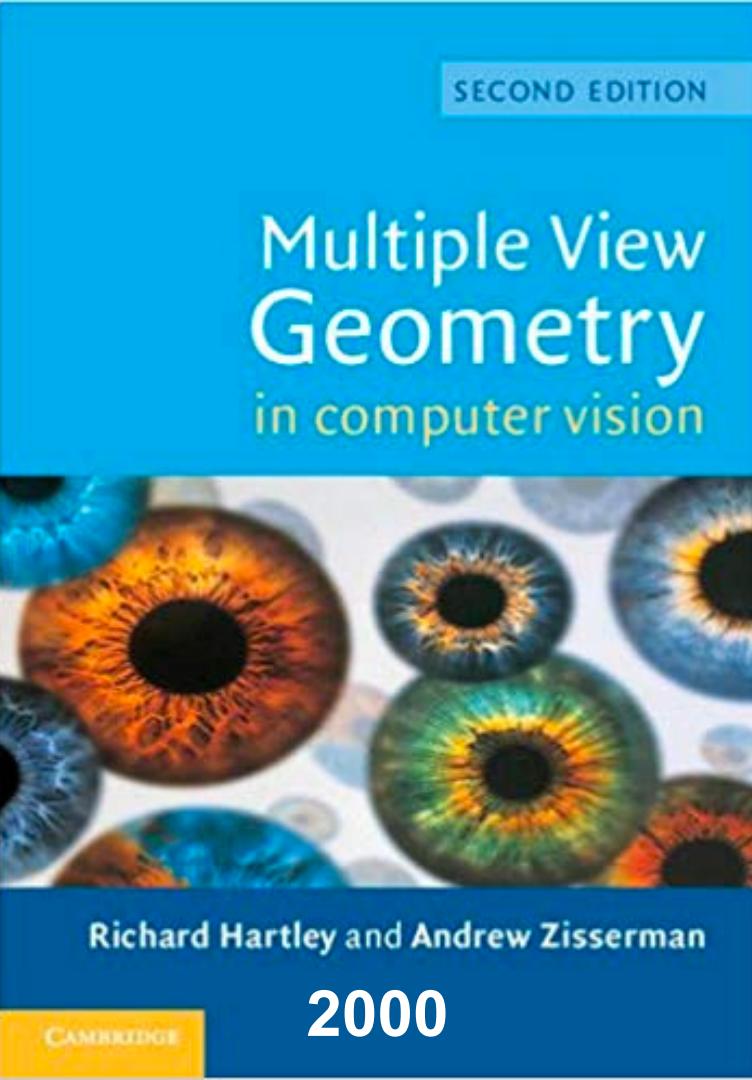
Bernhard Schölkopf and Alexander J. Smola



机器学习

In the 1990s, a new type of learning algorithm was developed, based on results from statistical learning theory: the Support Vector Machine (SVM). This gave rise to a new class of theoretically elegant learning machines that use a central concept of SVMs – kernels – for a number of

- 特征提取
- 选择核函数来计算相似性
- 凸优化问题
- 漂亮的定理



几何学

- 抽取特征
- 描述几何（例如多相机）
- （非）凸优化
- 漂亮定理
- 如果假设满足了，效果很好



特征工程



- 特征工程是关键
- 特征描述子：SIFT, SURF
- 视觉词袋（聚类）
- 最后用 SVM



Hardware

	1970	1980	1990	2000	2010	2020	
Data (samples)	10^2 (e.g. iris)	10^3	$10x$ 10^4 OCR 神经网络	$10x$ 10^4 OCR 神经网络	$100x$ 10^{7-8} web	$100x$ 10^{10} advertising 神经网络	$1,000x$ 10^{12} social nets
RAM	1kB	100kB	10MB	100MB	1GB	100GB	
CPU	100kF (8080)	1MF (80186)	10MF (80486)	1GF (Intel Core)	100GF NVIDIA	$>1PF$ (8xP3 Volta) $10,000x$	



ImageNet (2010)



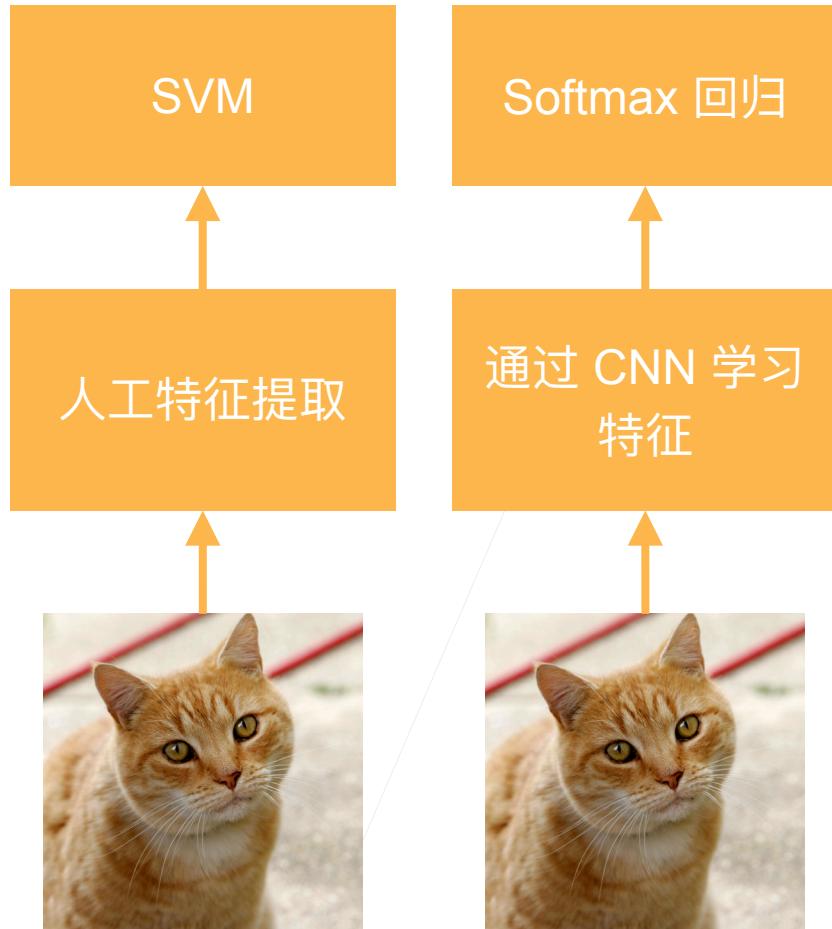
2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6

图片	自然物体的彩色图片	手写数字的黑白图片
大小	469 x 387	28 x 28
样本数	1.2 M	60 K
类数	1,000	10

AlexNet



- AlexNet 赢了 2012 年 ImageNet 竞赛
- 更深更大的 LeNet
- 主要改进：
 - 丢弃法
 - ReLu
 - MaxPooling
- 计算机视觉方法论的改变

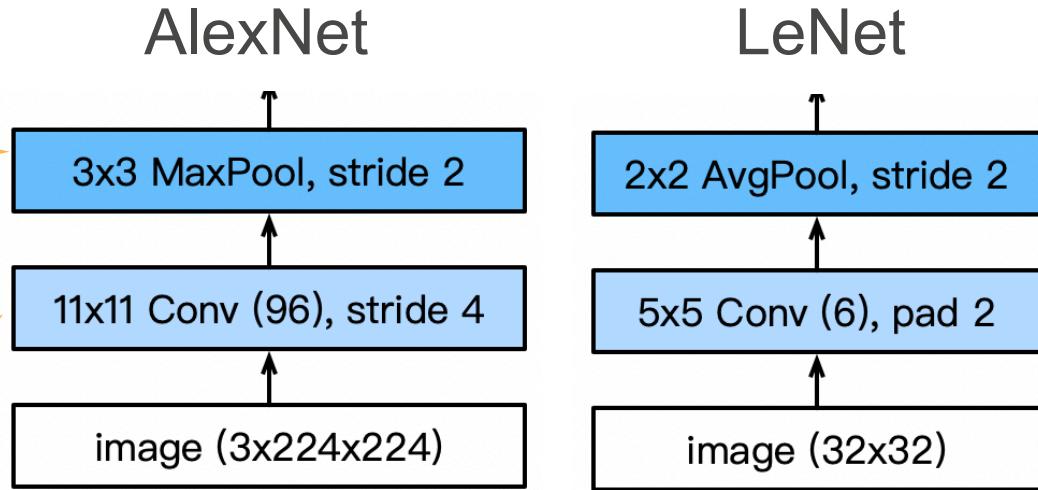


AlexNet 架构

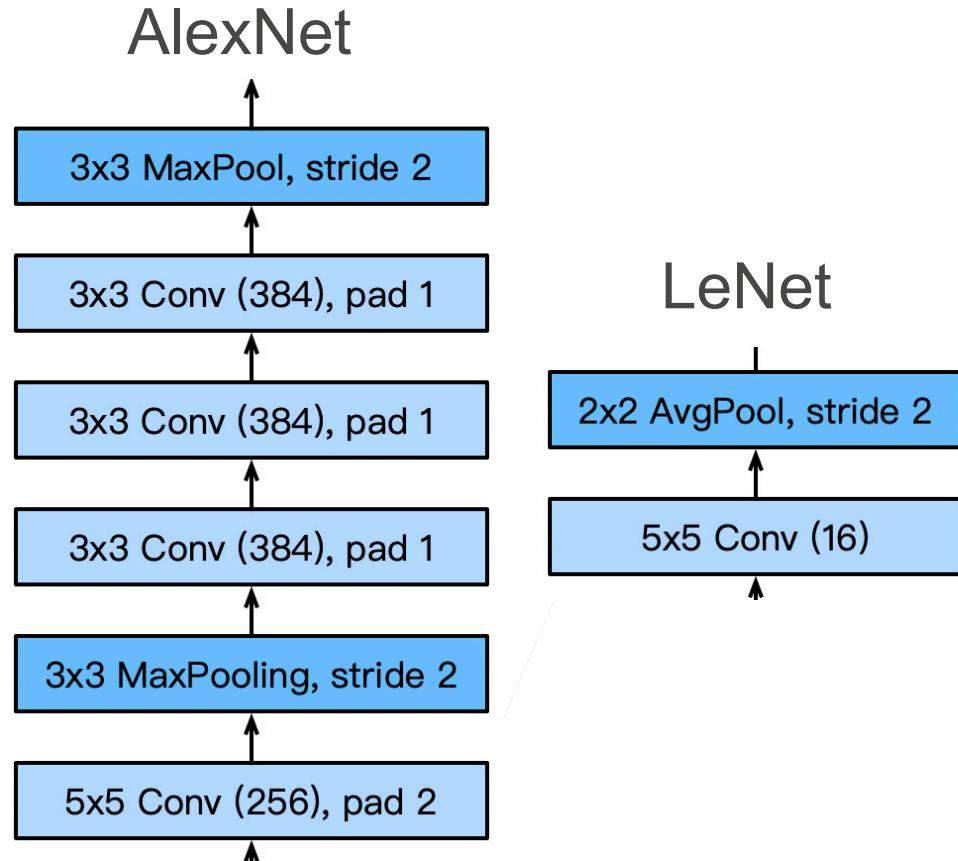


更大的池化窗口，使用最大池化层

更大的核窗口和步长，因为图片更大了



AlexNet 架构

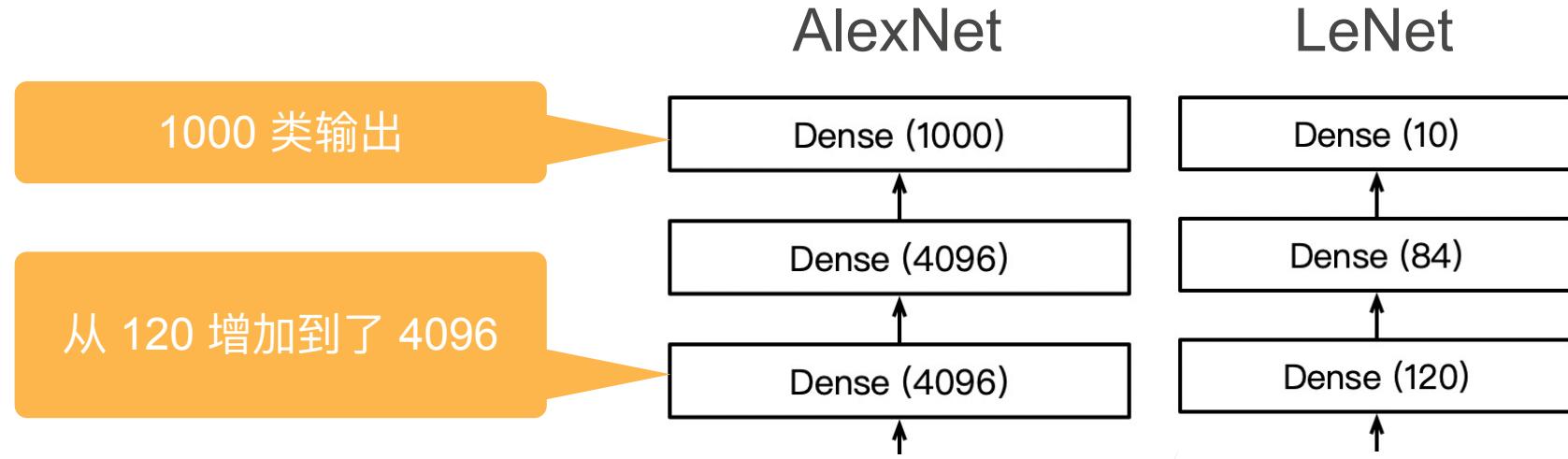


新加了 3 层卷积层

更多的输出通道



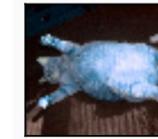
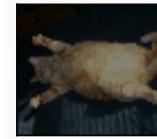
AlexNet 架构



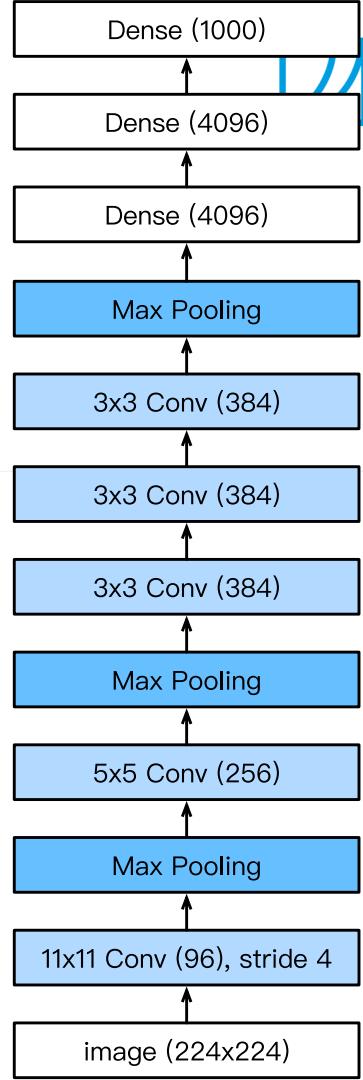


更多细节

- 激活函数从 sigmoid 变到了 ReLu（减缓梯度消失）
- 隐藏全连接层后加入了丢弃层
- 数据增强



复杂度



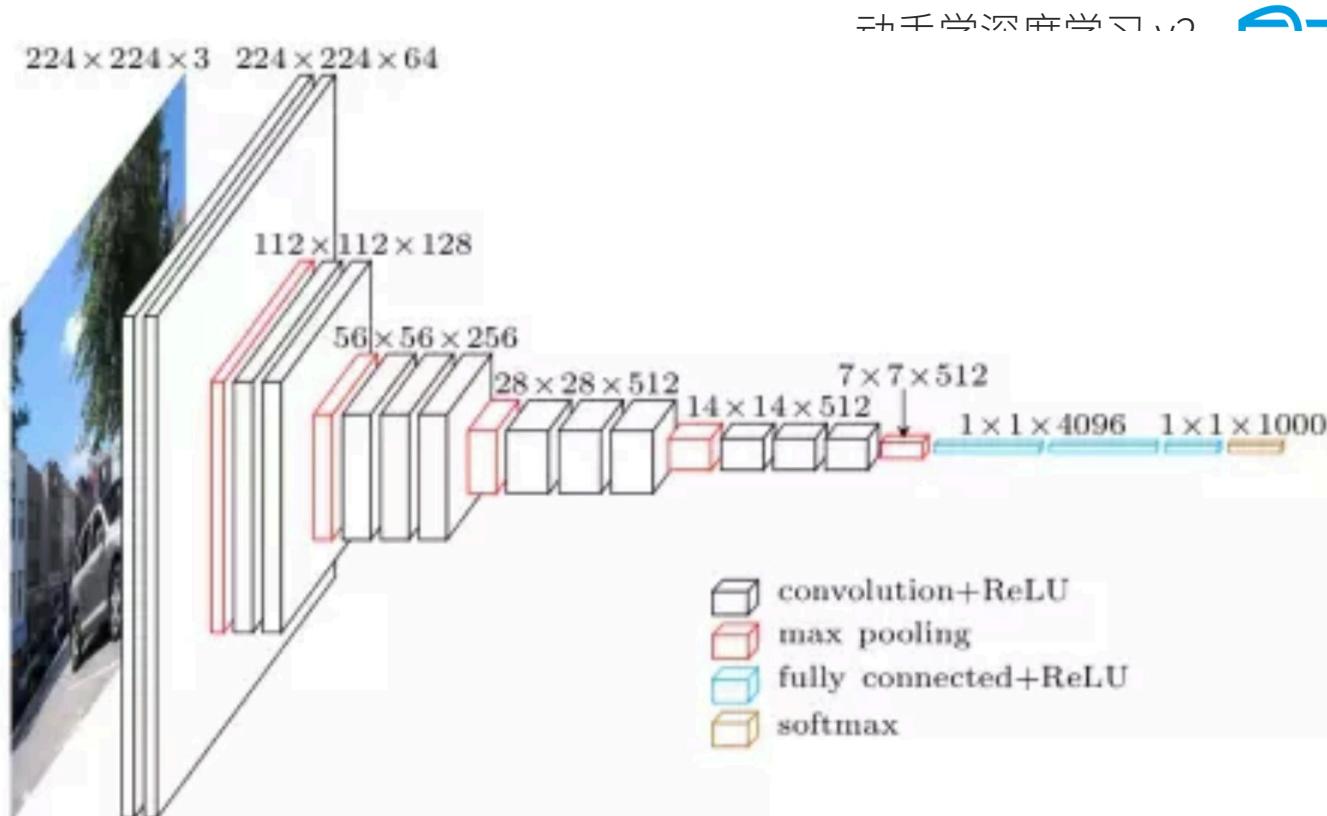
	参数个数		FLOP	
	AlexNet	LeNet	AlexNet	LeNet
Conv1	35K	150	101M	1.2M
Conv2	614K	2.4K	415M	2.4M
Conv3-5	3M		445M	
Dense1	26M	0.48M	26M	0.48M
Dense2	16M	0.1M	16M	0.1M
Total	46M	0.6M	1G	4M
Increase	11x	1x	250x	1x



总结

- AlexNet 是更大更深的 LeNet， $10\times$ 参数个数， $260\times$ 计算复杂度
- 新进入了丢弃法，ReLU，最大池化层，和数据增强
- AlexNet 赢下了 2012 ImageNet 竞赛后，标志着新一轮神经网络热潮的开始

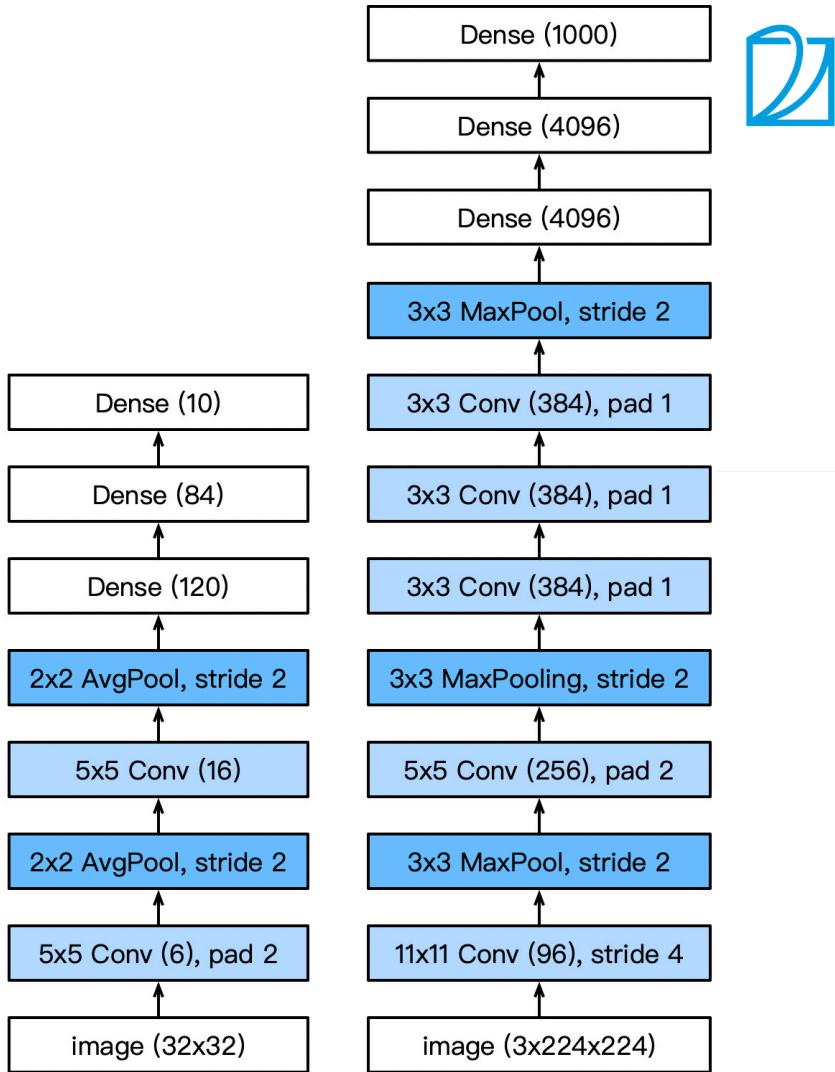
使用块的网络 VGG



VGG



- AlexNet 比 LeNet 更深更大来得到更好的精度
- 能不能更深和更大？
- 选项
 - 更多的全连接层（太贵）
 - 更多的卷积层
 - 将卷积层组合成块

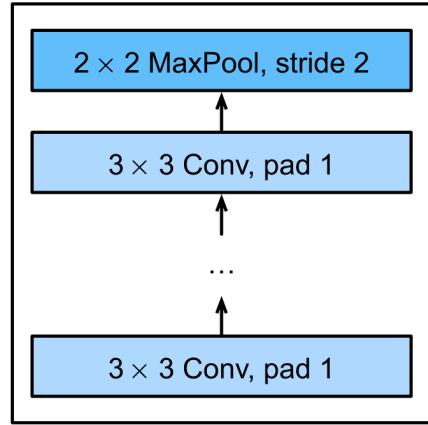




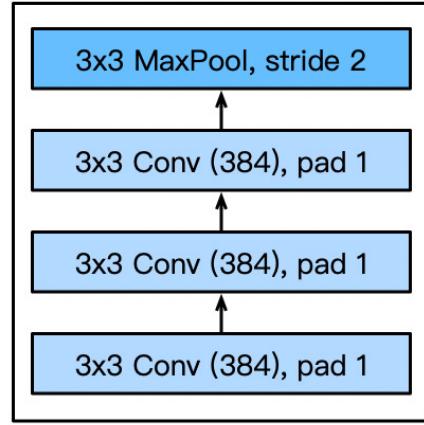
VGG 块

- 深 vs. 宽?
 - 5×5 卷积
 - 3×3 卷积
 - 深但窄效果更好
- VGG 块
 - 3×3 卷积 (填充 1)
(n 层, m 通道)
 - 2×2 最大池化层 (步幅 2)

VGG 块



AlexNet的一部分



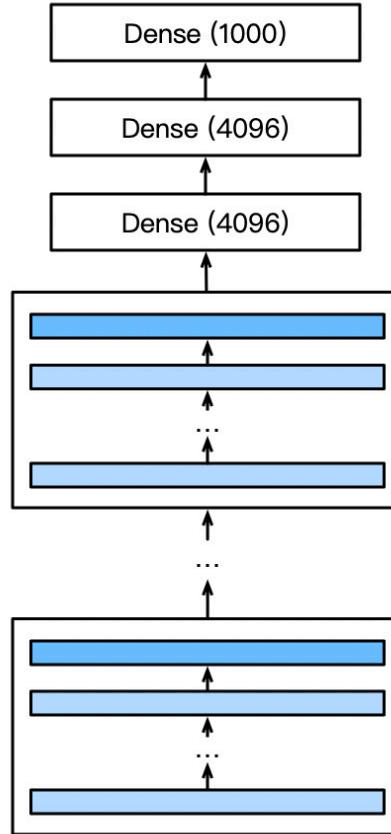


VGG 架构

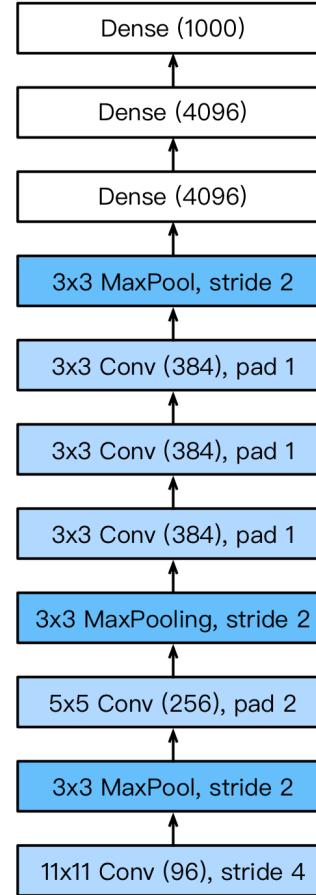
- 多个VGG块后接全连接层
- 不同次数的重复块得到不同的架构
VGG-16, VGG-19,

...

VGG



AlexNet

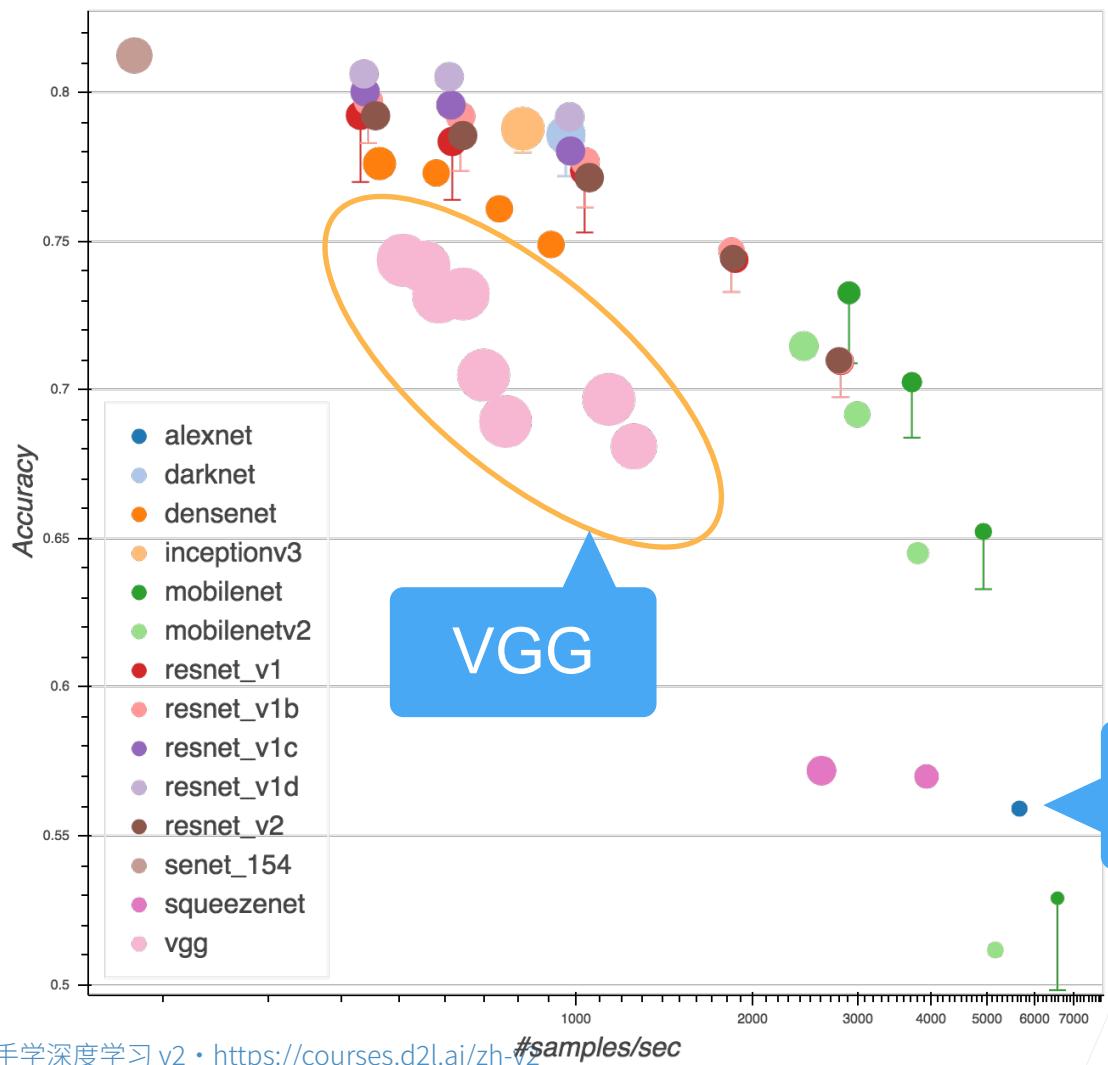




进度

- LeNet (1995)
 - 2 卷积 + 池化层
 - 2 全连接层
- AlexNet
 - 更大更深
 - ReLu, Dropout, 数据增强
- VGG
 - 更大更深的 AlexNet (重复的 VGG 块)

[https://cv.gluon.ai/
model_zoo/
classification.html](https://cv.gluon.ai/model_zoo/classification.html)



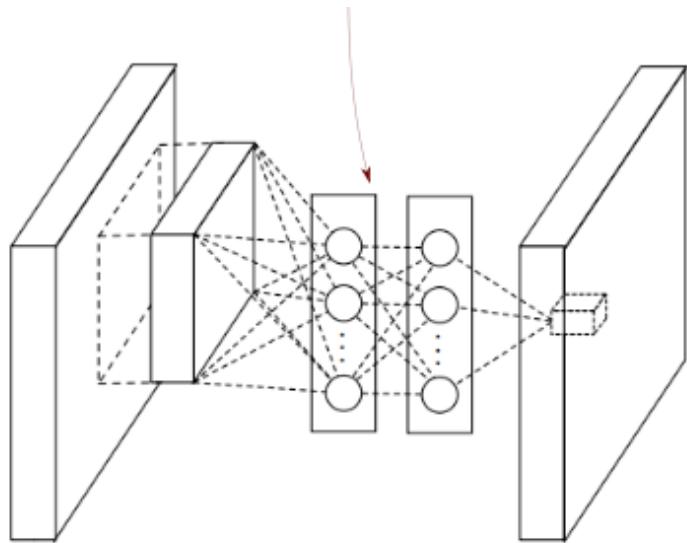


总结

- VGG使用可重复使用的卷积块来构建深度卷积神经网络
- 不同的卷积块个数和超参数可以得到不同复杂度的变种



网络中的网络 (NiN)





全连接层的问题

- 卷积层需要较少的参数
 $c_i \times c_o \times k^2$
- 但卷积层后的第一个全连接层的参数
 - LeNet $16 \times 5 \times 5 \times 120 = 48k$
 - AlexNet $256 \times 5 \times 5 \times 4096 = 26M$
 - VGG $512 \times 7 \times 7 \times 4096 = 102M$

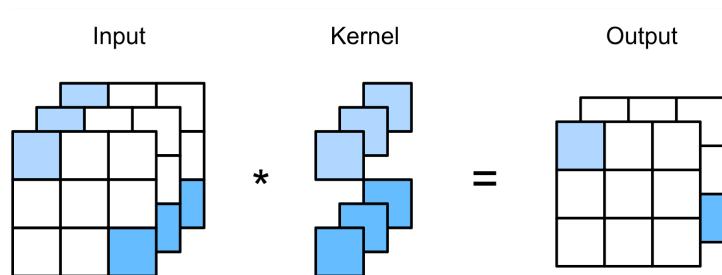
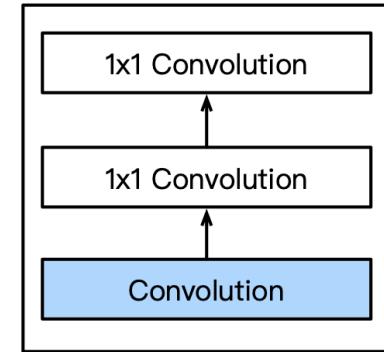
VGG:

```
sequential1 output shape: (1, 64, 112, 112)
sequential2 output shape: (1, 128, 56, 56)
sequential3 output shape: (1, 256, 28, 28)
sequential4 output shape: (1, 512, 14, 14)
sequential5 output shape: (1, 512, 7, 7)
dense0 output shape: (1, 4096)
dropout0 output shape: (1, 4096)
dense1 output shape: (1, 4096)
dropout1 output shape: (1, 4096)
dense2 output shape: (1, 10)
```



NIN 块

- 一个卷积层后跟两个全连接层
 - 步幅 1，无填充，输出形状跟卷积层输出一样
 - 起到全连接层的作用





NiN架构

- 无全连接层
- 交替使用NiN块和步幅为2的最大池化层
 - 逐步减小高宽和增大通道数
- 最后使用全局平均池化层得到输出
 - 其输入通道数是类别数



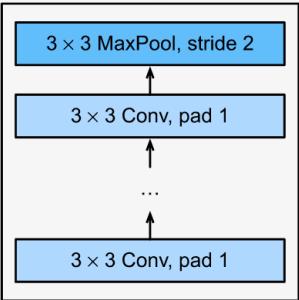
NiN Networks

VGG Net

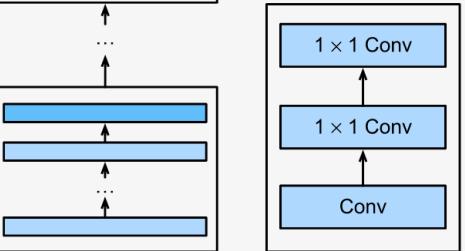
NiN Net

VGG

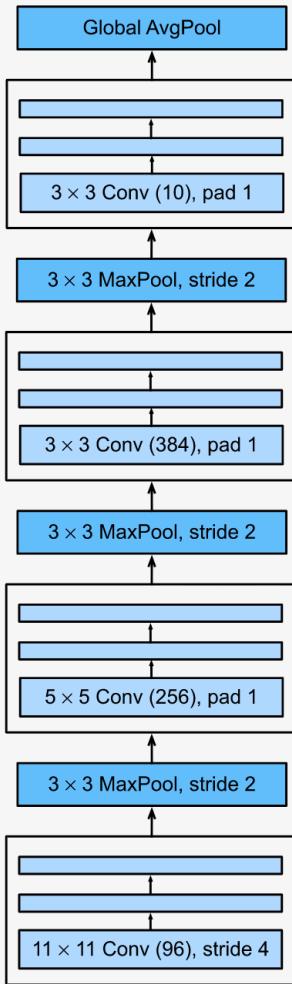
VGG block



NiN block



NiN



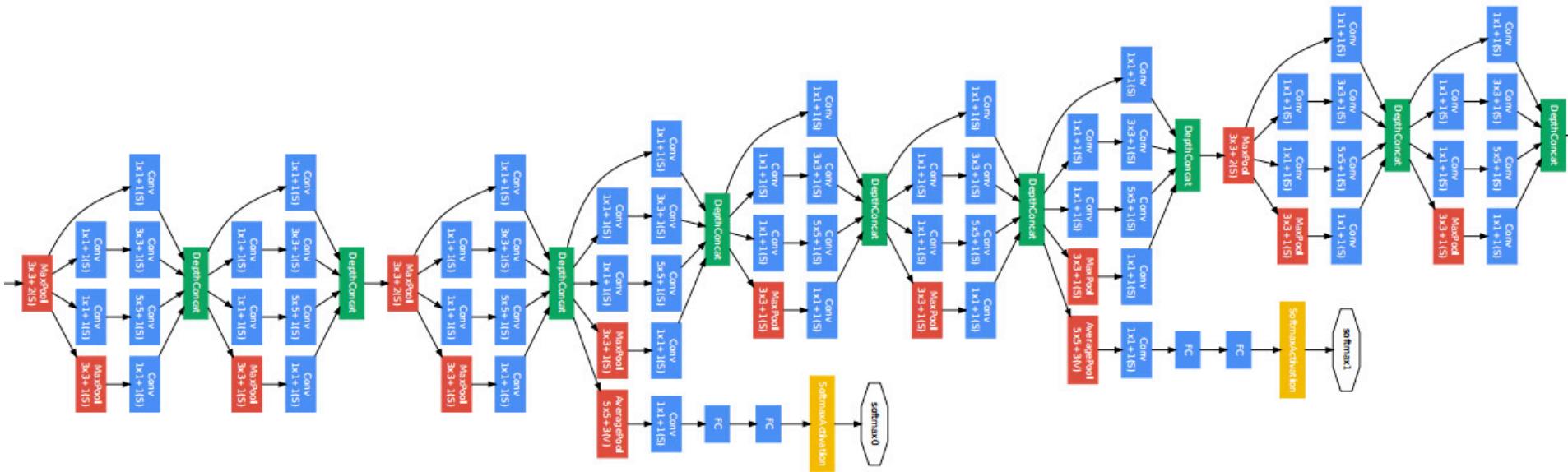


总结

- NiN块使用卷积层加两个 1×1 卷积层，
 - 后者对每个像素增加了非线性性
- NiN使用全局平均池化层来替代VGG和AlexNet中的全连接层
 - 不容易过拟合，更少的参数个数



含并行连结的网络 (GoogLeNet)





最好的卷积层超参数?

1x1

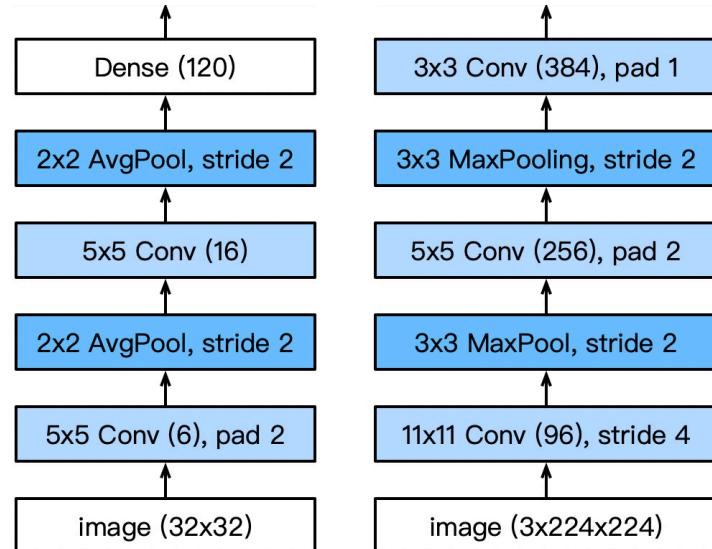
3x3

5x5

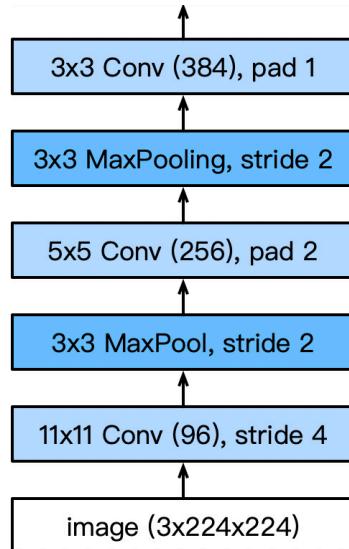
Max pooling

Multiple 1x1

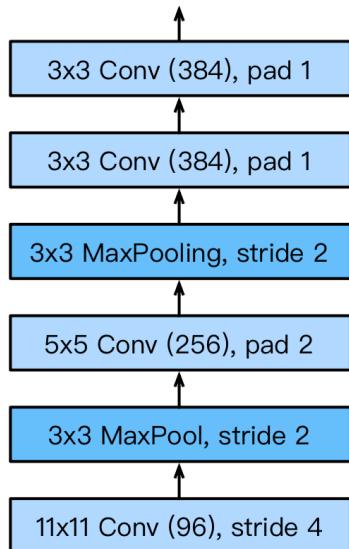
LeNet



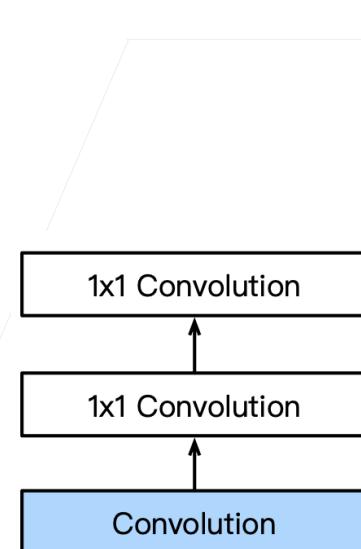
AlexNet



VGG



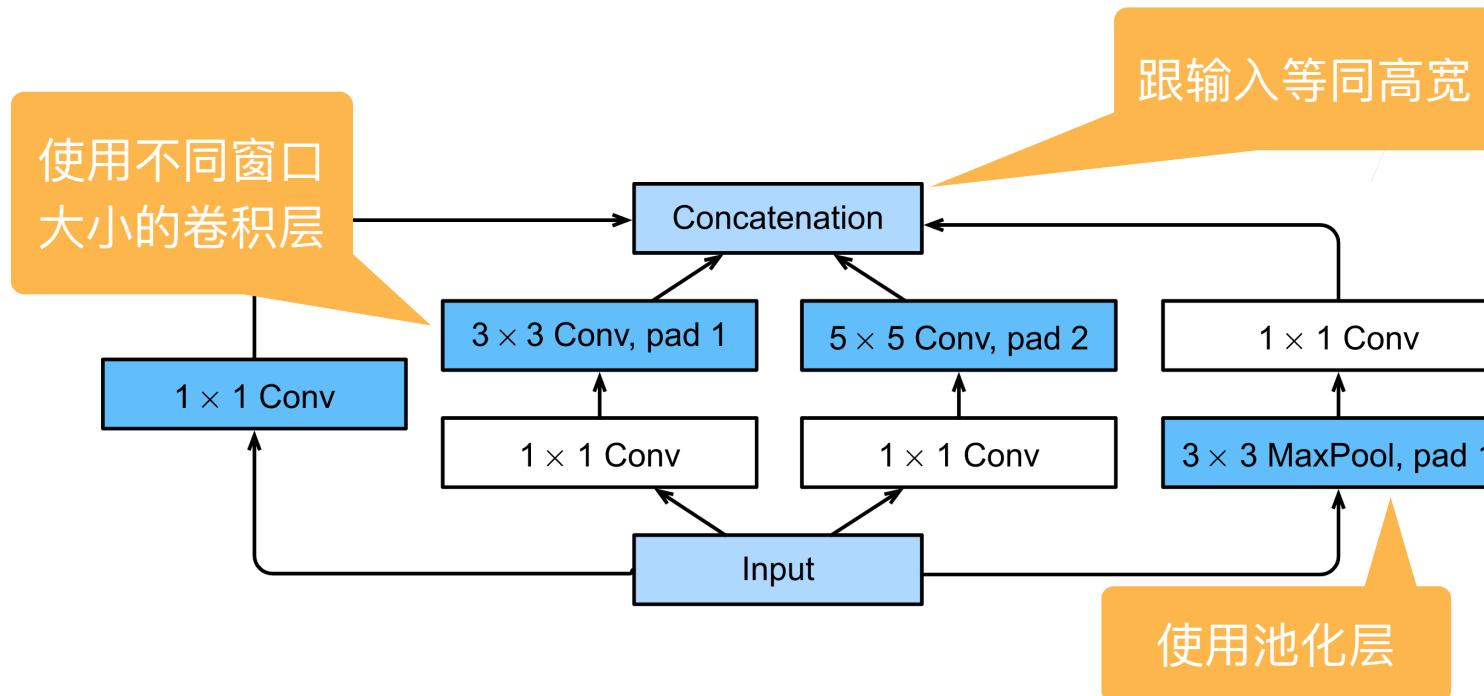
NiN





Inception块：小学生才做选择题，我全要了

4个路径从不同层面抽取信息，然后在输出通道维合并

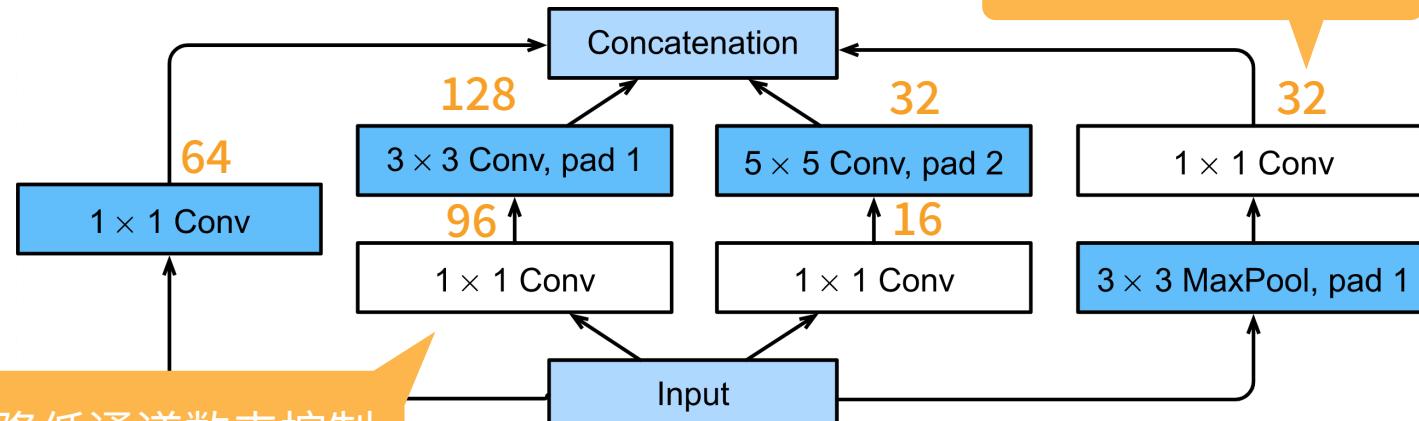




Inception块

第一个Inception块，图示通道数

每条路上通道数可能不同



降低通道数来控制
模型复杂度

$192 \times 28 \times 28$



Inception块

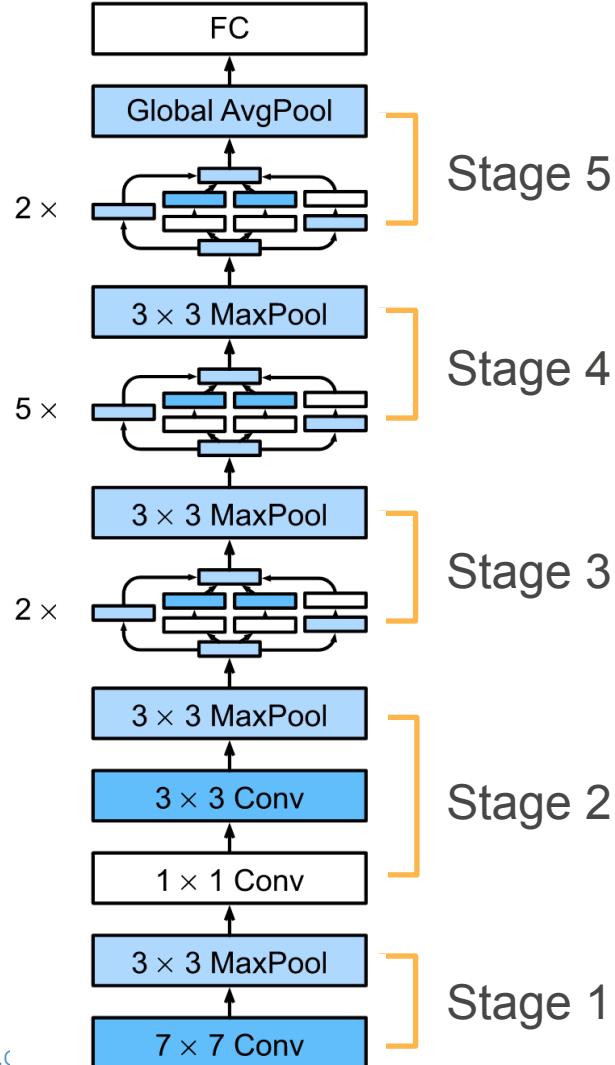
跟单3x3或5x5卷积层比， Inception块有更少的参数个数和计算复杂度

	#parameters	FLOPS
Inception	0.16 M	128 M
3x3 Conv	0.44 M	346 M
5x5 Conv	1.22 M	963 M



GoogLeNet

- 5段，9个 Inception块



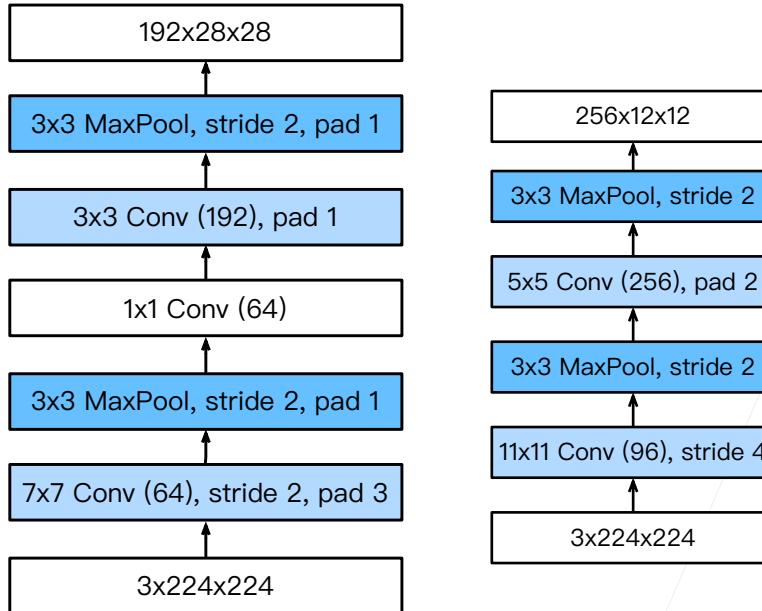


段 1 & 2

GoogLeNet

AlexNet

- 更小的窗口，
更多的通道

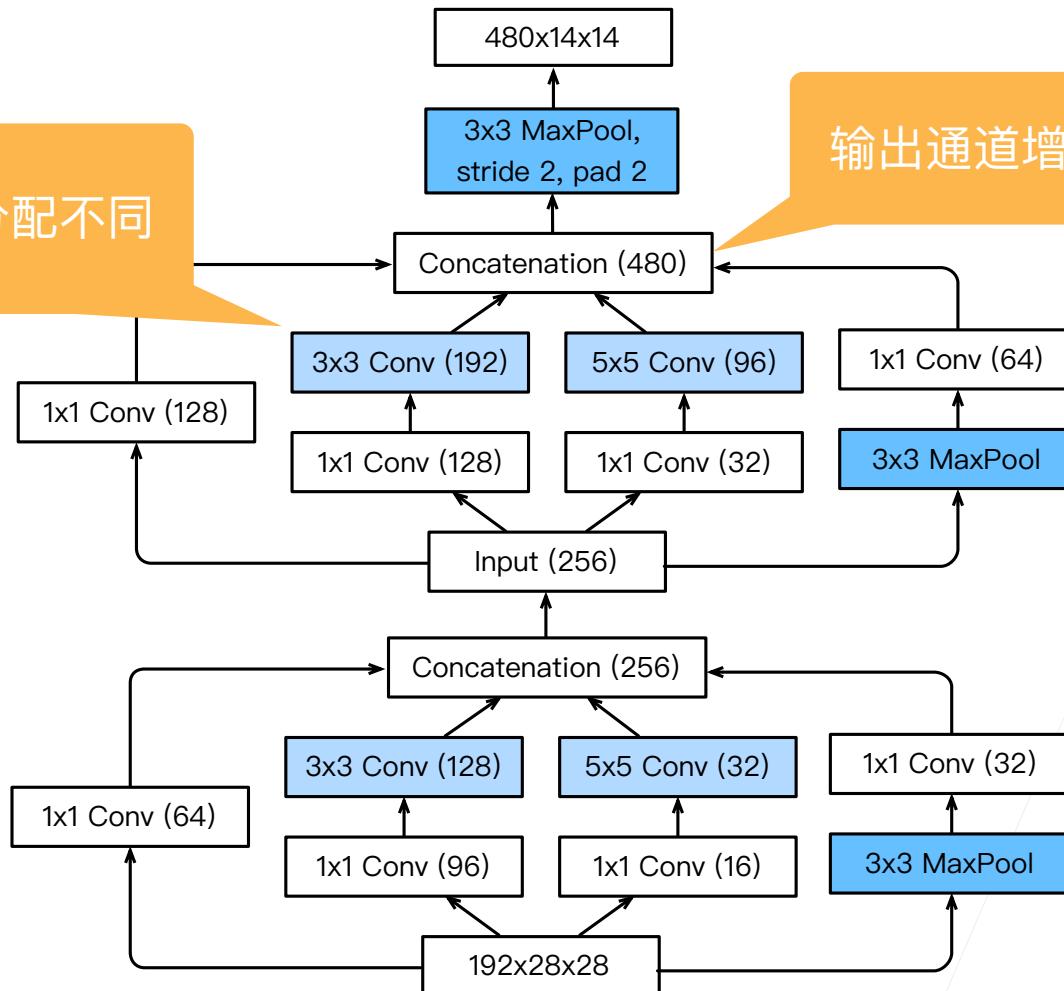




段 3

通道分配不同

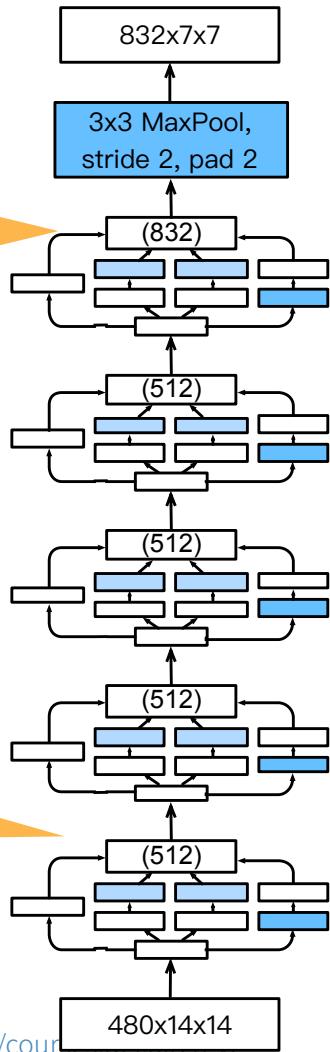
输出通道增加





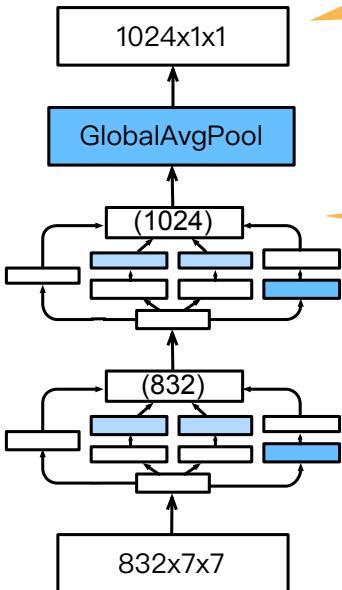
段 4 & 5

增加通道数



1024维特征输出

增加通道数



增加通道数

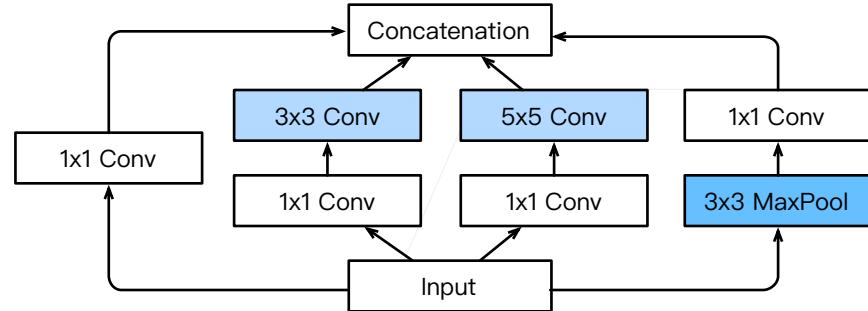
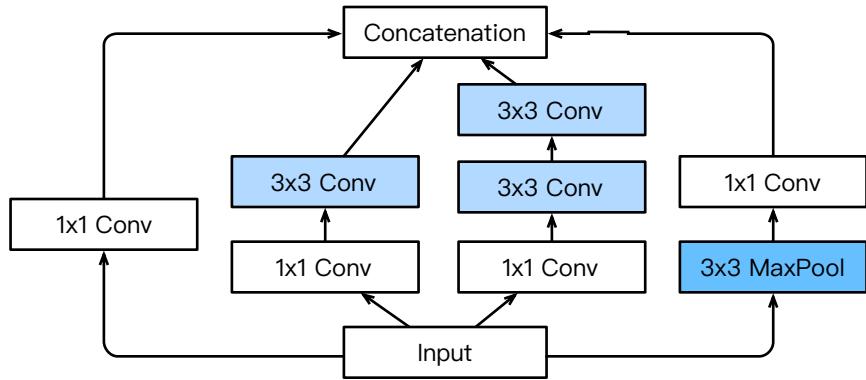


Inception 有各种后续变种

- Inception-BN (v2) - 使用 batch normalization (后面介绍)
- Inception-V3 - 修改了Inception块
 - 替换 5×5 为多个 3×3 卷积层
 - 替换 5×5 为 1×7 和 7×1 卷积层
 - 替换 3×3 为 1×3 和 3×1 卷积层
 - 更深
- Inception-V4 - 使用残差连接 (后面介绍)

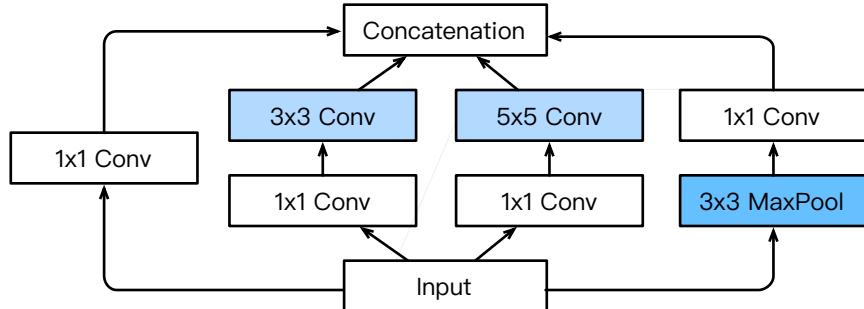
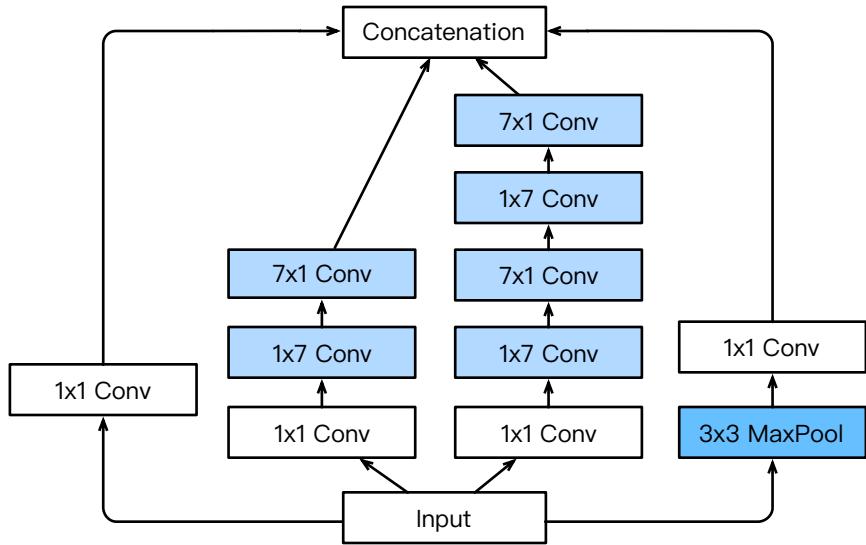


Inception V3 块, 段 3



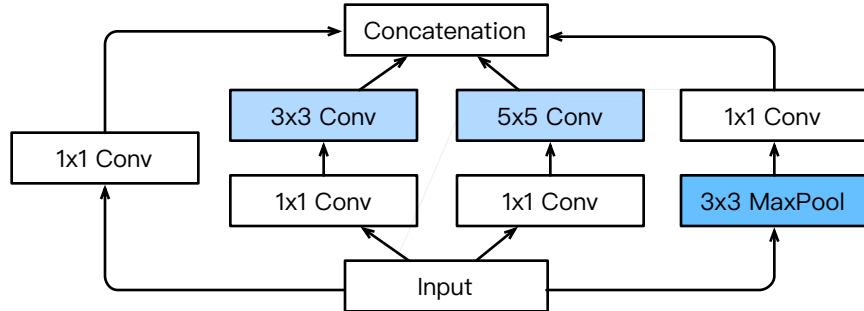
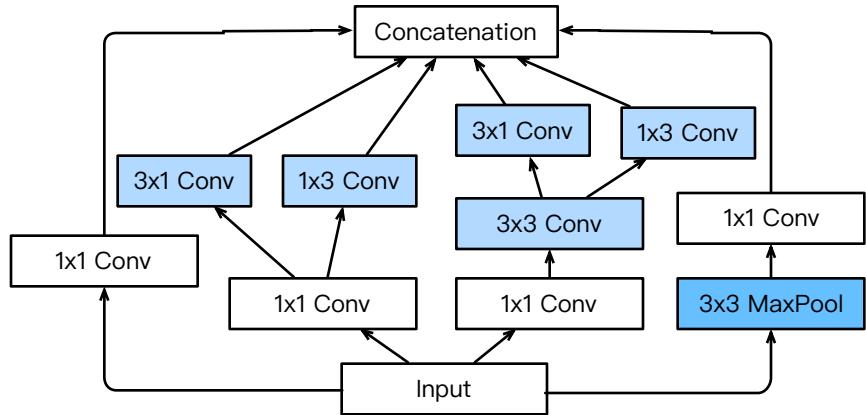


Inception V3 块，段 4



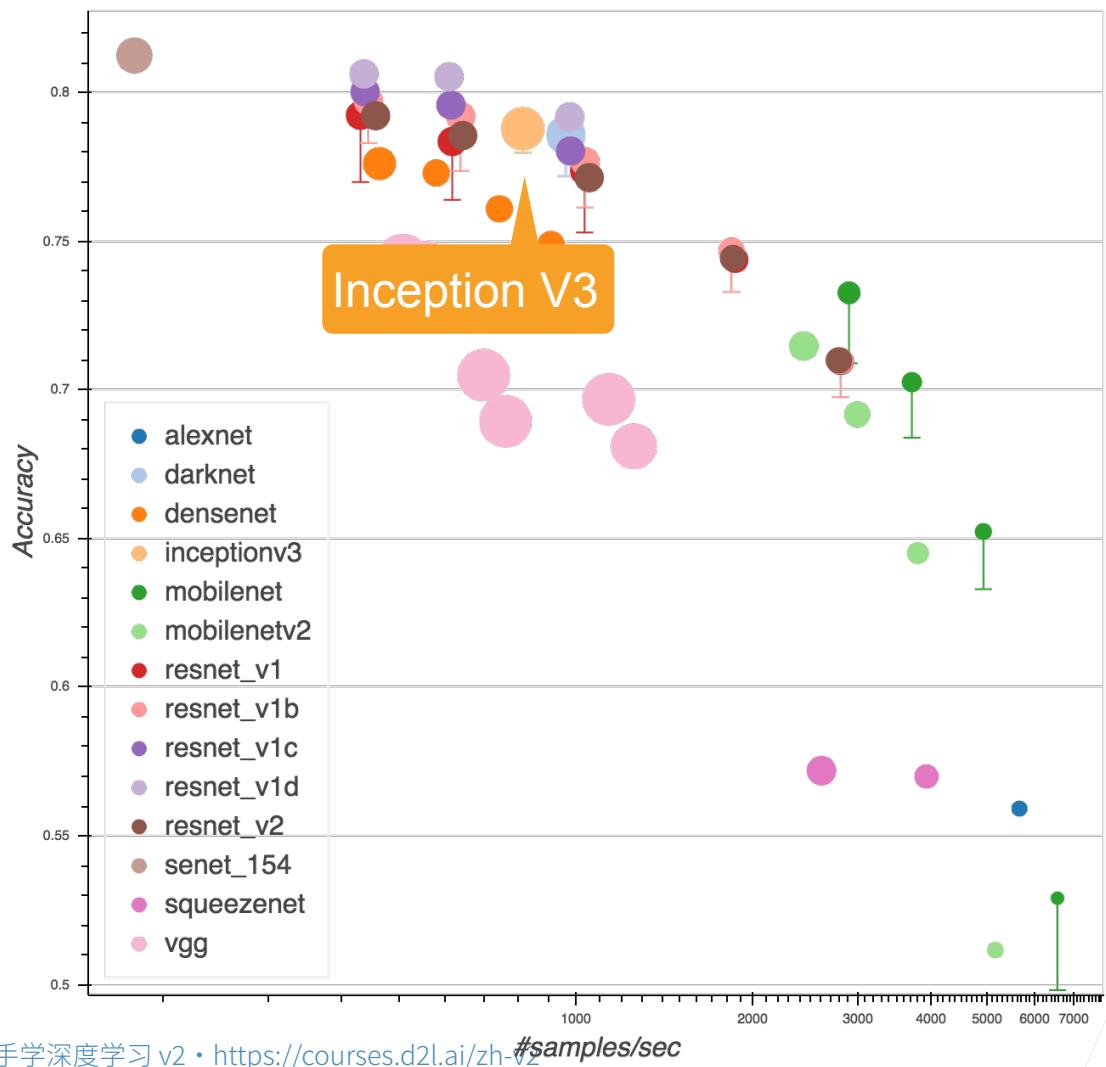


Inception V3 Block, 段 5





GluonCV Model Zoo
[https://cv.gluon.ai/
model_zoo/
classification.html](https://cv.gluon.ai/model_zoo/classification.html)





总结

- Inception块用4条有不同超参数的卷积层和池化层的路来抽取不同的信息
 - 它的一个主要优点是模型参数小，计算复杂度低
- GoogleNet使用了9个Inception块，是第一个达到上百层的网络
 - 后续有一系列改进



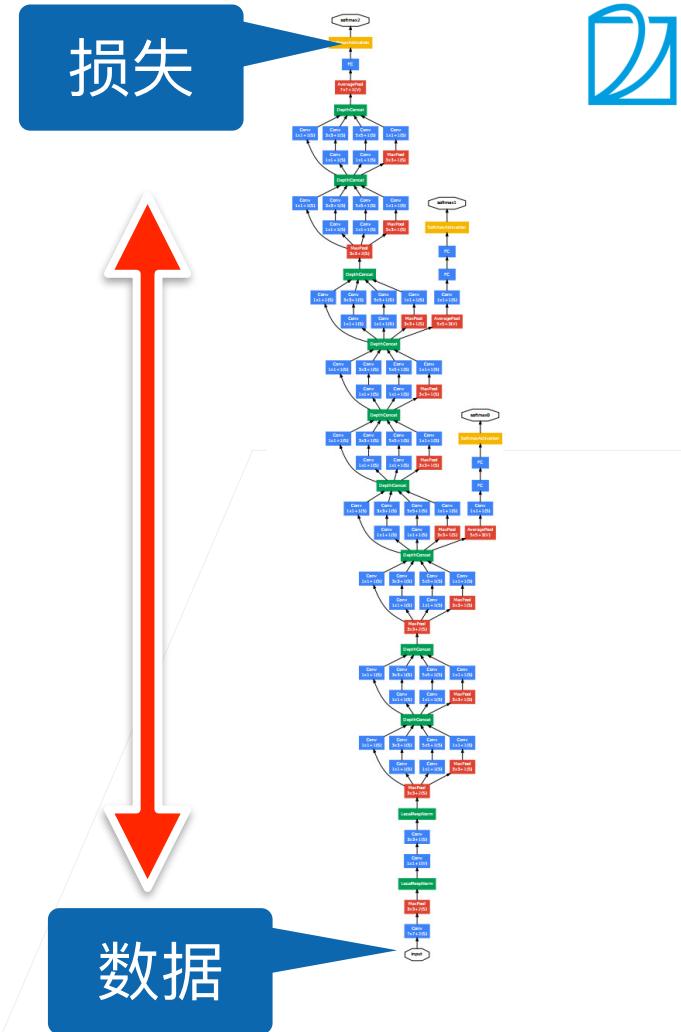
动手学深度学习 v2
李沐 · AWS



批量归一化



- 损失出现在最后，后面的层训练较快
 - 数据在最底部
 - 底部的层训练较慢
 - 底部层一变化，所有都得跟着变
 - 最后的那些层需要重新学习多次
 - 导致收敛变慢
 - 我们可以在学习底部层的时候避免变化顶部层吗？





批量归一化

- 固定小批量里面的均值和方差

$$\mu_B = \frac{1}{|B|} \sum_{i \in B} x_i \text{ and } \sigma_B^2 = \frac{1}{|B|} \sum_{i \in B} (x_i - \mu_B)^2 + \epsilon$$

然后再做额外的调整（可学习的参数）

$$x_{i+1} = \gamma \frac{x_i - \mu_B}{\sigma_B} + \beta$$

均值

方差



批量归一化层

- 可学习的参数为 γ 和 β
- 作用在
 - 全连接层和卷积层输出上，激活函数前
 - 全连接层和卷积层输入上
- 对全连接层，作用在特征维
- 对于卷积层，作用在通道维



批量归一化在做什么？

- 最初论文是想用它来减少内部协变量转移
- 后续有论文指出它可能就是通过在每个小批量里加入噪音来控制模型复杂度

$$x_{i+1} = \gamma \frac{x_i - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

随机偏移

随机缩放

- 因此没必要跟丢弃法混合使用

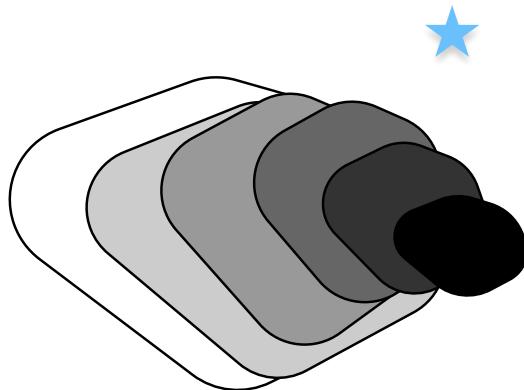


总结

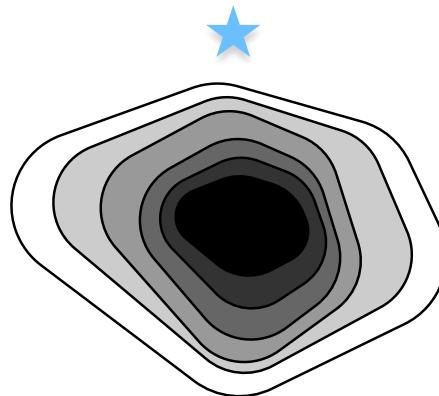
- 批量归一化固定小批量中的均值和方差，然后学习出适合的偏移和缩放
- 可以加速收敛速度，但一般不改变模型精度



残差网络 (ResNet)



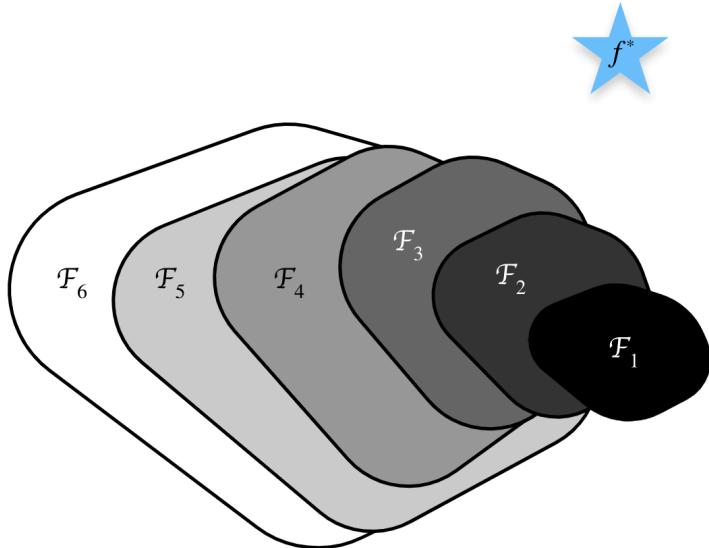
generic function classes



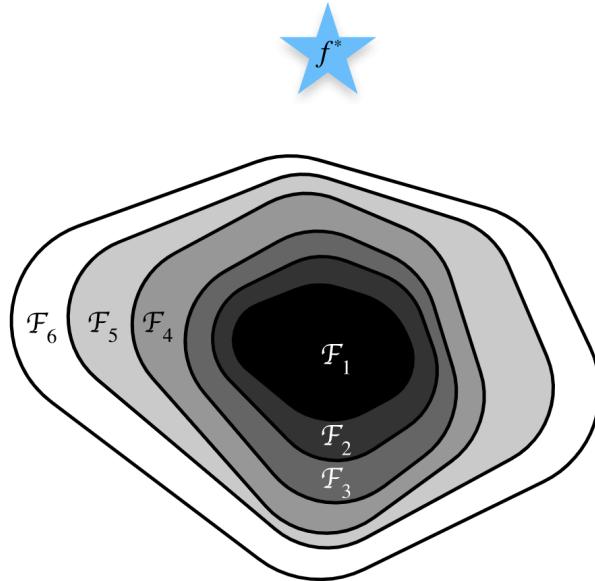
nested function classes



加更多的层总是改进精度吗?



Non-nested function classes



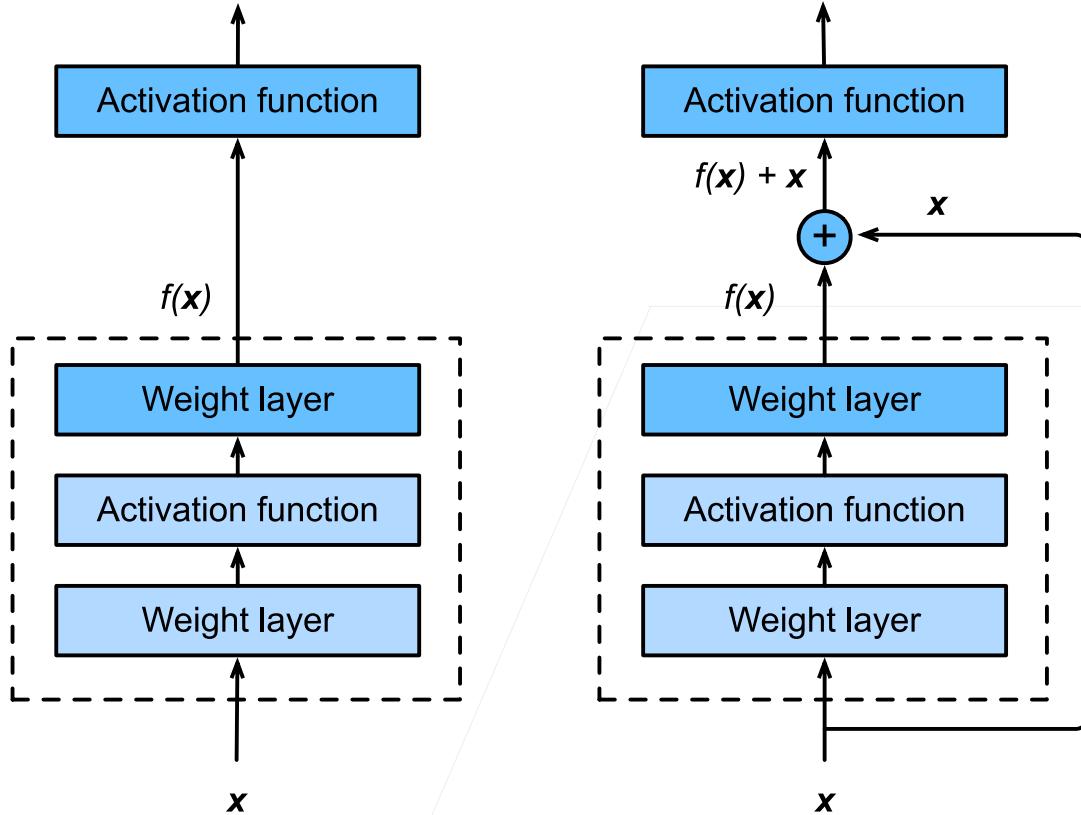
Nested function classes





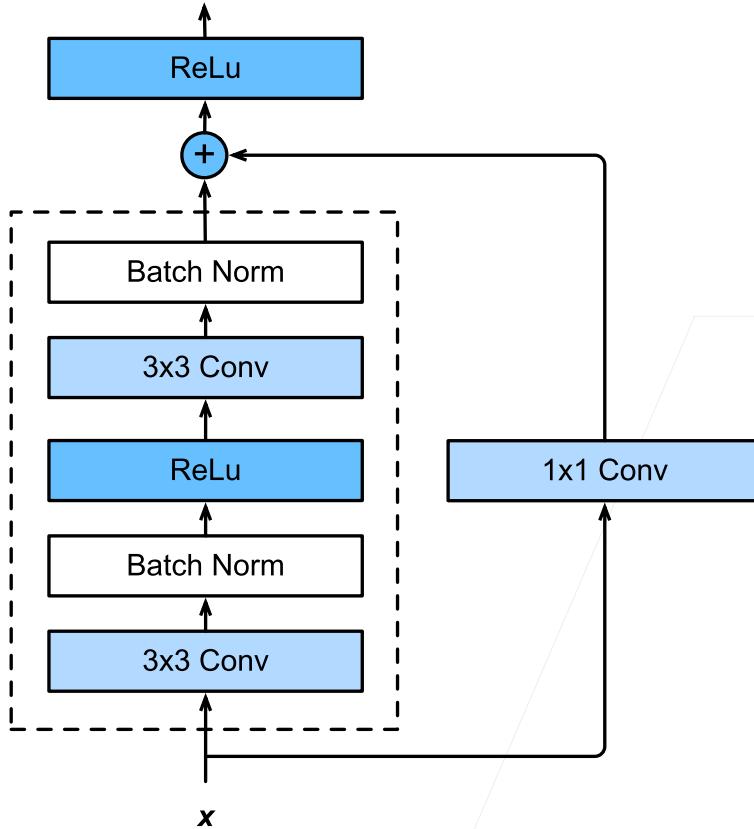
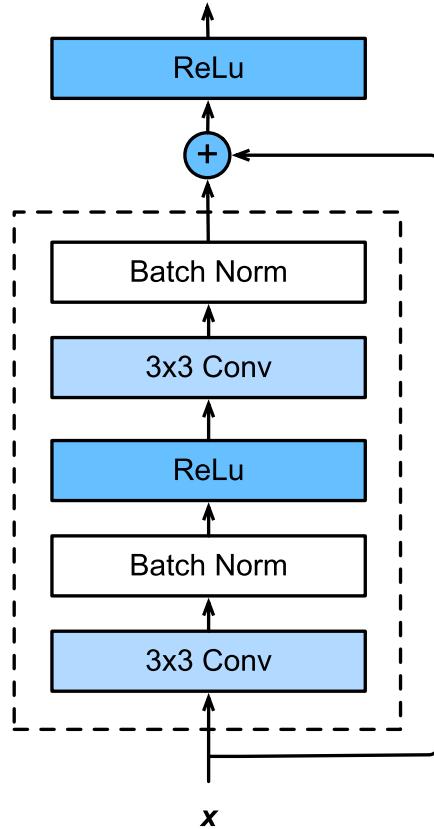
残差块

- 串联一个层改变函数类，我们希望能扩大函数类
- 残差块加入快速通道（右边）来得到 $f(x) = x + g(x)$ 的结构



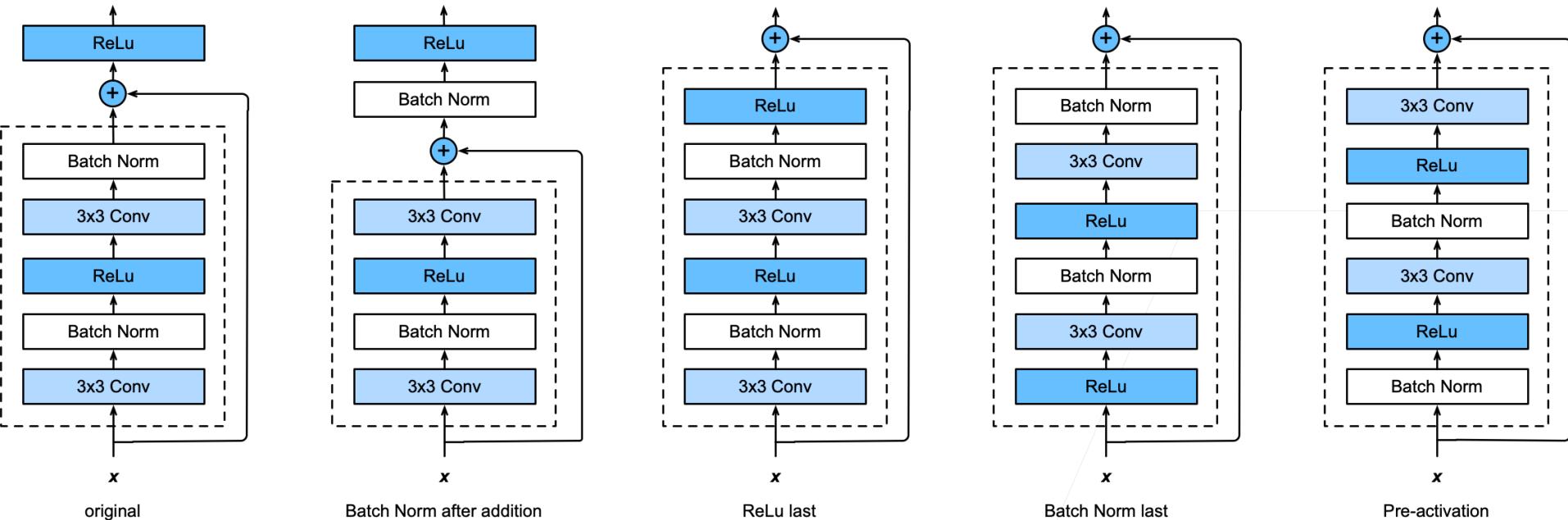


ResNet 块细节





不同的残差块

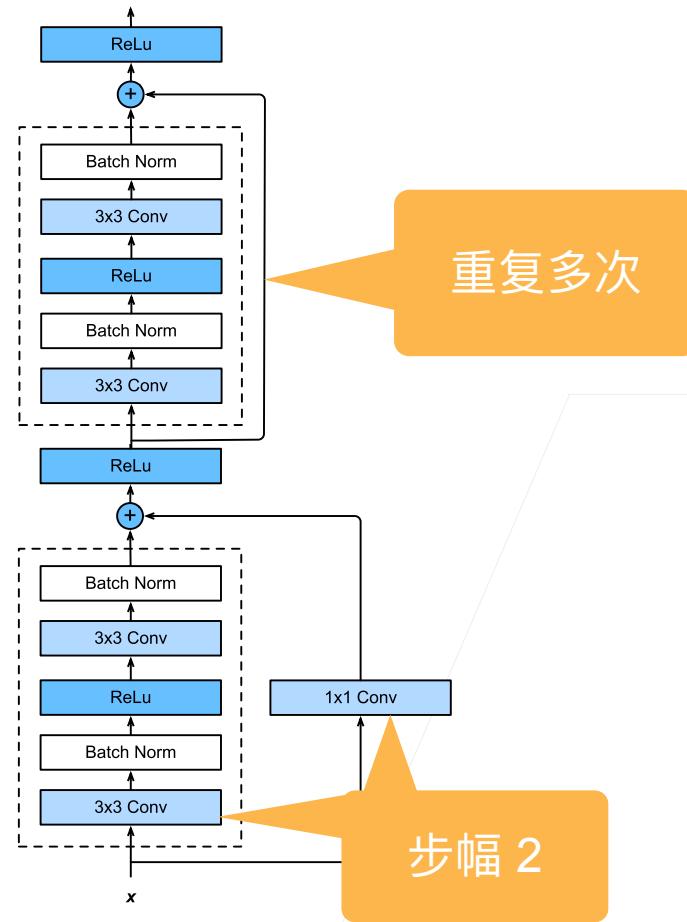


试各种不同的组合



ResNet 块

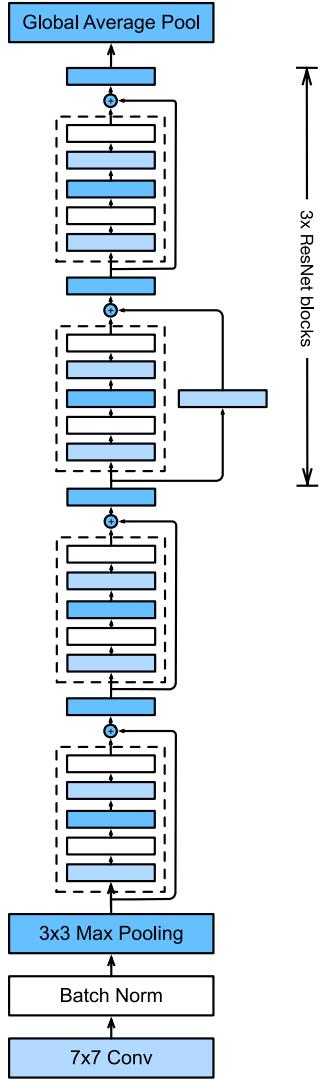
- 高宽减半 ResNet 块
(步幅 2)
- 后接多个高宽不变
ResNet 块





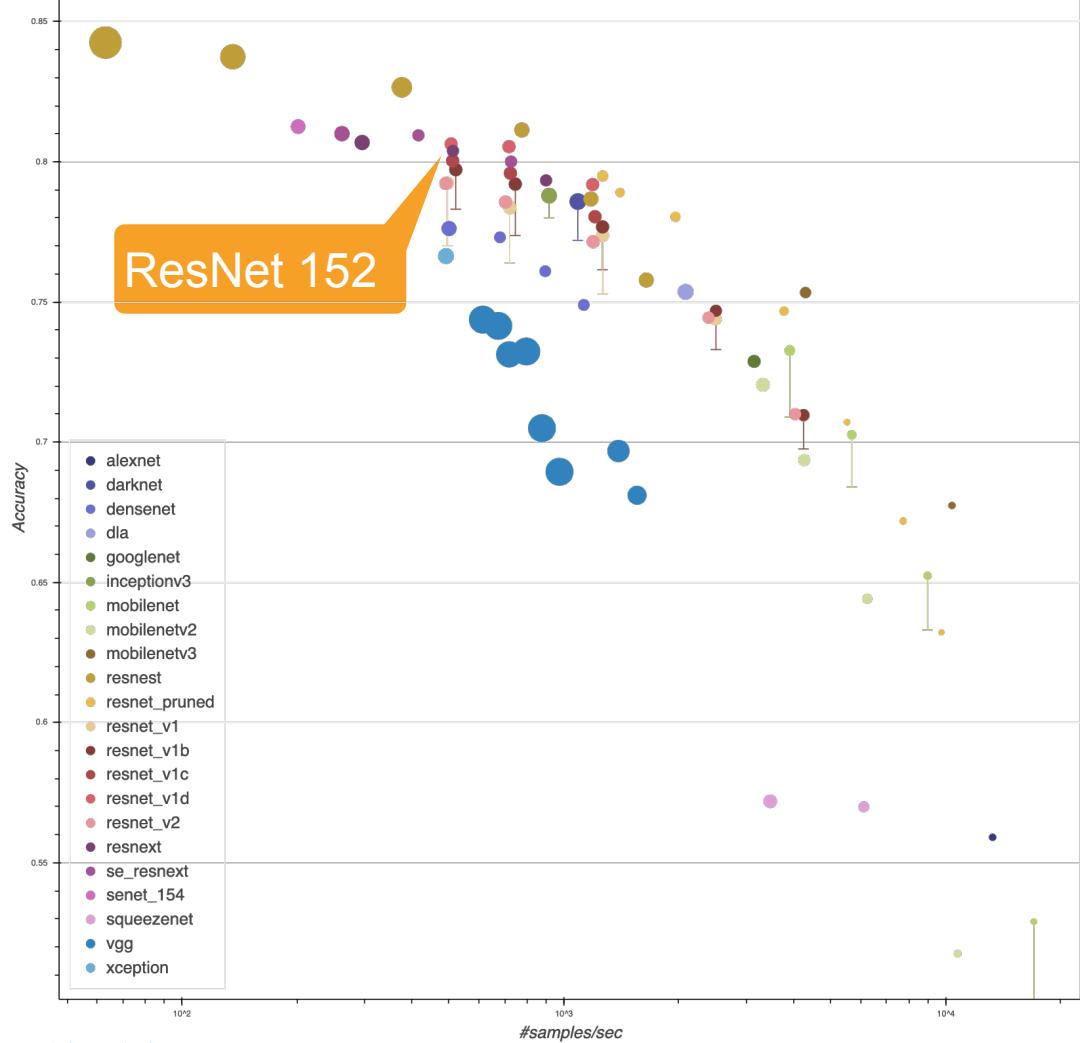
ResNet 架构

- 类似 VGG 和 GoogleNet 的总体架构
- 但替换成了 ResNet 块





GluonCV Model Zoo
[https://cv.gluon.ai/
model_zoo/
classification.html](https://cv.gluon.ai/model_zoo/classification.html)





总结

- 残差块使得很深的网络更加容易训练
 - 甚至可以训练一千层的网络
- 残差网络对随后的深层神经网络设计产生了深远影响，无论是否是卷积类网络还是全连接类网络。



图片分类竞赛

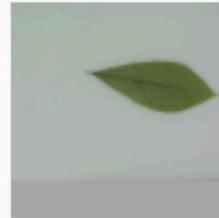
pinus_nigra



magnolia_stellata



asimina_triloba



syringa_reticulata



ulmus_glabra



paulownia_tomentosa



saegusa_pruinosa



pinus_pungens





规则

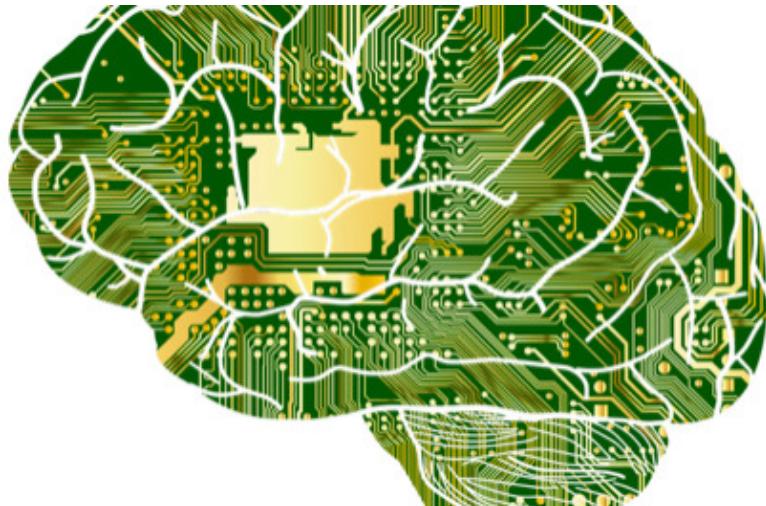
- 竞赛地址：<https://www.kaggle.com/c/classify-leaves>
- 时间：6月26日直播前1小时
- 规则：（跟上次一样）
 - 前5名将获得签名书
 - 自由组队，自由使用算法
 - （你可能需要之后几次直播介绍的方法）

动手学深度学习 v2

李沐 · AWS



深度学习硬件

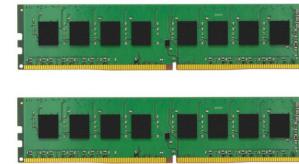
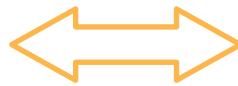




你的GPU电脑

Intel i7

0.15 TFLOPS



DDR4

32 GB



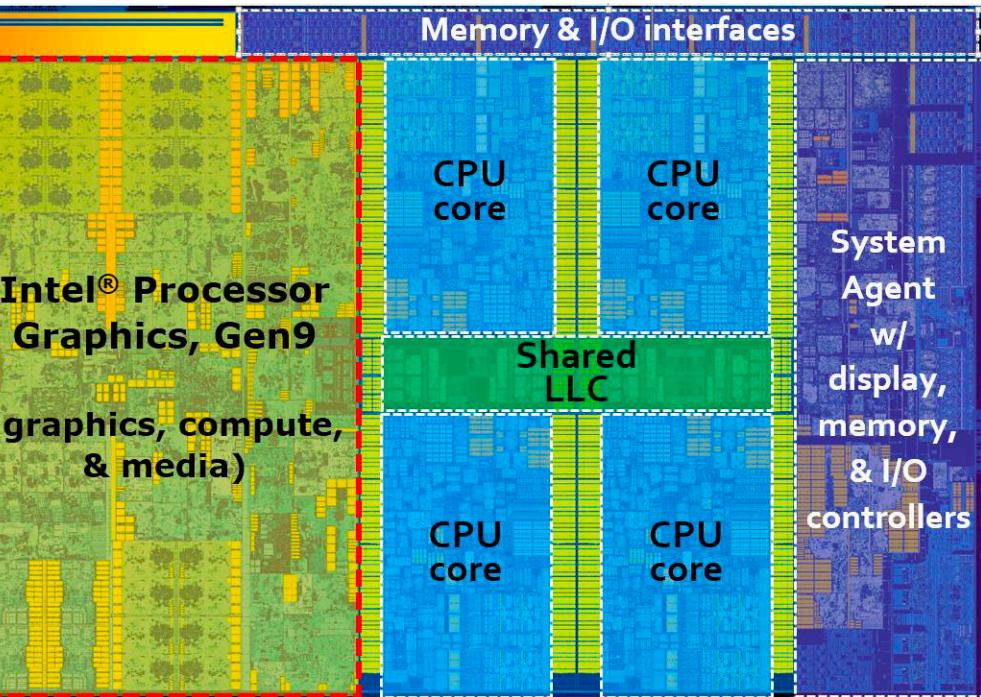
Nvidia Titan X

12 TFLOPS

16 GB



Intel i7-6700K





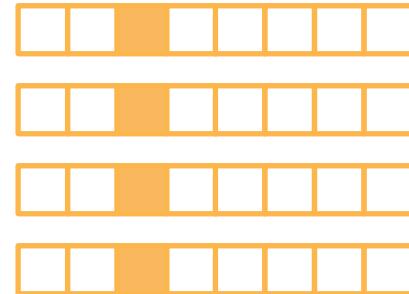
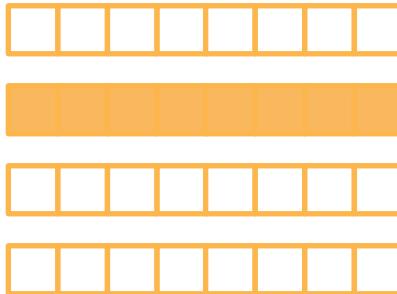
提升CPU利用率 I

- 在计算 $a + b$ 之前，需要准备数据
 - 主内存 -> L3 -> L2 -> L1 -> 寄存器
 - L1 访问延时：0.5 ns
 - L2 访问延时：7 ns ($14 \times L1$)
 - 主内存访问延时：100ns ($200 \times L1$)
- 提升空间和时间的内存本地性
 - 时间：重用数据使得保持它们在缓存里
 - 空间：按序读写数据使得可以预读取



样例分析

- 如果一个矩阵是按列存储，访问一行会比访问一列要快
 - CPU 一次读取 64 字节（缓存线）
 - CPU 会“聪明的”提前读取下一个（缓存线）





提升CPU利用率 II

- 高端 CPU 有几十个核
 - EC2 P3.16xlarge: 2 Intel Xeon CPUs, 32 物理核
- 并行来利用所有核
 - 超线程不一定提升性能，因为它们共享寄存器



样例分析

- 左边比右边慢 (python)

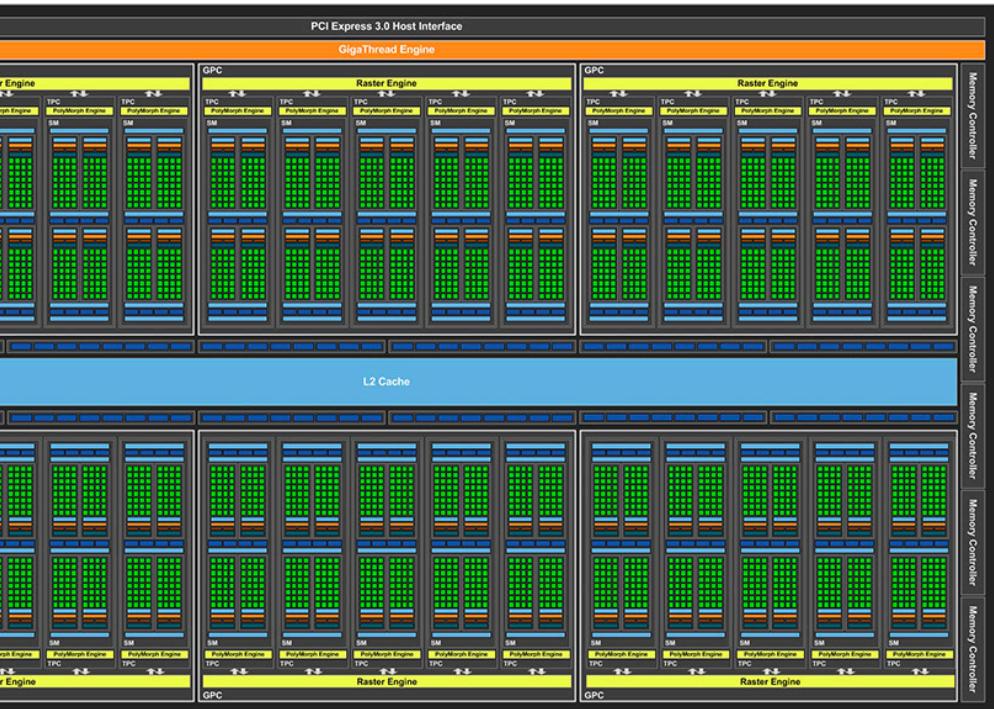
```
for i in range(len(a)):  
    c[i] = a[i] + b[i]
```

```
c = a + b
```

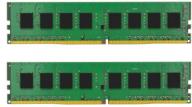
- 左边调用 n 次函数，每次调用有开销
- 右边很容易被并行（例如下面C++实现）

```
#pragma omp for  
for (i=0; i<a.size(); i++) {  
    c[i] = a[i] + b[i]  
}
```

Nvidia Titan X (Pascal)



CPU vs GPU



一般 / 高端

# 核	6 / 64	2K / 4K
TFLOPS	0.2 / 1	10 / 100
内存大小	32 GB / 1 TB	16 GB / 32 GB
内存带宽	30 GB/s / 100 GB/s	400 GB/s / 1 TB/s
控制流	强	弱



提升GPU利用率

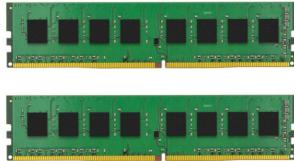
- 并行
 - 使用数千个线程
- 内存本地性
 - 缓存更小，架构更加简单
- 少用控制语句
 - 支持有限
 - 同步开销很大



CPU/GPU 带宽



30 GB/s



PCIe 3.0 16x: 16 GB/s



480 GB/s

- 不要频繁在CPU和GPU之前传数据：带宽限制，同步开销



更多的 CPUs 和 GPUs

- CPU: AMD, ARM
- GPU: AMD, Intel, ARM, Qualcomm...





CPU/GPU高性能计算编程

- CPU: C++ 或者任何高性能语言
 - 编译器成熟
- GPU
 - Nvidia 上用 CUDA
 - 编译器和驱动成熟
 - 其他用 OpenCL
 - 质量取决于硬件厂商



总结

- CPU：可以处理通用计算。性能优化考虑数据读写效率和多线程
- GPU：使用更多的小核和更好的内存带宽，适合能大规模并行的计算任务

动手学深度学习 v2

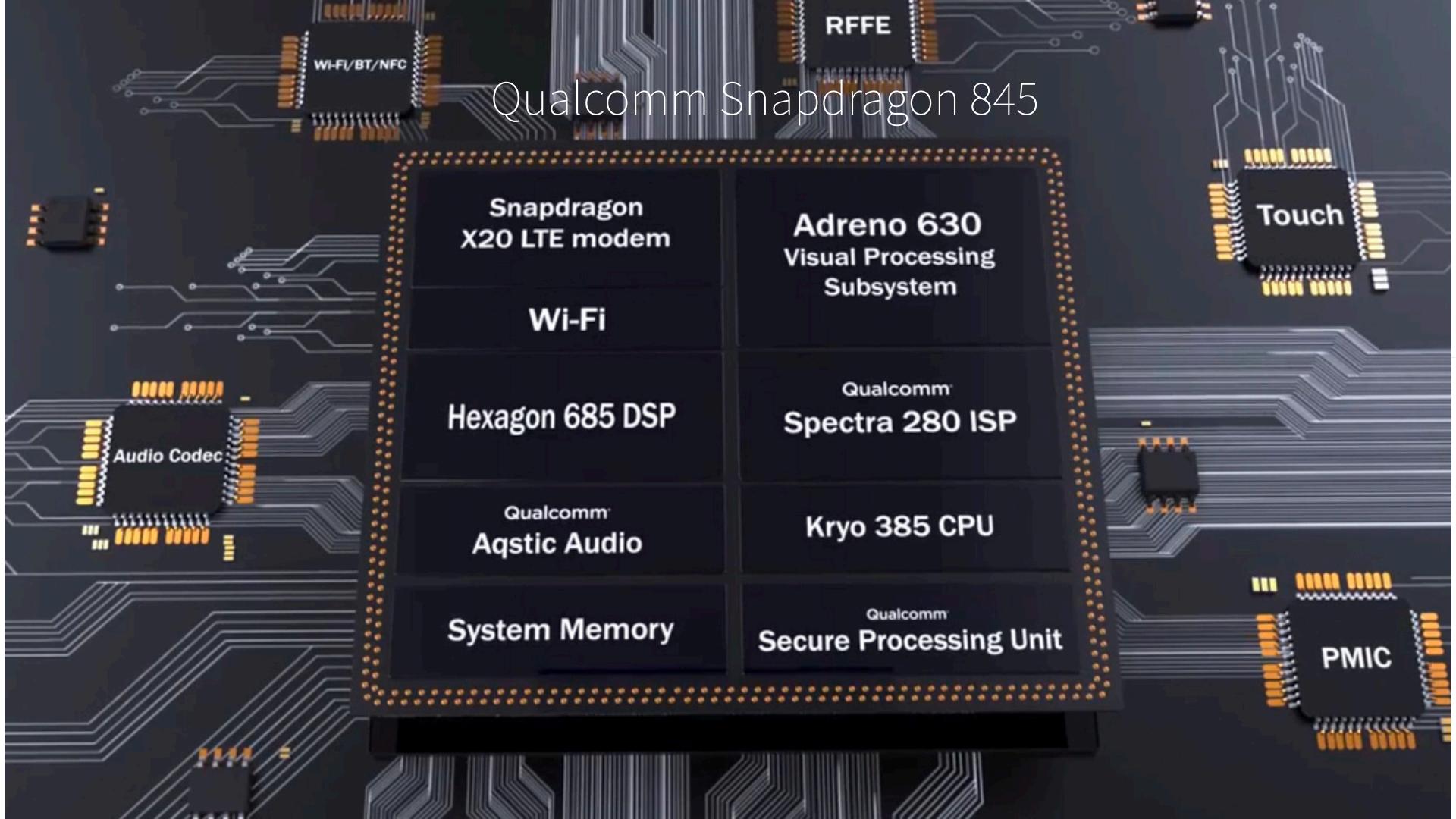
李沐 · AWS



更多的芯片



Qualcomm Snapdragon 845





DSP：数字信号处理

- 为数字信号处理算法设计：点积，卷积，FFT
- 低功耗、高性能
 - 比移动GPU快5x，功耗更低
- VLIW: Very long instruction word
 - 一条指令计算上百次乘累加
- 编程和调试困难
- 编译器质量良莠不齐



可编程阵列 (FPGA)

- 有大量可以编程逻辑单元和可配置的连接
- 可以配置成计算复杂函数
 - 编程语言：VHDL，Verilog
- 通常比通用硬件更高效
- 工具链质量良莠不齐
- 一次“编译”需要数小时

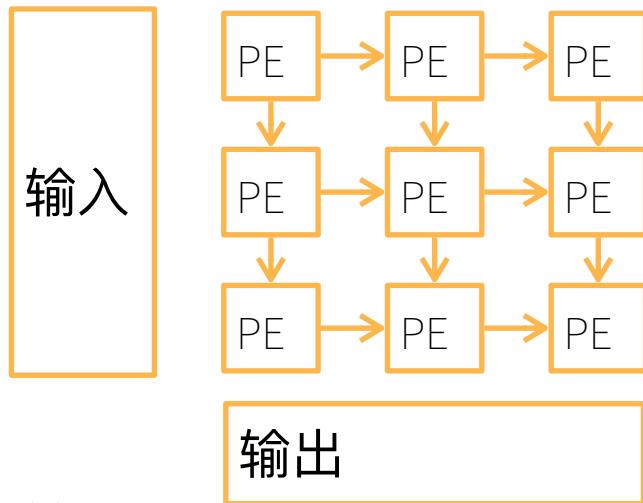


- 深度学习的热门领域
 - 大公司都在造自己的芯片 (Intel, Qualcomm, Google, Amazon, Facebook, ...)
- Google TPU 是标志性芯片
 - 能够媲美Nvidia GPU性能
 - 在Google大量部署
 - 核心是 systolic array



Systolic Array

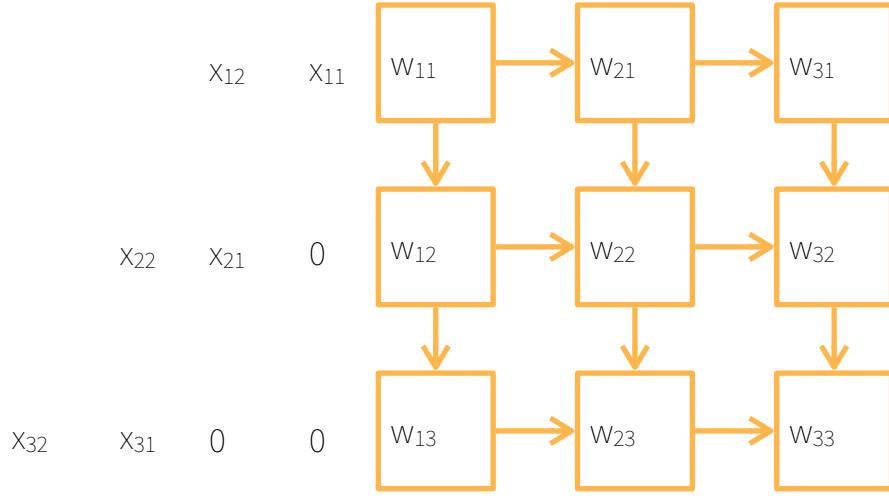
- 计算单元（PE）阵列
- 特别适合做矩阵乘法
- 设计和制造相对简单





Systolic Array的矩阵乘法

Time 0



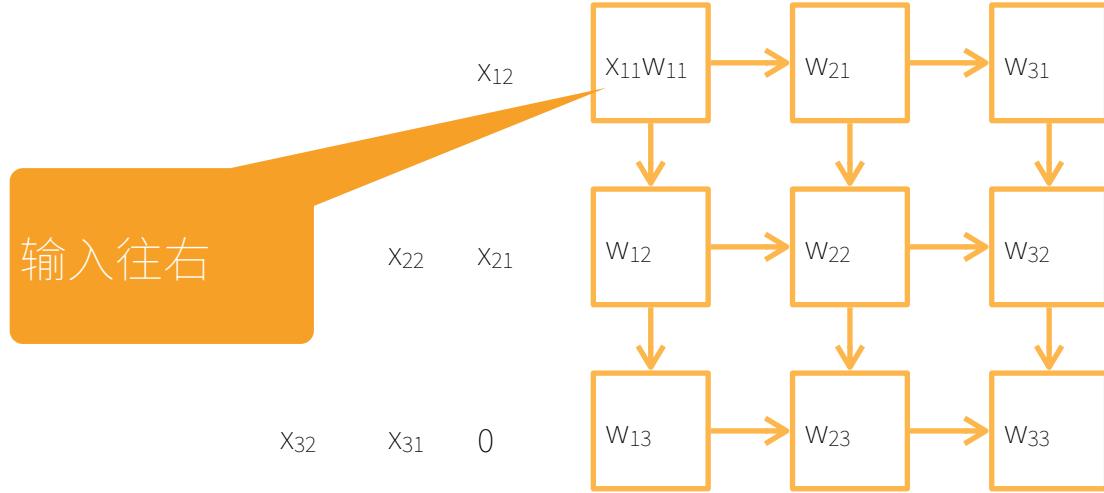
$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

$3 \times 2 \quad 3 \times 3 \quad 3 \times 2$



Systolic Array的矩阵乘法

Time 1



$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

$3 \times 2 \quad 3 \times 3 \quad 3 \times 2$



Matrix Multiplication with Systolic Array

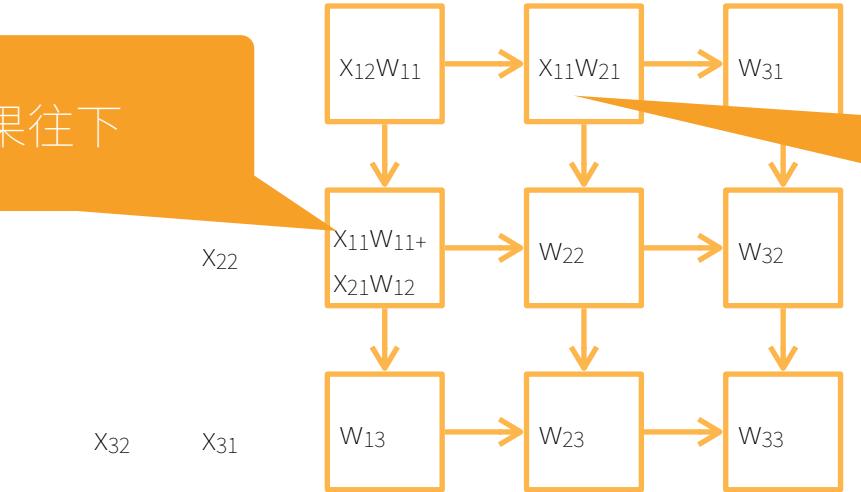
Time 2

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

3x2 3x3 3x2

结果往下

输入往右





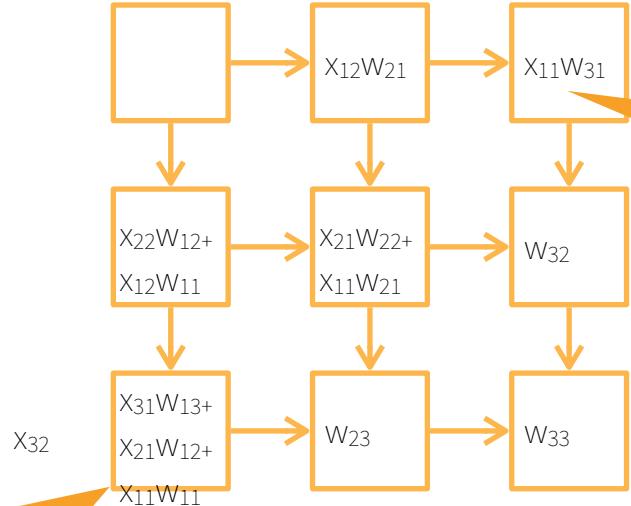
Matrix Multiplication with Systolic Array

Time 3

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

3x2 3x3 3x2

输入往右



结果往下

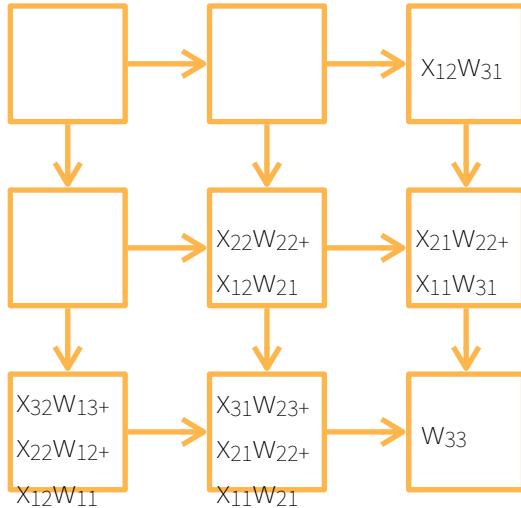


Matrix Multiplication with Systolic Array

Time 4

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

3x2 3x3 3x2



y_{11}

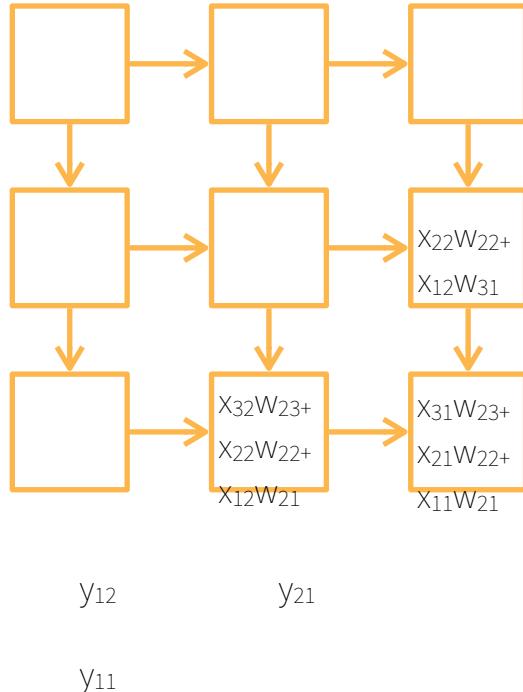


Matrix Multiplication with Systolic Array

Time 5

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

3x2 3x3 3x2



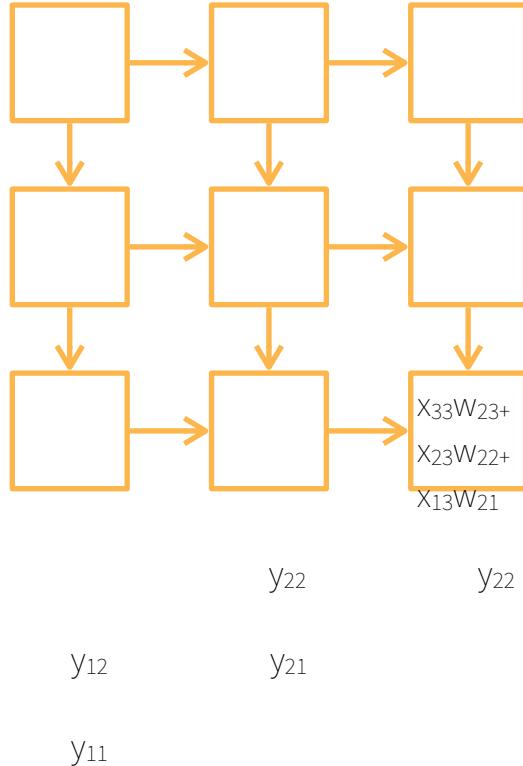


Matrix Multiplication with Systolic Array

Time 6

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

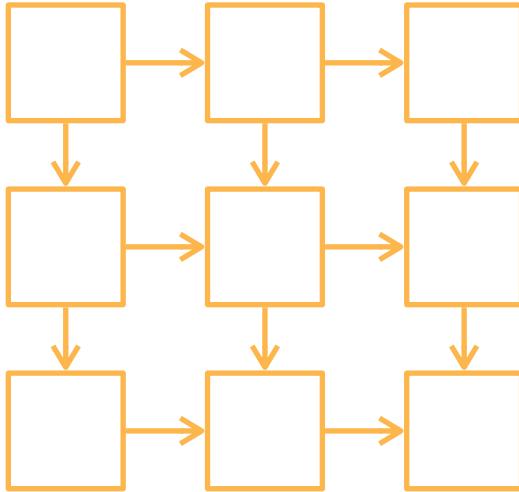
3x2 3x3 3x2





Matrix Multiplication with Systolic Array

Time 7



y_{12}

y_{11}

y_{21}

y_{22}

y_{32}

y_{31}

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

$3 \times 2 \quad 3 \times 3 \quad 3 \times 2$



Systolic Array

- 对于一般的矩阵乘法，通过切开和填充来匹配SA大小
- 批量输入来降低延时
- 通常有其他硬件单元来处理别的 NN 操作子，例如激活层



总结





多GPU并行



2014农历新年



单机多卡并行

- 一台机器可以安装多个GPU (1-16)
- 在训练和预测时，我们将一个小批量计算切分到多个GPU上来达到加速目的
- 常用切分方案有
 - 数据并行
 - 模型并行
 - 通道并行 (数据+模型并行)

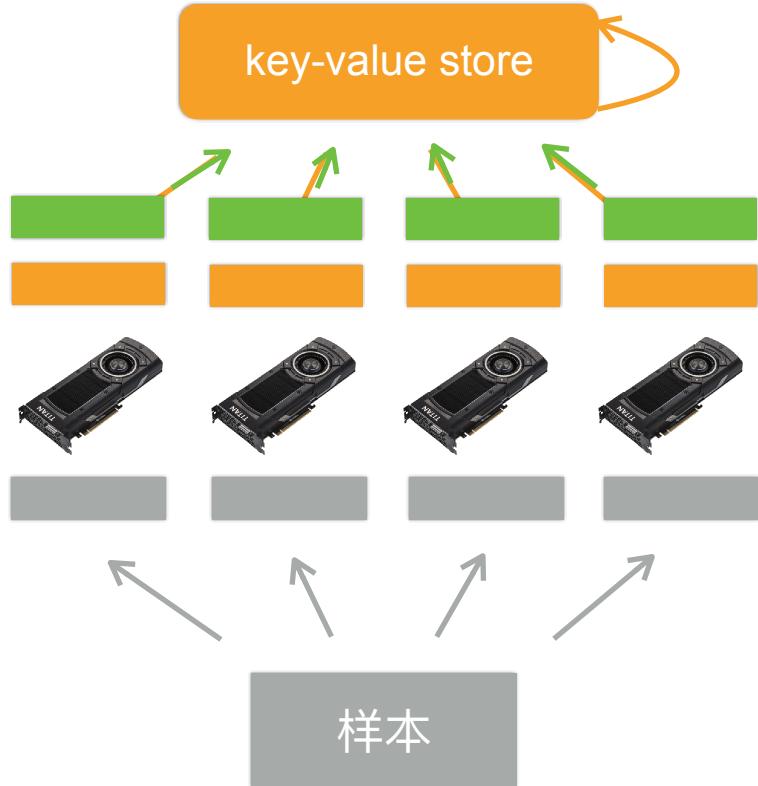


数据并行 vs 模型并行

- 数据并行：将小批量分成 n 块，每个GPU拿到完整参数计算一块数据的梯度
 - 通常性能更好
- 模型并行：将模型分成 n 块，每个GPU拿到一块模型计算它的前向和方向结果
 - 通常用于模型大到单GPU放不下



数据并行



1. 读一个数据块
2. 拿回参数
3. 计算梯度
4. 发出梯度
5. 更新梯度

总结



- 当一个模型能用单卡计算时，通常使用数据并行拓展到多卡上
- 模型并行则用在超大模型上



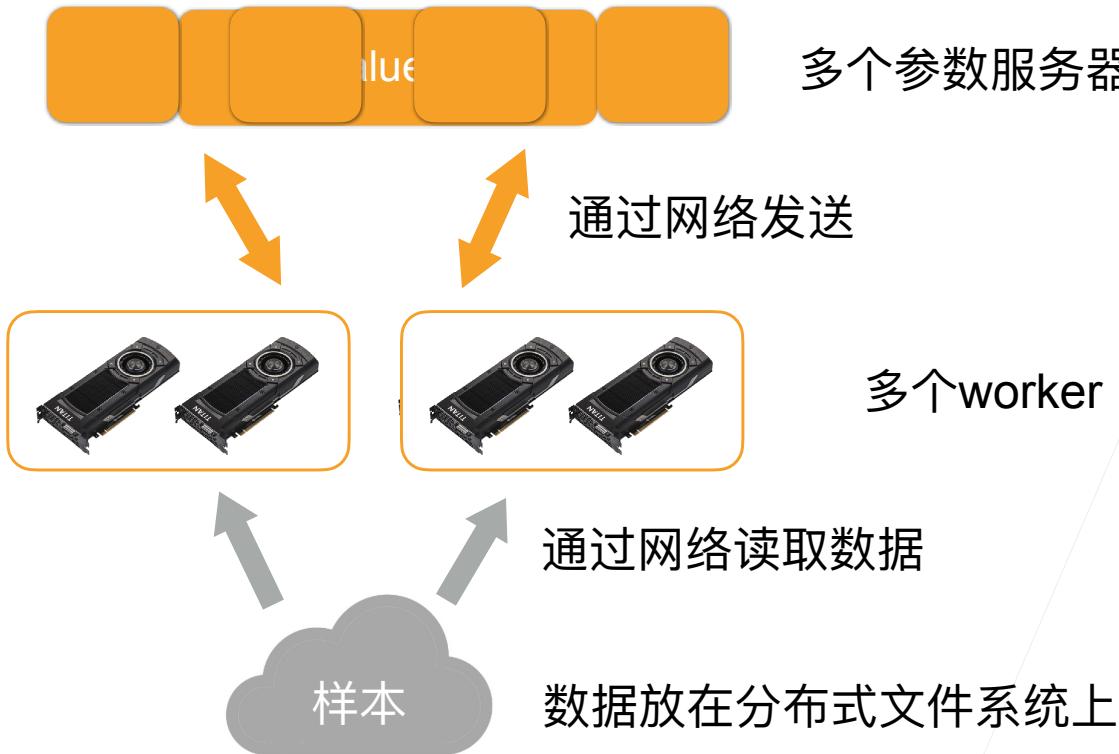
分布式计算



2015年在CMU装的高性价比集群

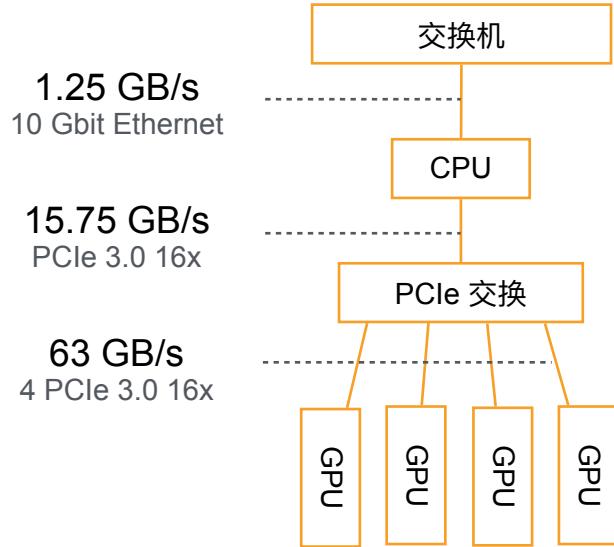


分布式计算

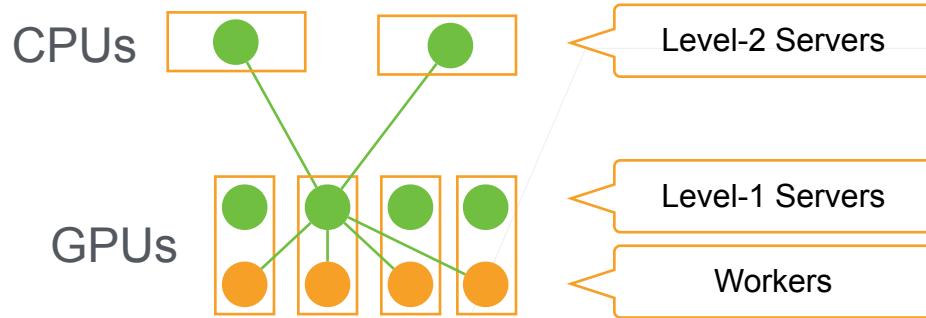




GPU 机器架构



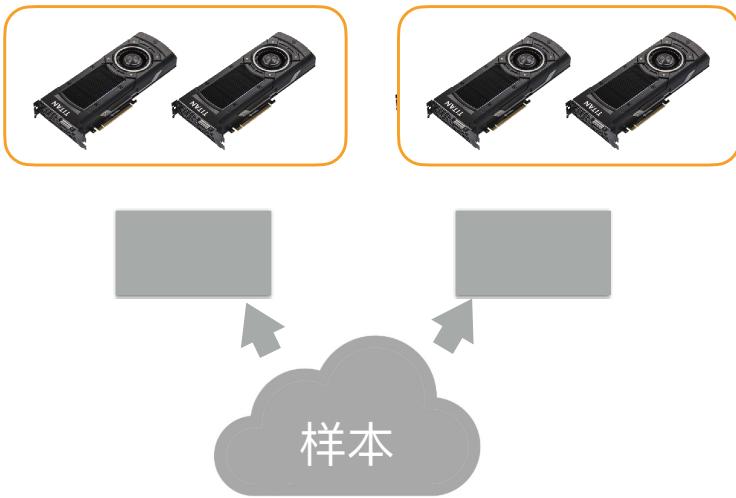
层次的参数服务器





计算一个小批量

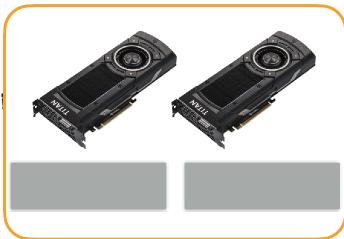
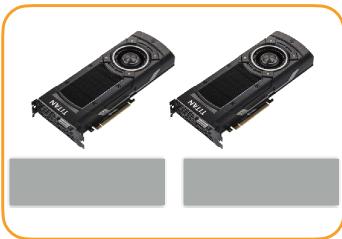
- 每个计算服务器读取小批量中的一块





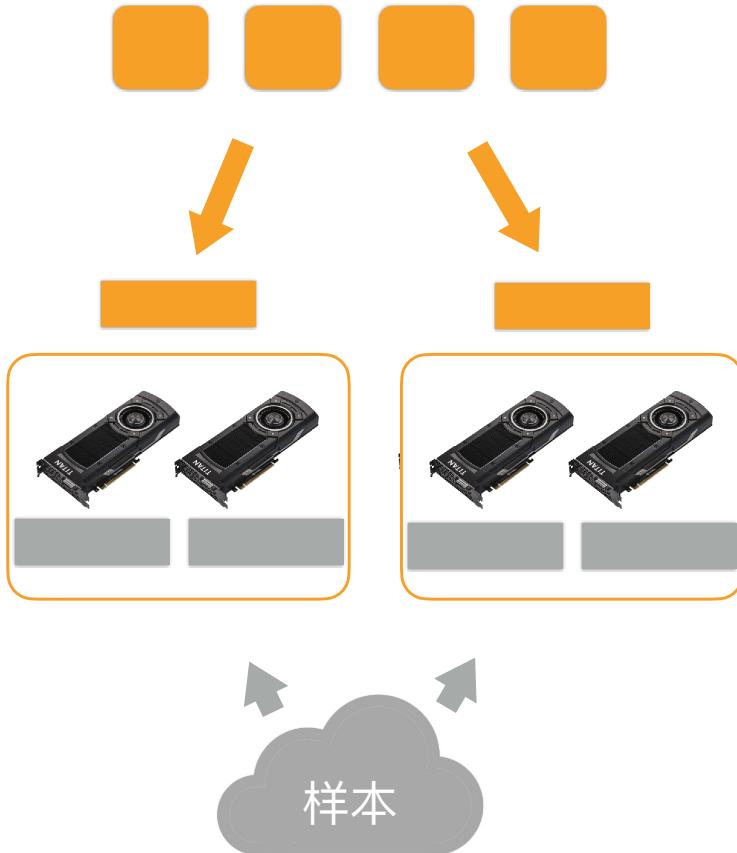
计算一个小批量

- 进一步将数据切分到每个 GPU 上





计算一个小批量



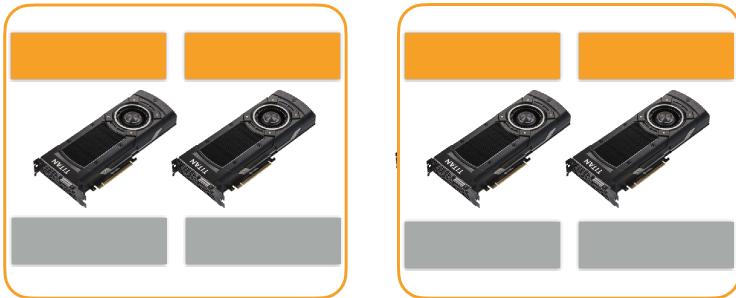
- 每个worker从参数服务器那里获取模型参数



计算一个小批量



- 复制参数到每个GPU上

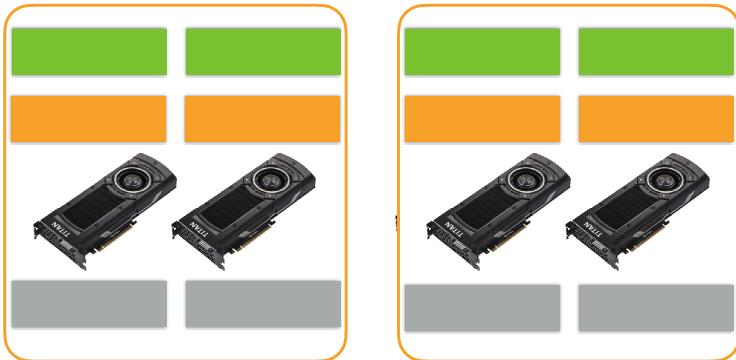




计算一个小批量

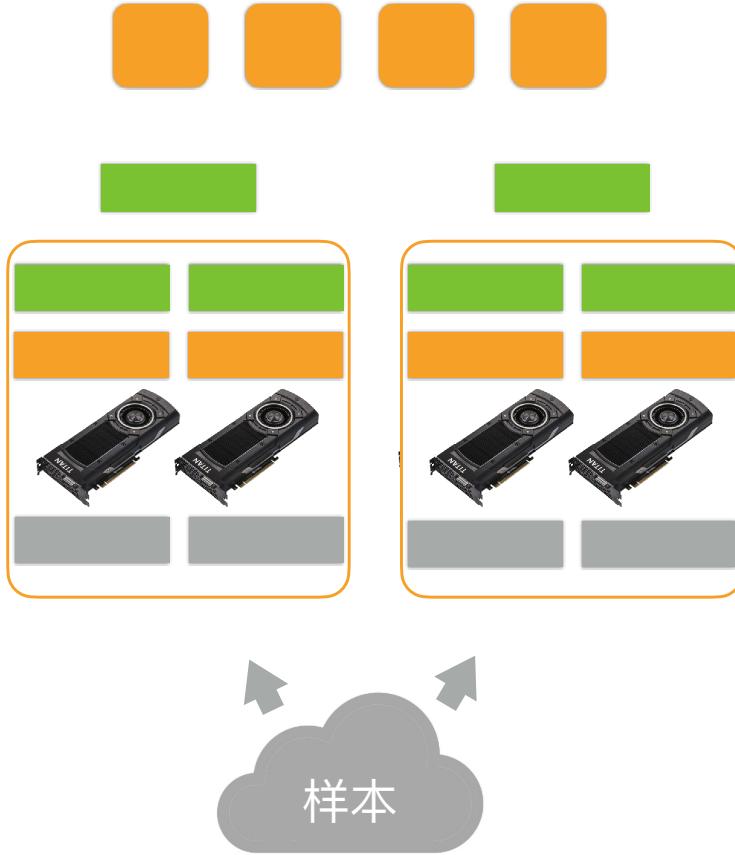


- 每个GPU计算梯度





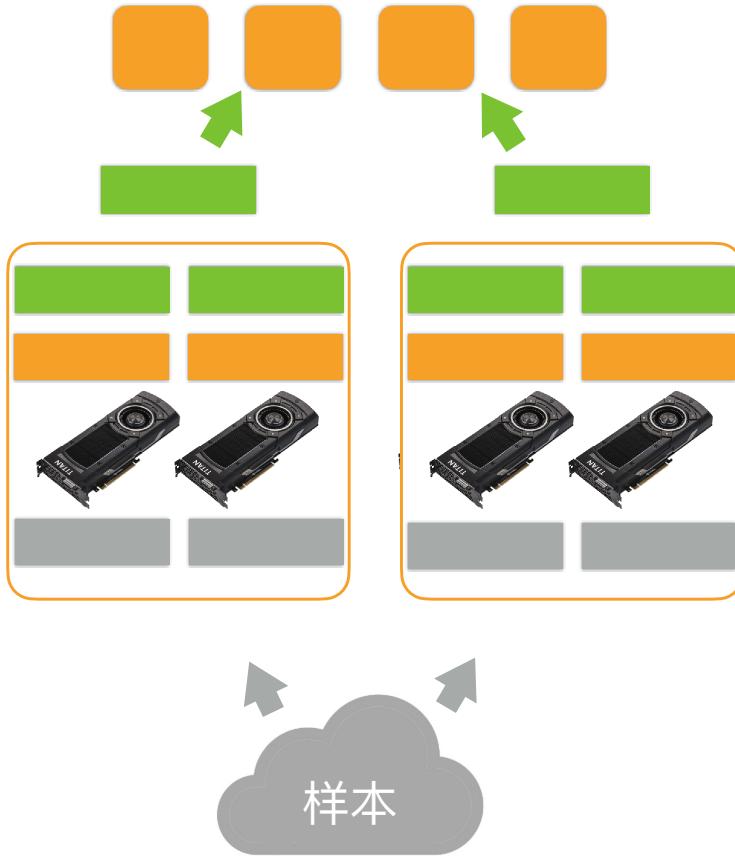
计算一个小批量



- 讲所有GPU上的梯度求和



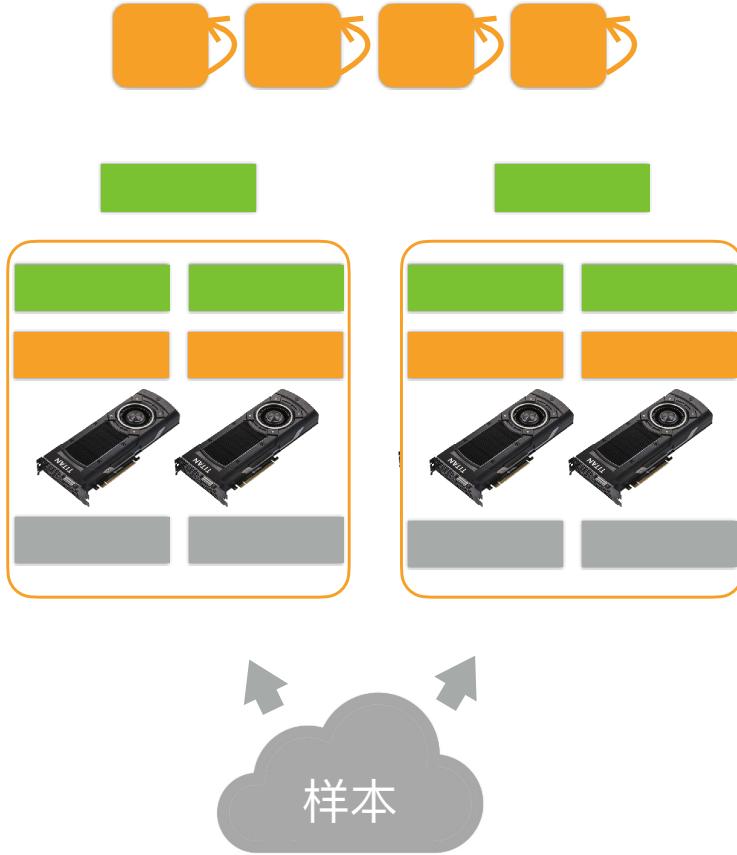
计算一个小批量



- 梯度传回服务器



计算一个小批量



- 每个服务器对梯度求和，并更新参数



同步 SGD

- 这里每个worker都是同步计算一个批量，称为同步SGD
- 假设有 n 个GPU，每个GPU每次处理 b 个样本，那么同步SGD等价于在单 GPU 运行批量大小为 nb 的SGD
- 在理想情况下， n 个GPU可以得到相对单GPU的 n 倍加速

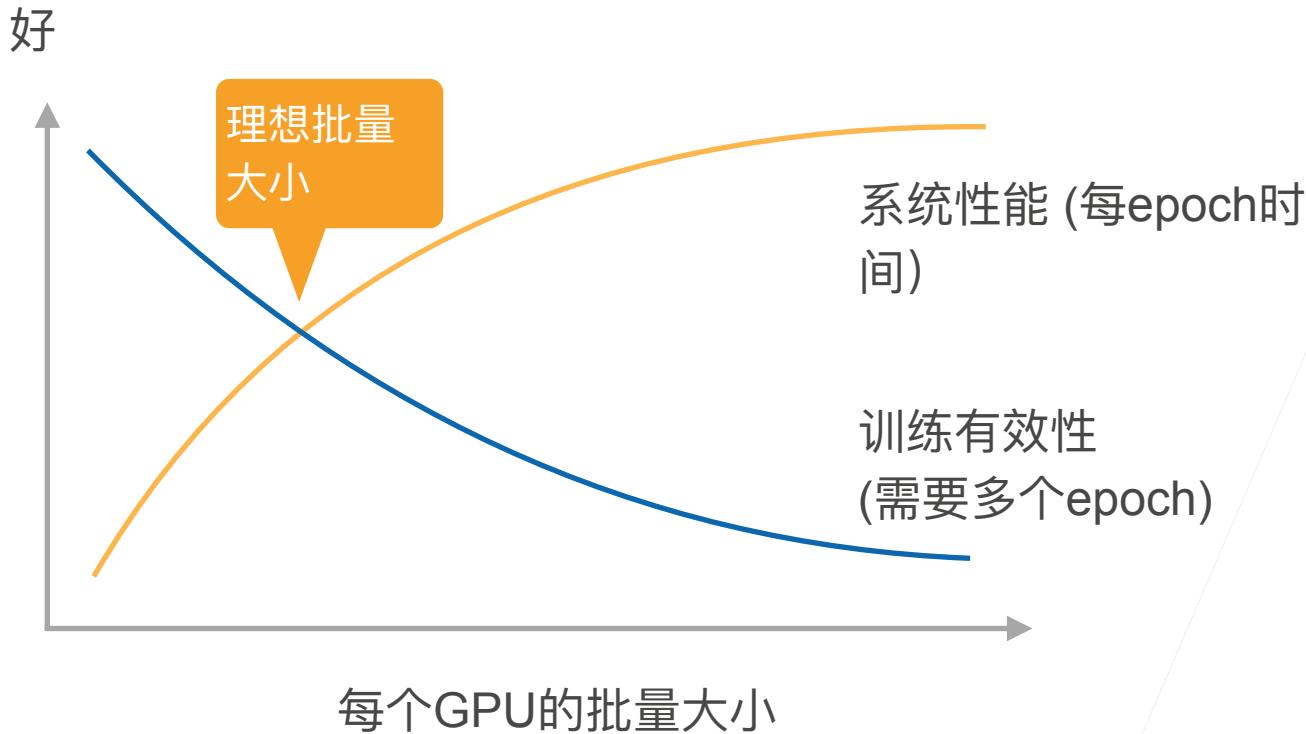


性能

- t_1 = 在单GPU上计算 b 个样本梯度时间
- 假设有 m 个参数，一个 worker 每次发送和接收 m 个参数、梯度
 - t_2 = 发送和接收所用时间
- 每个批量的计算时间为 $\max(t_1, t_2)$
 - 选取足够大的 b 使得 $t_1 > t_2$
 - 增加 b 或 n 导致更大的批量大小，导致需要更多计算来得到给定的模型精度



性能的权衡





实践时的建议

- 使用一个大数据集
- 需要好的GPU-GPU和机器-机器带宽
- 高效的数据读取和预处理
- 模型需要有好的计算 (FLOP) 通讯 (model size) 比
 - Inception > ResNet > AlexNet
- 使用足够大的批量大小来得到好的系统性能
- 使用高效的优化算法对对应大批量大小



总结

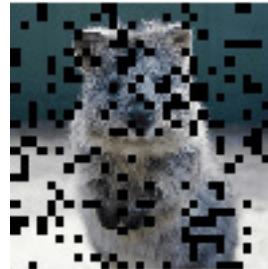
- 分布式同步数据并行是多GPU数据并行在多机器上的拓展
- 网络通讯通常是瓶颈
- 需要注意使用特别大的批量大小时收敛效率
- 更复杂的分布式有异步、模型并行



数据增广



$p=1.0$



$\text{size_percent}=0.30$



$p=0.50$



$\text{cutoff}=0.00$



CES上的真实故事

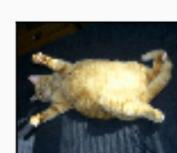
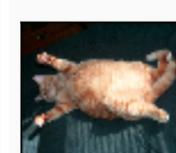
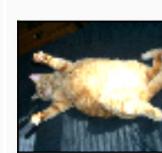
- 有一家做智能售货机的公司，发现他们的演示机器在现场效果很差，因为现场
 - 不同的色温
 - 桌面的灯光反射不一样
- 他们连夜现场收集了数据，训练了一个新的模型，同时买了一块新桌布





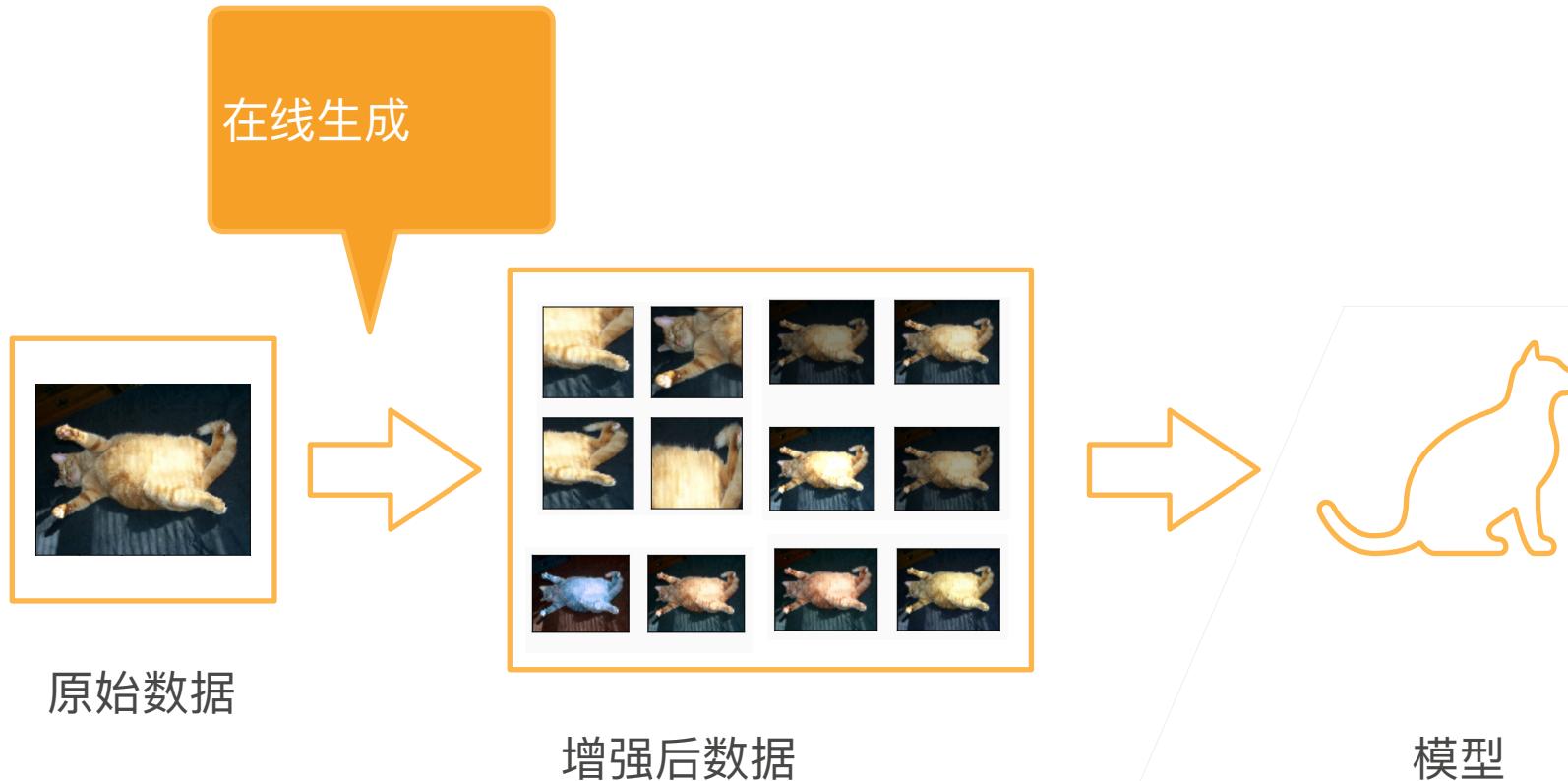
数据增强

- 增加一个已有数据集，使得有更多的多样性
 - 在语言里面加入各种不同的背景噪音
 - 改变图片的颜色和形状





使用增强数据训练





翻转

- 左右翻转



- 上下翻转



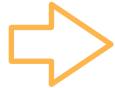
- 不总是可行





切割

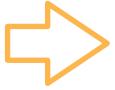
- 从图片中切割一块，然后变形到固定形状
 - 随机高宽比 (e.g. [3/4, 4/3])
 - 随机大小 (e.g. [8%, 100%])
 - 随机位置





颜色

- 改变色调，饱和度，明亮度(e.g. [0.5, 1.5])



亮度

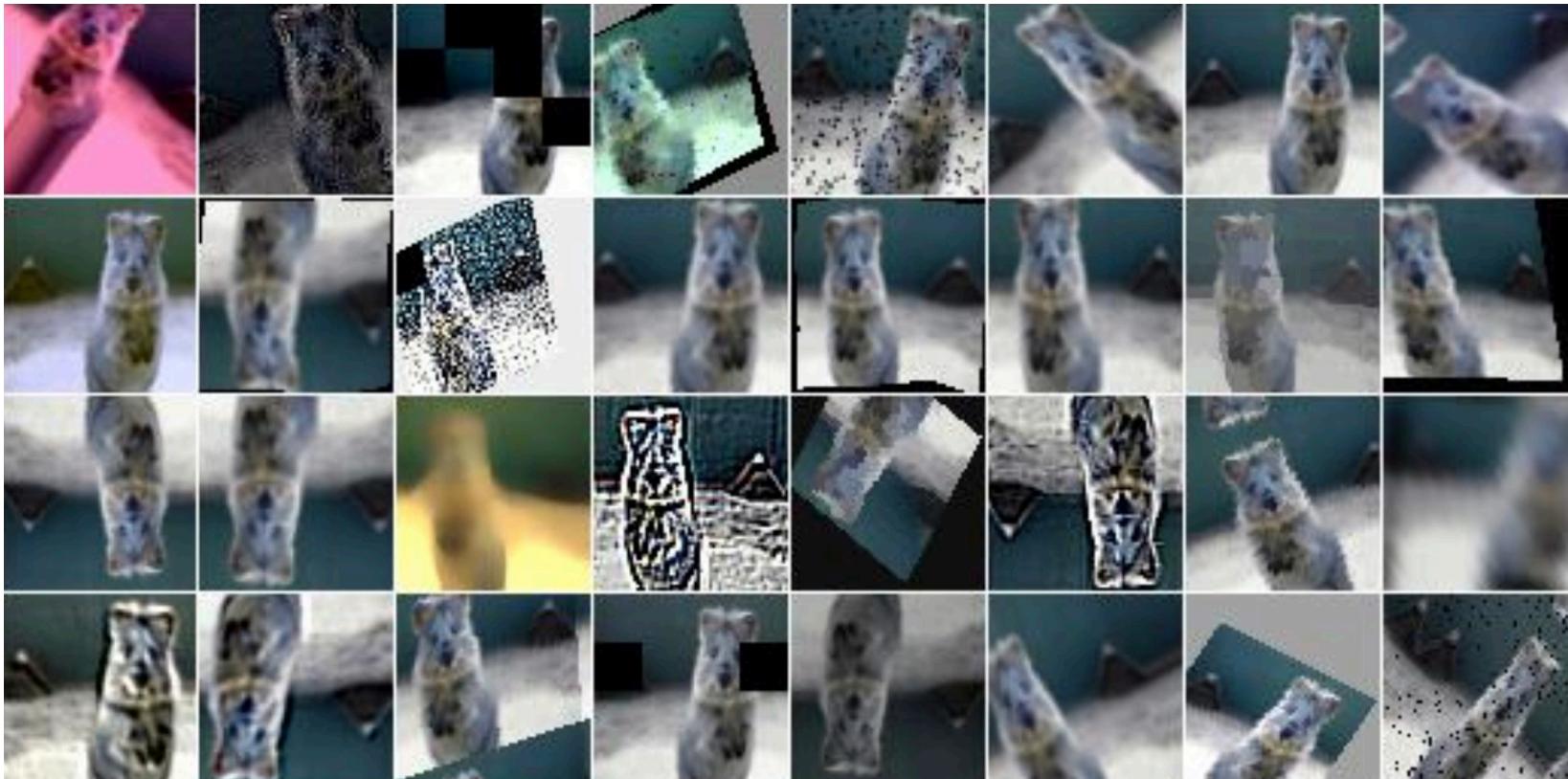


色调





几十种其他的方法





总结

- 数据增广通过变形数据来获取多样性从而使得模型泛化性能更好
- 常见图片增广包括翻转、切割、变色

动手学深度学习 v2

李沐 · AWS



微调





标注一个数据集很贵

我的数据及



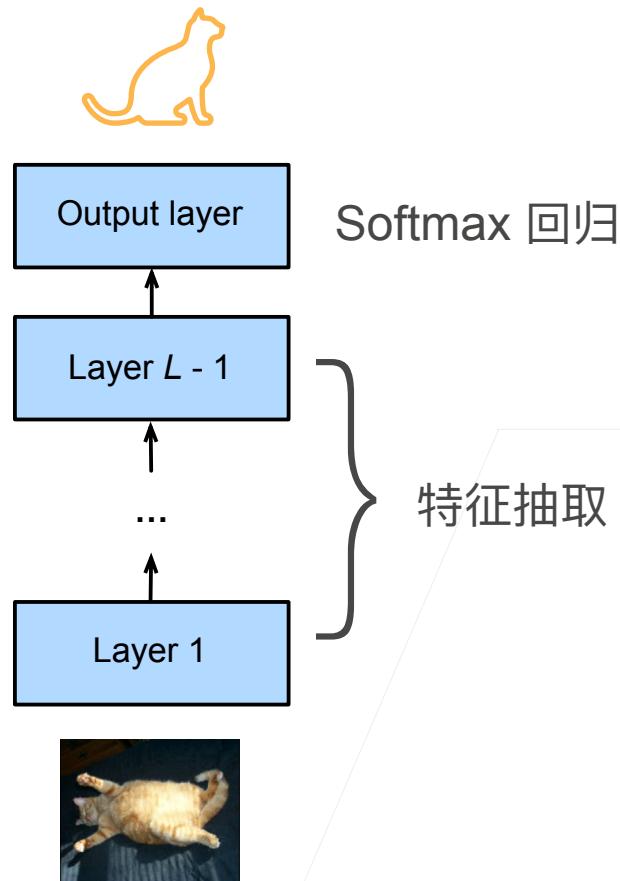
2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6

样本数	1.2 M	50K	60 K
类别数	1,000	100	10



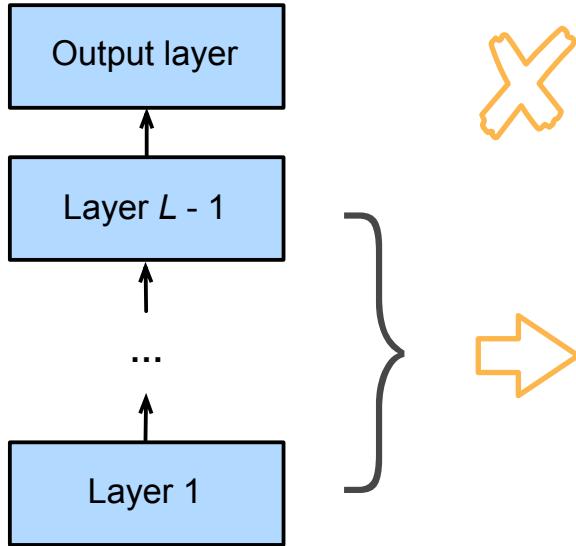
网络架构

- 一个神经网络一般可以分成两块
 - 特征抽取将原始像素变成容易线性分割的特征
 - 线性分类器来做分类





微调



不能直接使用，因为标
号可能变了

可能仍然可以对我数据
集做特征抽取



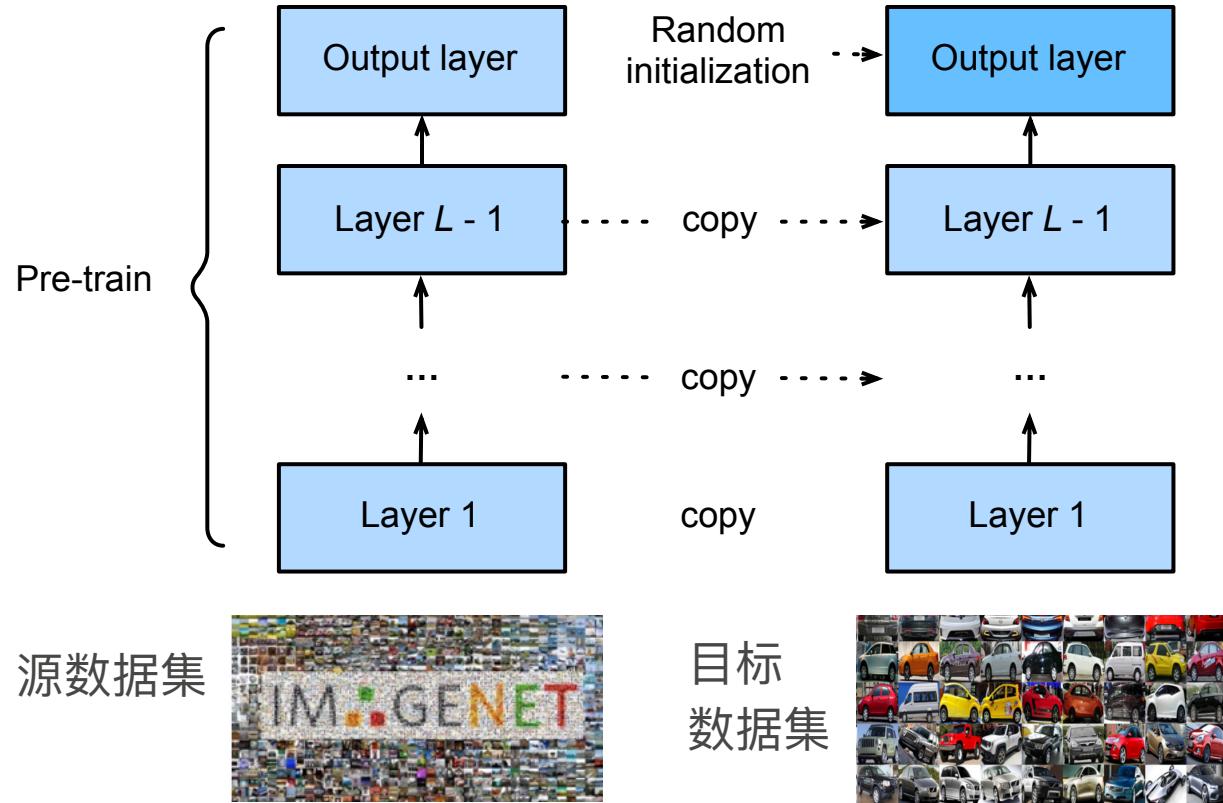
源数据集



目标数据集



微调中的权重初始化





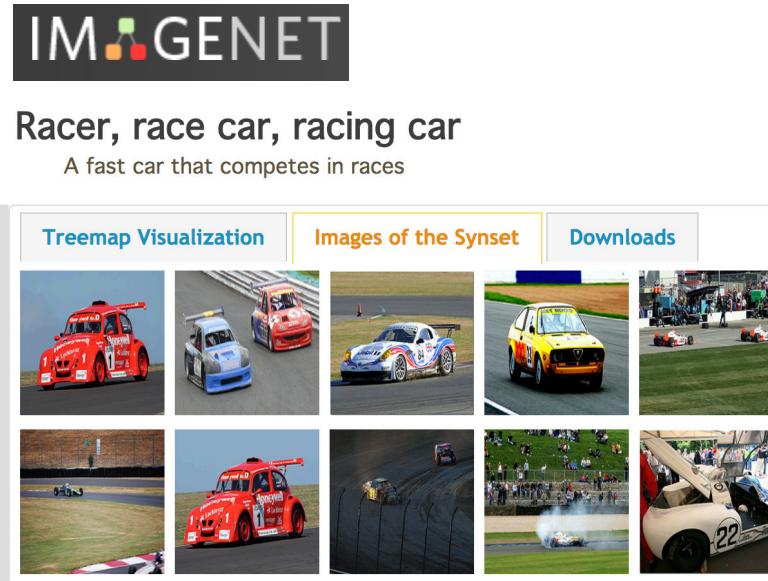
训练

- 是一个目标数据集上的正常训练任务，但使用更强的正则化
 - 使用更小的学习率
 - 使用更少的数据迭代
- 源数据集远复杂于目标数据，通常微调效果更好



重用分类器权重

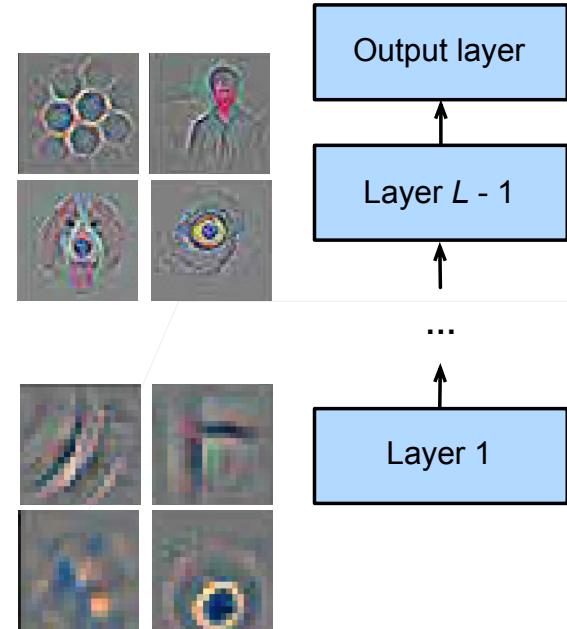
- 源数据集可能也有目标数据中的部分标号
- 可以使用预训练好模型分类器中对应标号对应的向量来做初始化





预定一些层

- 神经网络通常学习有层次的特征表示
 - 低层次的特征更加通用
 - 高层次的特征则更跟数据集相关
- 可以固定底部一些层的参数，不参与更新
 - 更强的正则

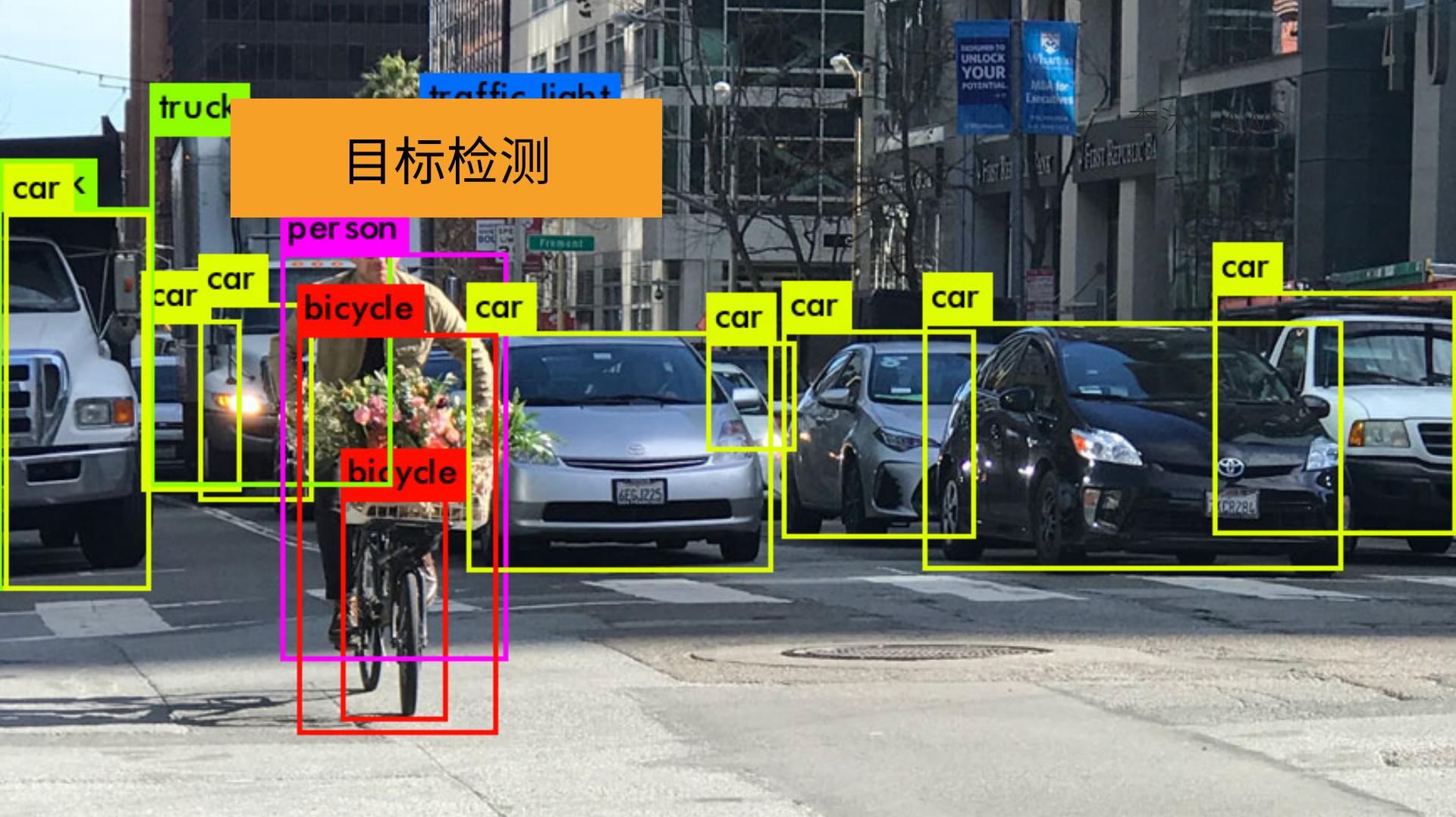




总结

- 微调通过使用在大数据上得到的预训练好的模型来初始化模型权重来完成提升精度
- 预训练模型质量很重要
- 微调通常速度更快、精度更高

目标检测



李沐 · AWS

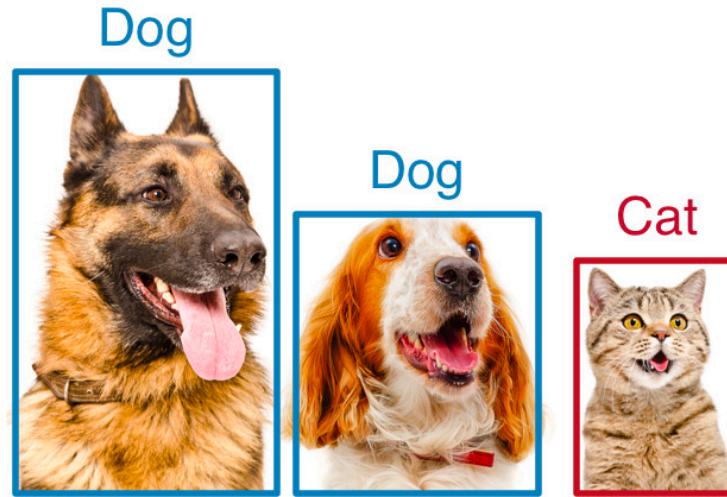


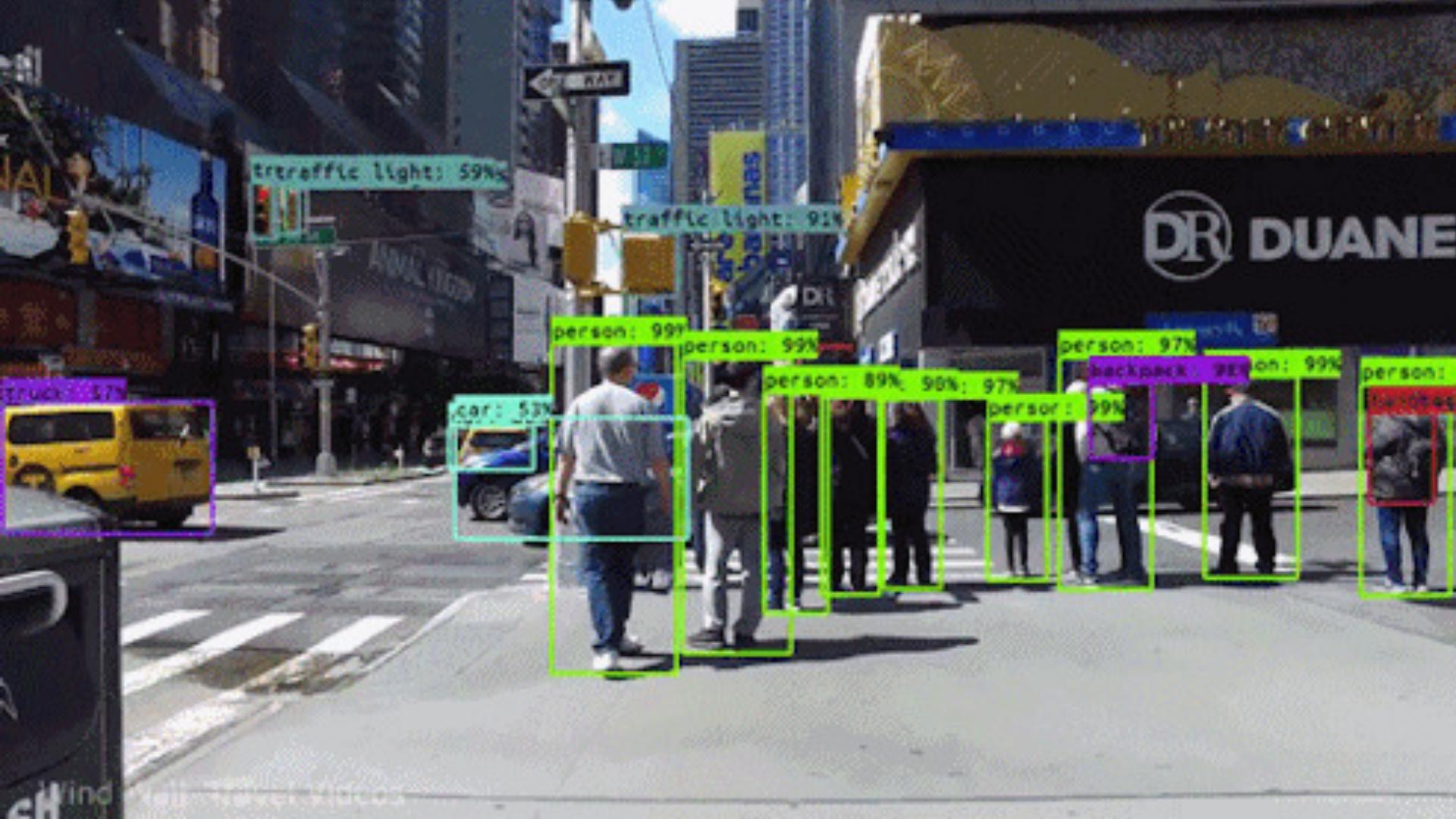
图片分类



Dog

目标检测

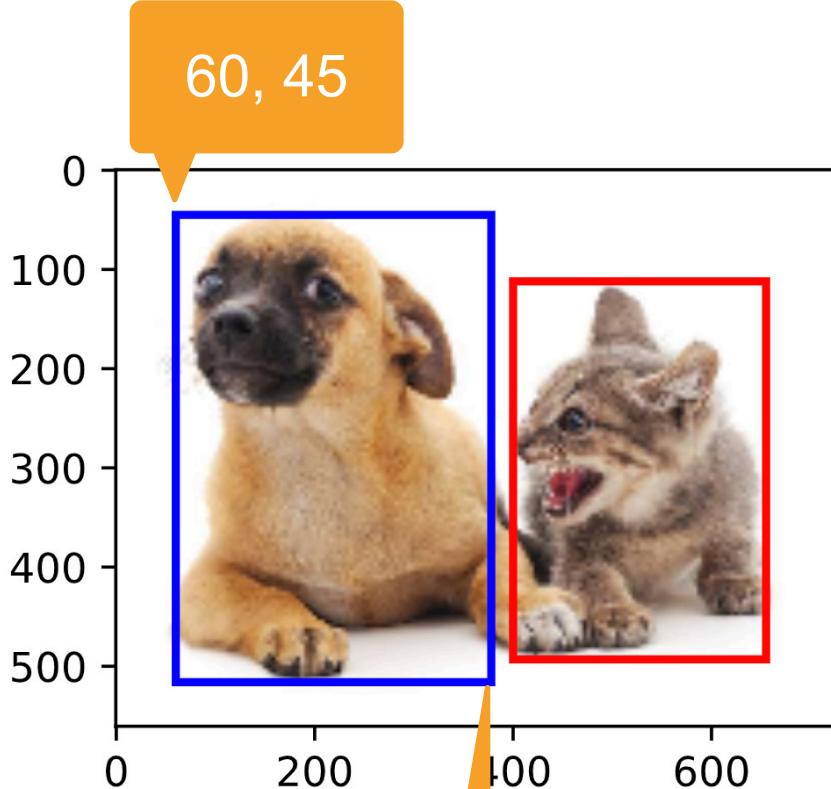






边缘框

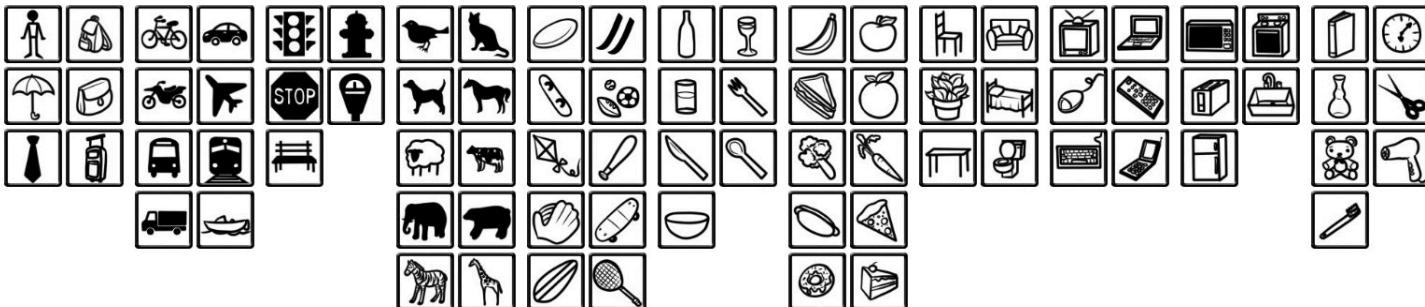
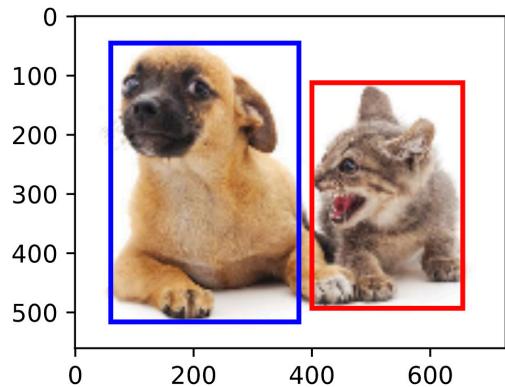
- 一个边缘框可以
通过4个数字定义，
 - (左上 x, 左上 y,
右下 x, 右下 y)
 - (左上 x, 左上 y,
宽, 高)





目标检测数据集

- 每行表示一个物体
 - 图片文件名，物体类别，边缘框
- COCO (cocodataset.org)
 - 80 物体，330K 图片，1.5M 物体





总结

- 物体检测识别图片里的多个物体的类别和位置
- 位置通常用边缘框表示



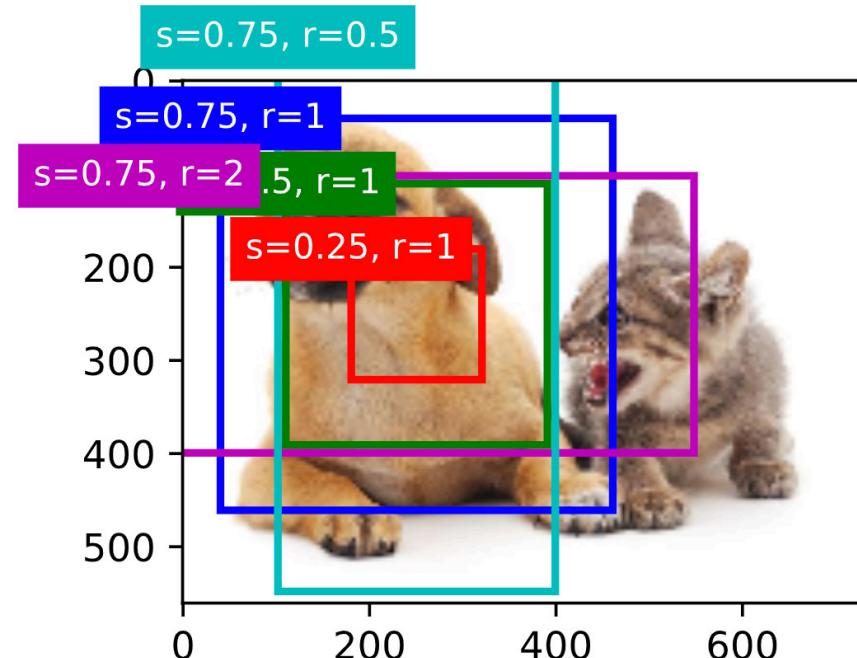
锚框





锚框

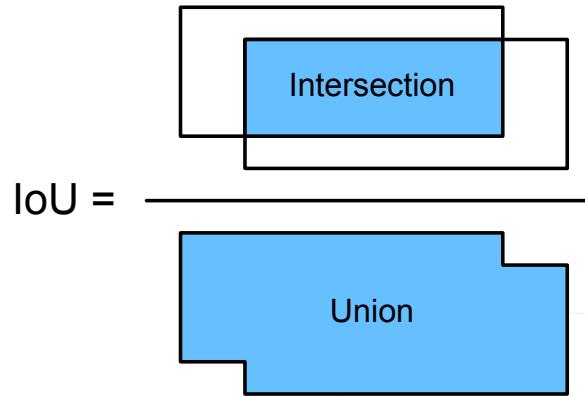
- 一类目标检测算法是基于锚框
 - 提出多个被称为锚框的区域（边缘框）
 - 预测每个锚框里是否含有关注的物体
 - 如果是，预测从这个锚框到真实边缘框的偏移





IoU - 交并比

- IoU 用来计算两个框之间的相似度
 - 0 表示无重叠，1 表示重合
- 这是 Jacquard 指数的一个特殊情况
 - 给定两个集合 A 和 B



$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$



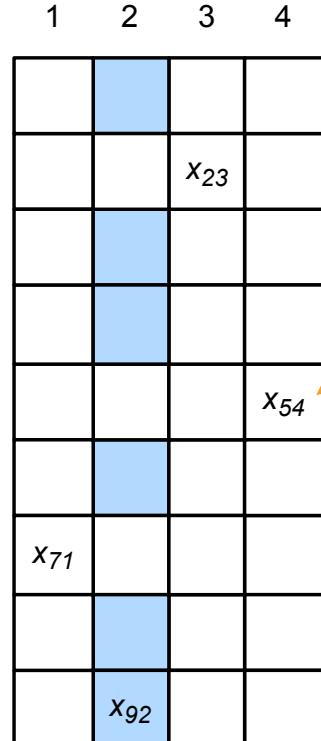
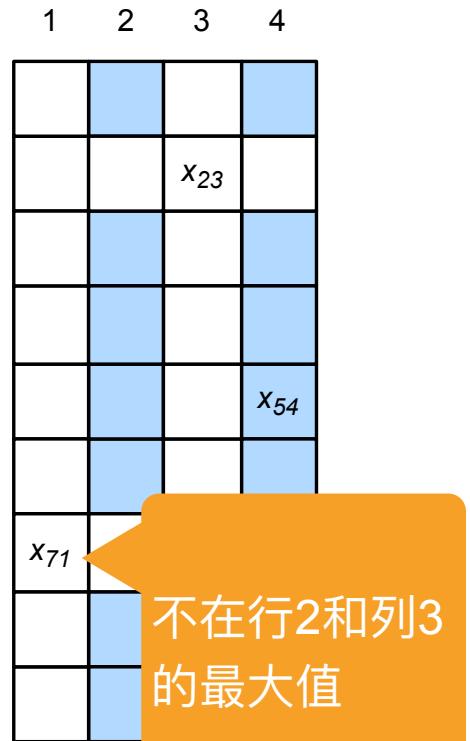
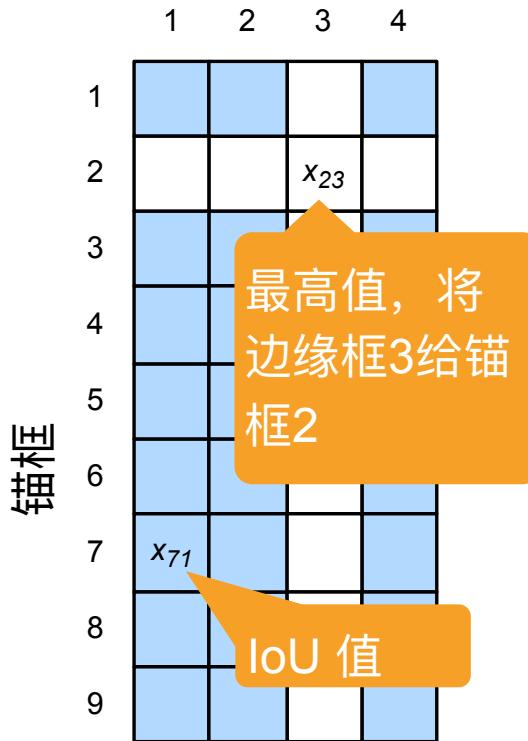
赋予锚框标号

- 每个锚框是一个训练样本
- 将每个锚框，要么标注成背景，要么关联上一个真实边缘框
- 我们可能会生成大量的锚框
 - 这个导致大量的负类样本



赋予锚框标号

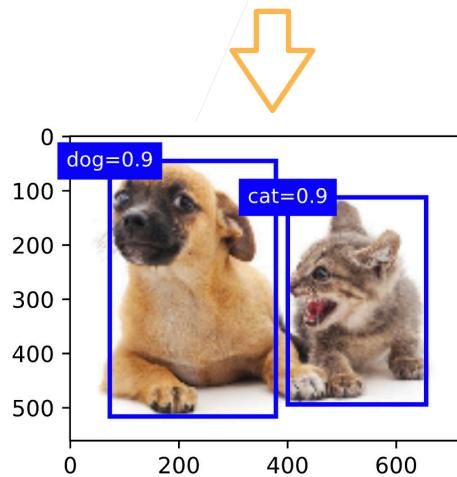
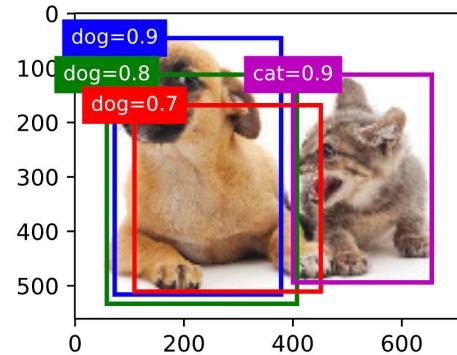
边缘框





使用非极大值抑制 (NMS) 输出

- 每个锚框预测一个边缘框
- NMS可以合并相似的预测
 - 选中是非背景类的最大预测值
 - 去掉所有其它和它IoU值大于 θ 的预测
 - 重复上述过程直到所有预测要么被选中，要么被去掉





总结

- 一类目标检测算法基于锚框来预测
- 首先生成大量锚框，并赋予标号，每个锚框作为一个样本进行训练
- 在预测时，使用NMS来去掉冗余的预测

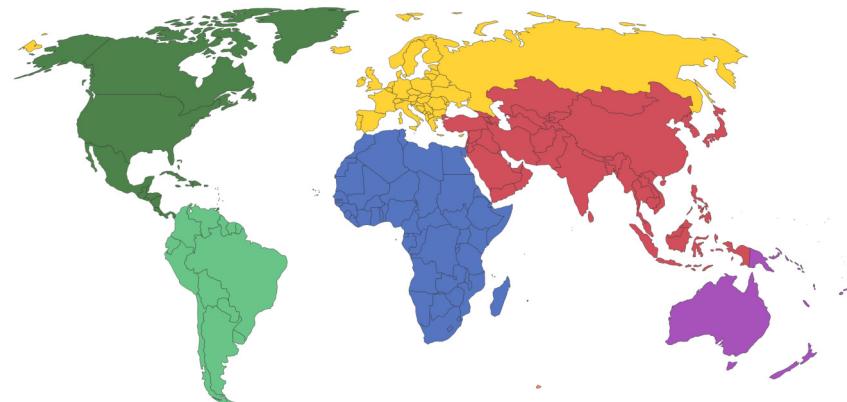


区域卷积神经网络

Continents according to Our World in Data

This is how continents are defined on Our World in Data

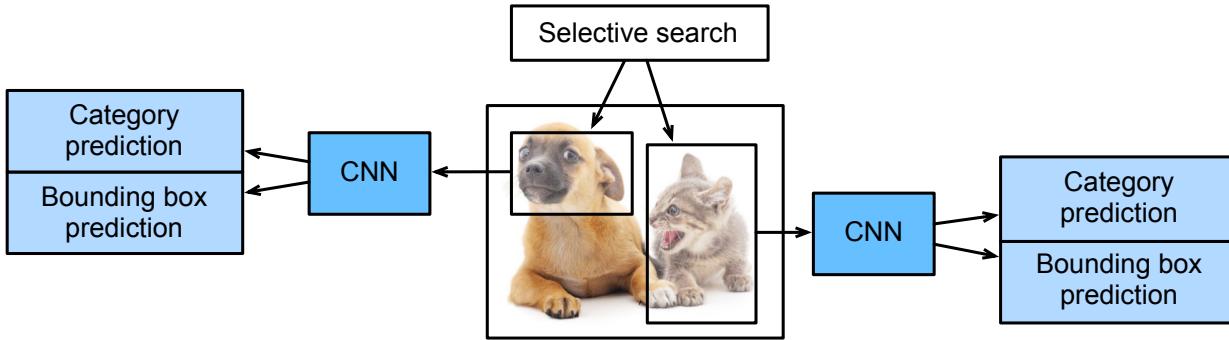
Our World
in Data



■ Africa ■ Asia ■ Europe ■ North America ■ Oceania ■ South America

CC BY

R-CNN



- 使用启发式搜索算法来选择锚框
- 使用预训练模型来对每个锚框抽取特征
- 训练一个SVM来对类别分类
- 训练一个线性回归模型来预测边缘框偏移



兴趣区域 (RoI) 池化层

- 给定一个锚框，均匀分割成 $n \times m$ 块，输出每块里的最大值
- 不管锚框多大，总是输出 nm 个值

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

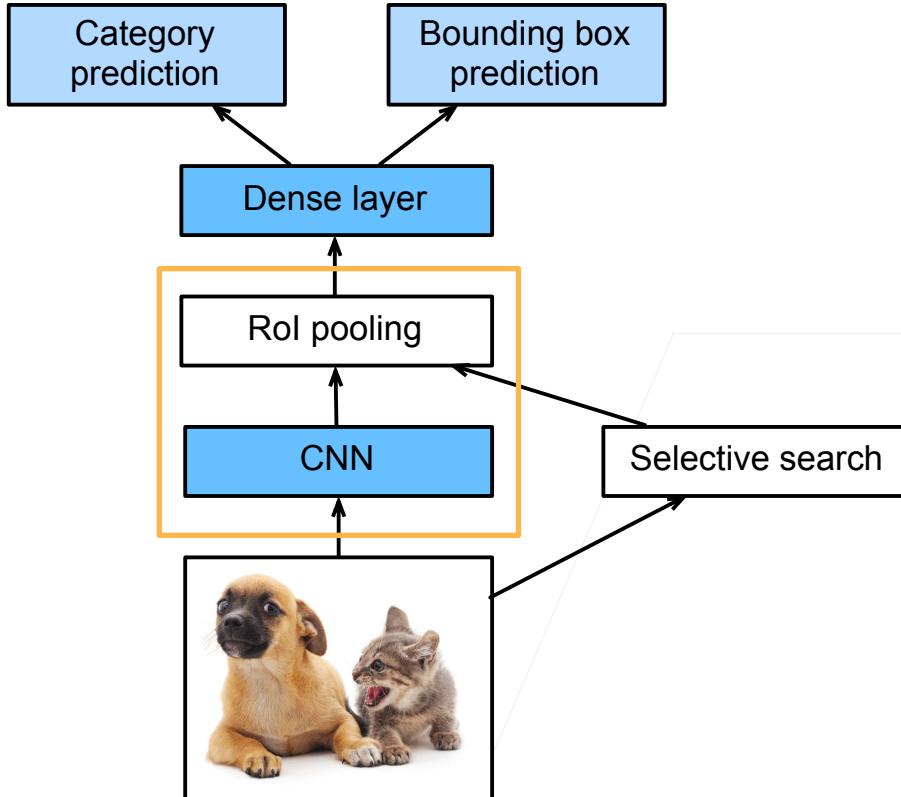
2 x 2 RoI
Pooling

5	6
9	10



Fast RCNN

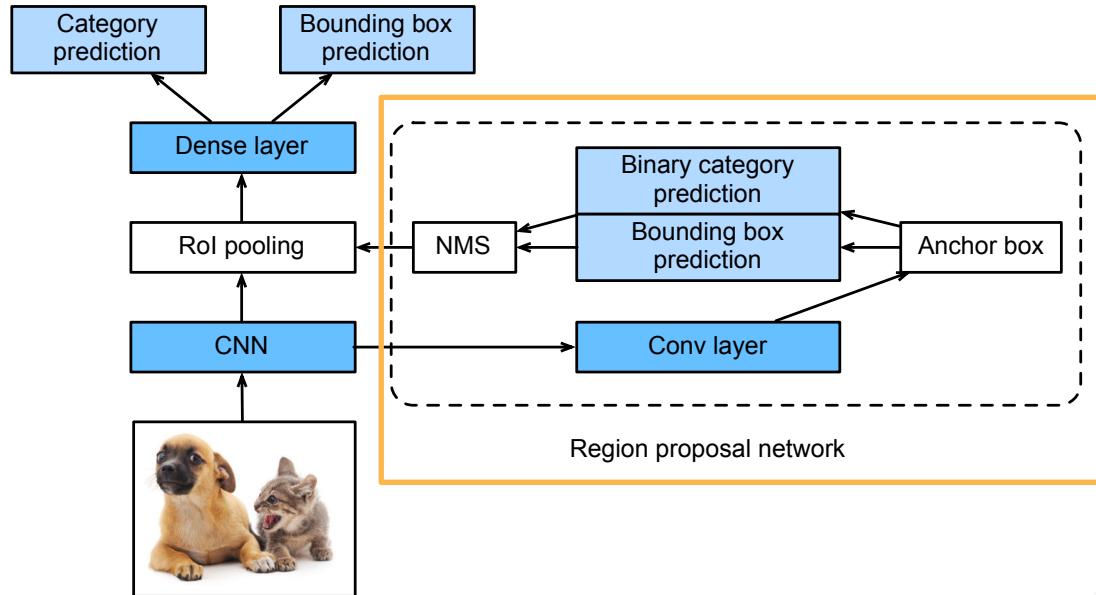
- 使用CNN对图片抽取特征
- 使用RoI池化层对每个锚框生成固定长度特征





Faster R-CNN

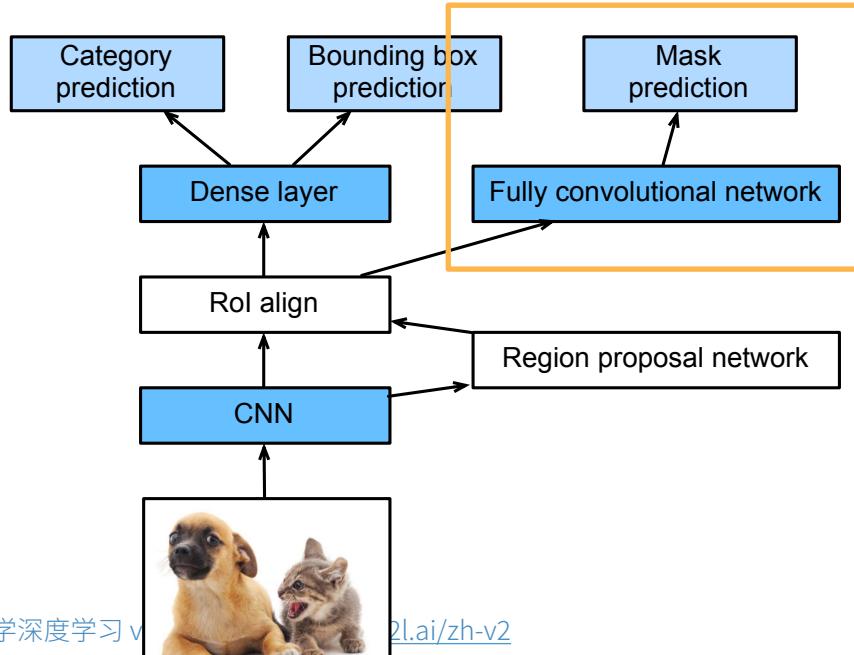
- 使用一个区域提议网络来替代启发式搜索来获得更好的锚框



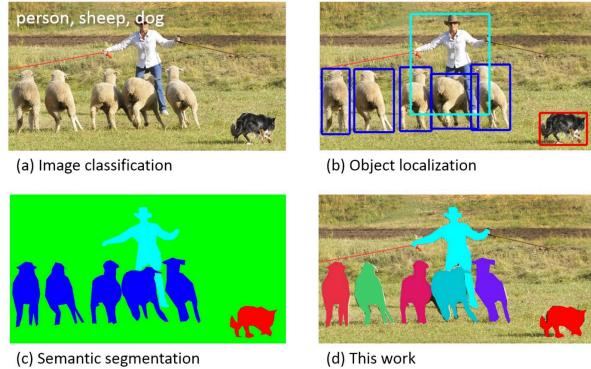


Mask R-CNN

- 如果有像素级别的标号，使用FCN来利用这些信息

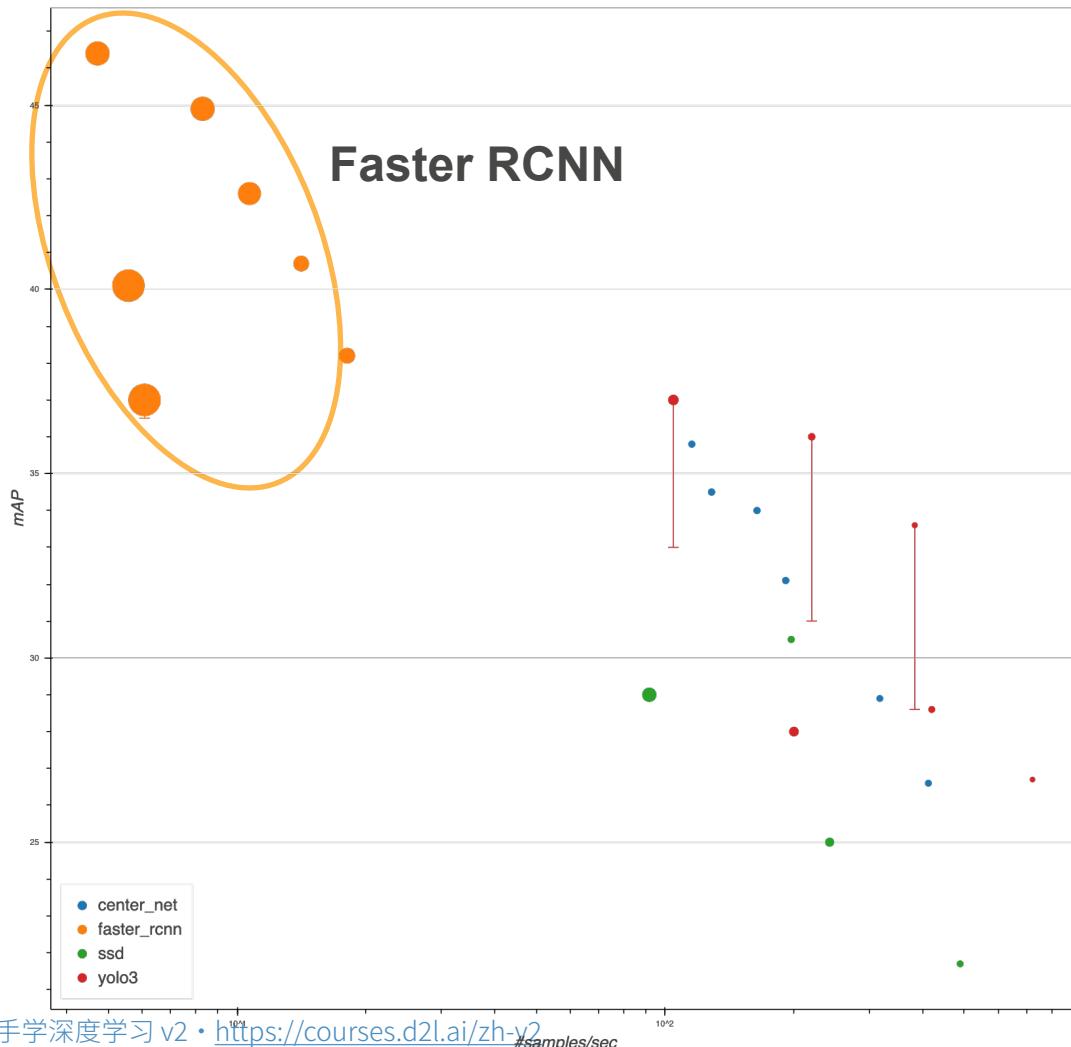


COCO





[https://cv.gluon.ai/
model_zoo/
detection.html](https://cv.gluon.ai/model_zoo/detection.html)





总结

- R-CNN 是最早、也是最有名的一类基于锚框和CNN的目标检测算法
- Fast/Faster R-CNN持续提升性能
- Faster R-CNN 和 Mask R-CNN是在最求高精度场景下的常用算法

动手学深度学习 v2

李沐 · AWS



单发多框检测 (SSD)

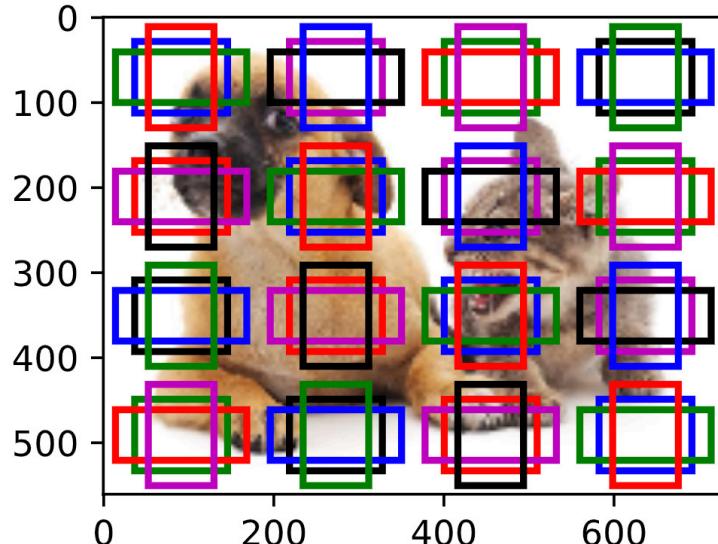




生成锚框

- 对每个像素，生成多个以它为中心的锚框
- 给定 n 个大小 s_1, \dots, s_n 和 m 个高宽比，那么生成 $n + m - 1$ 个锚框，其大小和高宽比分别为：

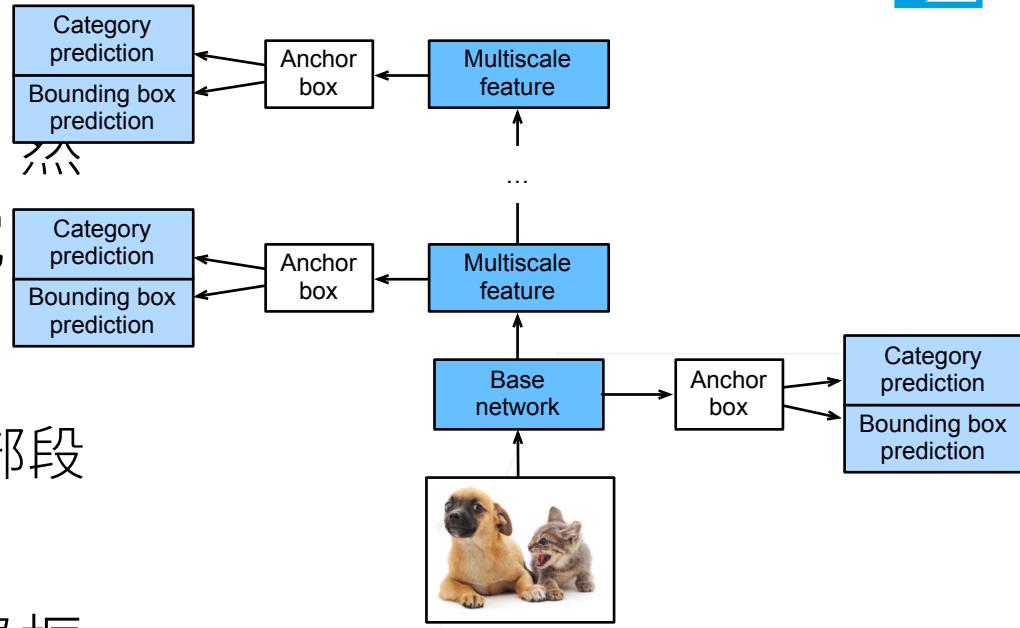
$$(s_1, r_1), (s_2, r_1), \dots, (s_n, r_1), (s_1, r_2), \dots, (s_1, r_m)$$

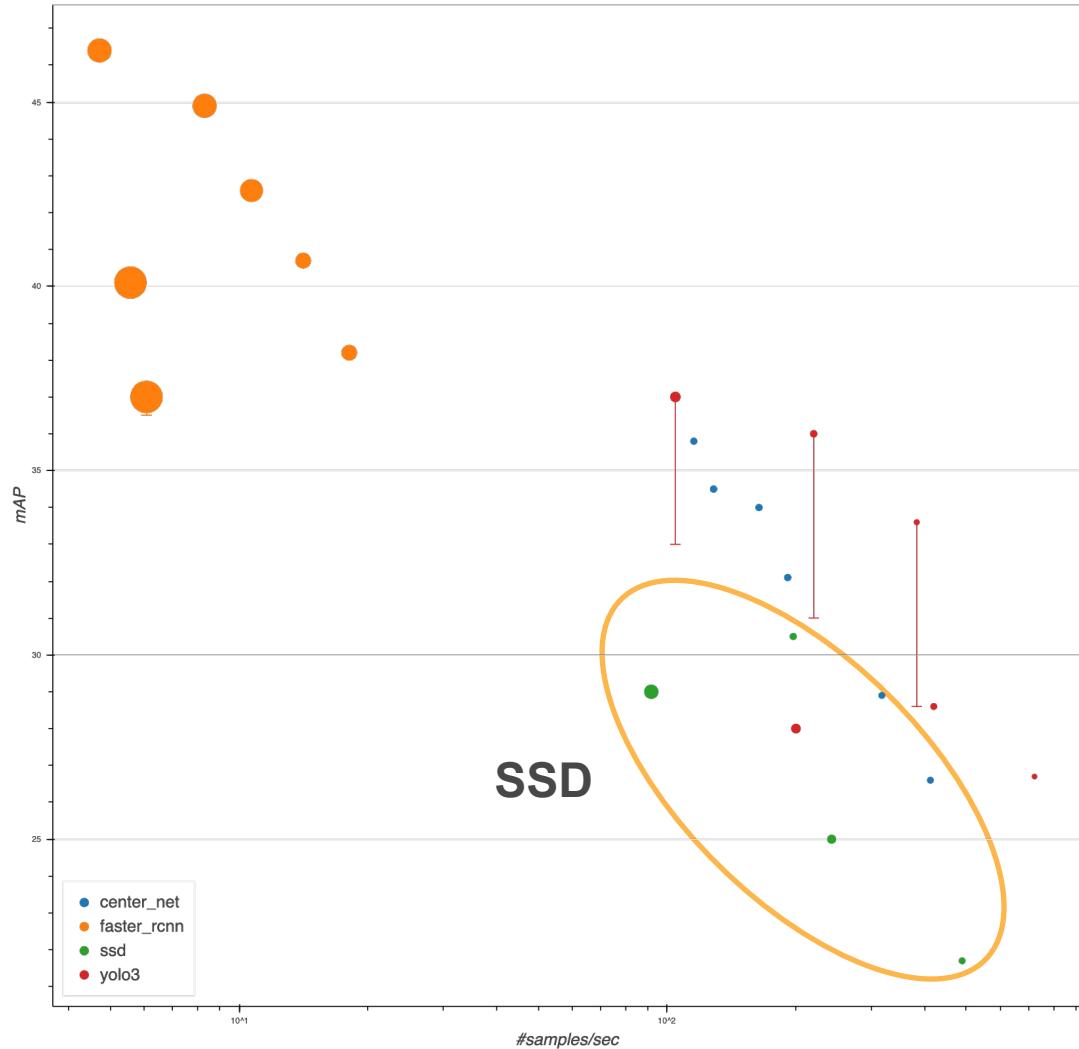




SSD 模型

- 一个基础网络来抽取特征，然后多个卷积层块来减半高宽
- 在每段都生成锚框
 - 底部段来拟合小物体，顶部段来拟合大物体
- 对每个锚框预测类别和边缘框





动手学深度学习 v2
[https://gluon-
cv.mxnet.io/](https://gluon-cv.mxnet.io/) 李洪 • AWS
[model_zoo/
detection.html](#)





总结

- SSD通过单神经网络来检测模型
- 以每个像素为中心的产生多个锚框
- 在多个段的输出上进行多尺度的检测



KEEP
CALM
BECAUSE
#YOLO

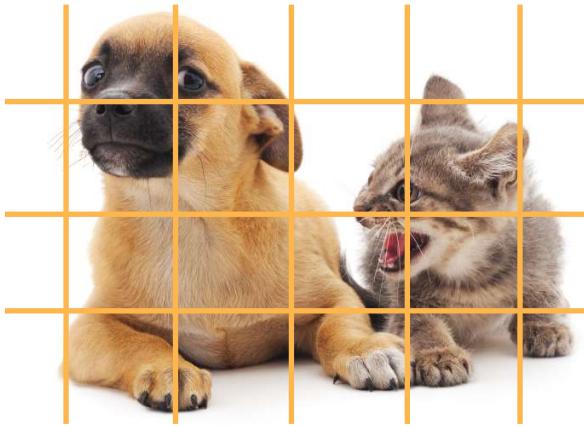
动手学深度学习 v2
李沐 · AWS

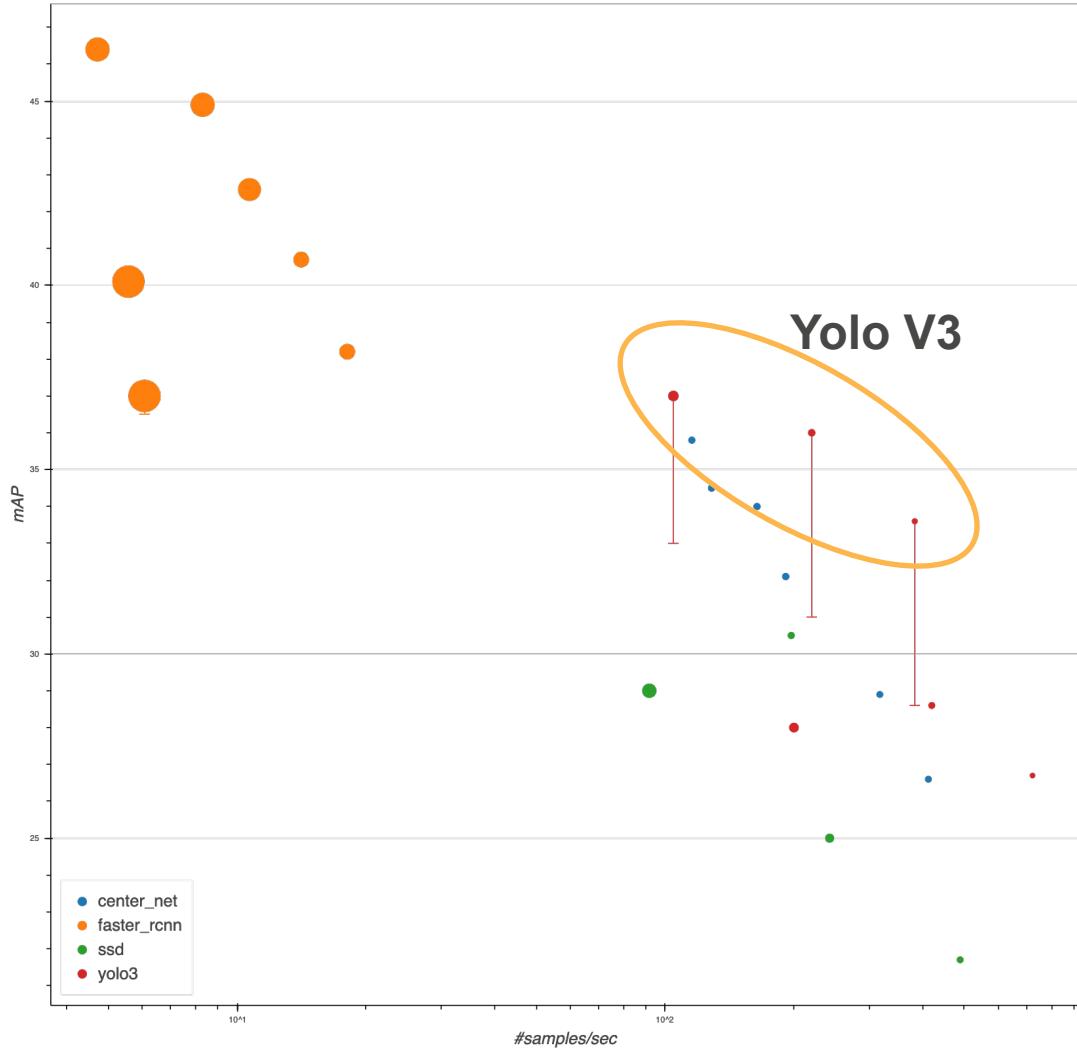




YOLO（你只看一次）

- SSD中锚框大量重叠，因此浪费了很多计算
- YOLO 将图片均匀分成 $S \times S$ 个锚框
- 每个锚框预测 B 个边缘框
- 后续版本（V2,V3,V4...）有持续改进



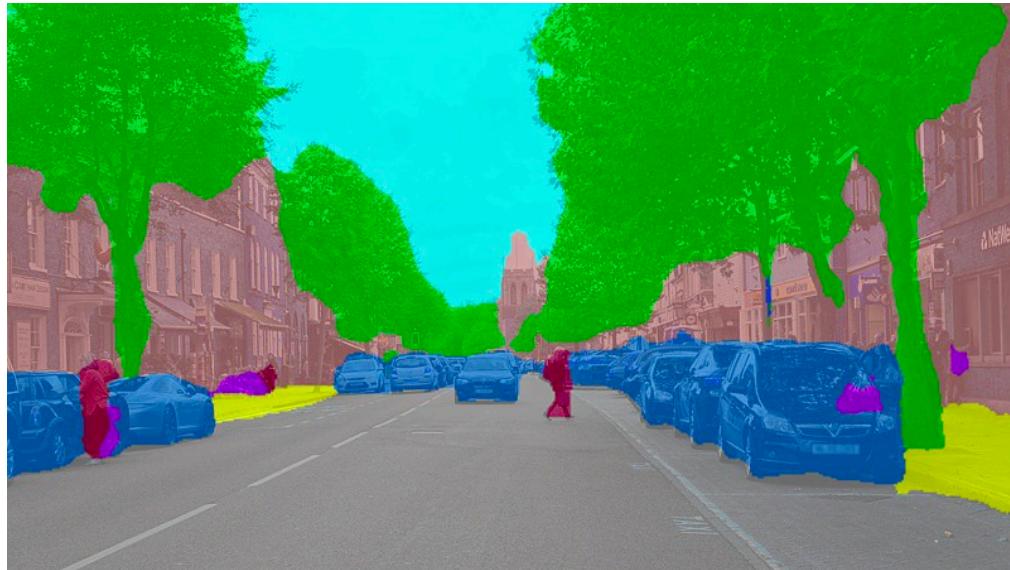


动手学深度学习 v2
[https://gluon-
cv.mxnet.io/](https://gluon-cv.mxnet.io/) 李洪 • AWS
[model_zoo/
detection.html](#)





语义分割





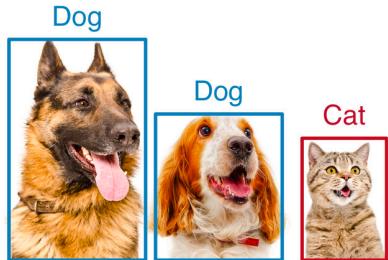
语义分割

- 语义分割将图片中的每个像素分类到对应的类别

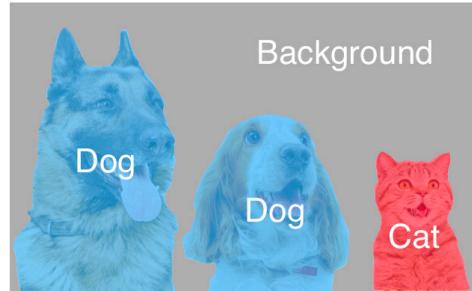
图片分类



目标检测



语义分割





应用：背景虚化





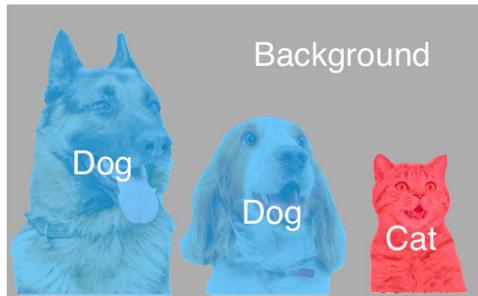
应用：路面分割



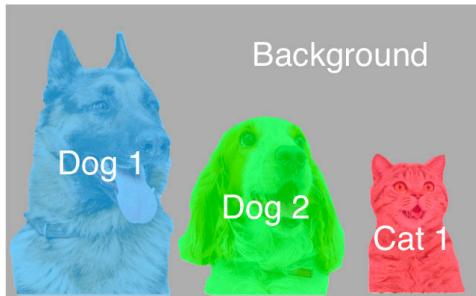


vs 实例分割

语义分割



实例分割





转置卷积

动手学深度学习 v2

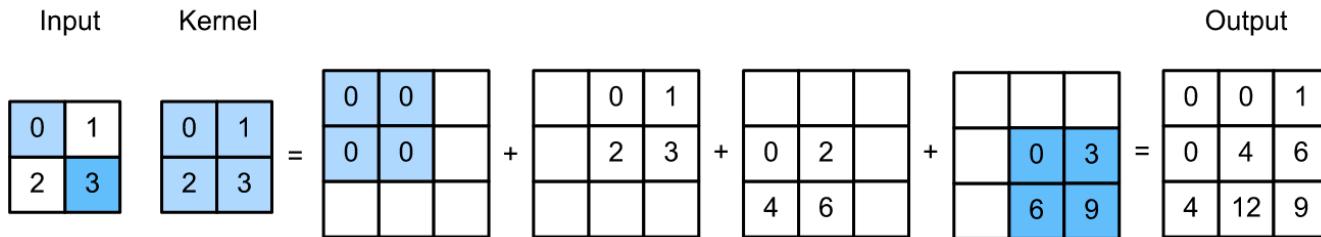
李沐 · AWS





转置卷积

- 卷积不会增大输入的高宽，通常要么不变、要么减半
- 转置卷积则可以用来增大输入高宽



$$Y[i : i + h, j : j + w] += X[i, j] \cdot K$$



为什么称之为“转置”

- 对于卷积 $Y = X \star W$
 - 可以对 W 构造一个 V , 使得卷积等价于矩阵乘法 $Y' = VX'$
 - 这里 Y', X' 是 Y, X 对应的向量版本
- 转置卷积则等价于 $Y' = V^T X'$
- 如果卷积将输入从 (h, w) 变成了 (h', w')
 - 同样超参数的转置卷积则从 (h', w') 变成 (h, w)



动手学深度学习 v2

李沐 · AWS



再谈转置卷积



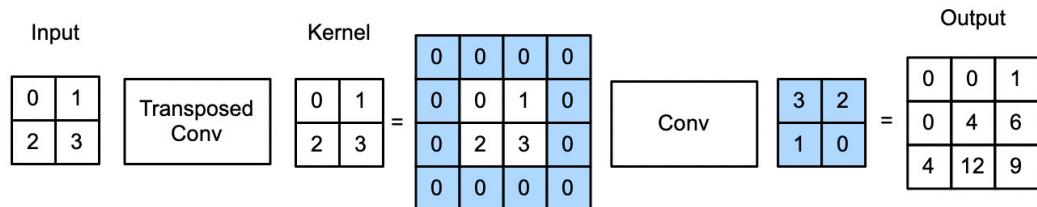
转置卷积

- 转置卷积是一种卷积
 - 它将输入和核进行了重新排列
 - 同卷积一般是做下采样不同，它通常用作上采样
 - 如果卷积将输入从 (h, w) 变成了 (h', w') ，同样超参数下它将 (h', w') 变成 (h, w)



重新排列输入和核

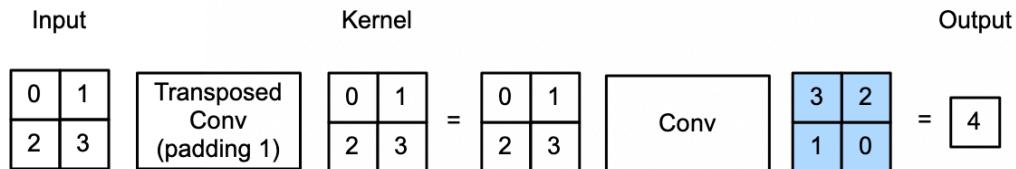
- 当填充为0步幅为1时
 - 将输入填充 $k - 1$ (k 是核窗口)
 - 将核矩阵上下、左右翻转
 - 然后做正常卷积 (填充0、步幅1)





重新排列输入和核

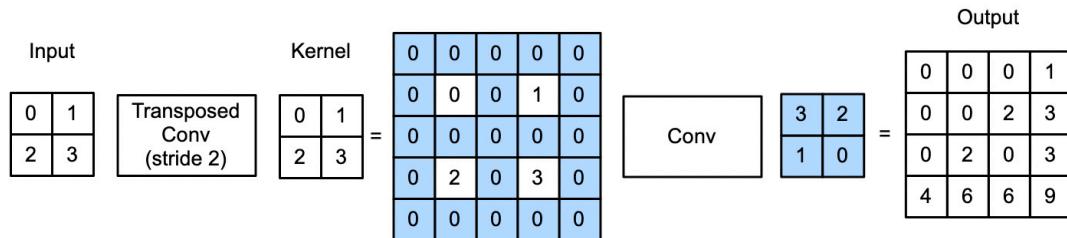
- 当填充为 p 步幅为 1 时
 - 将输入填充 $k - p - 1$ (k 是核窗口)
 - 将核矩阵上下、左右翻转
 - 然后做正常卷积（填充 0、步幅 1）





重新排列输入和核

- 当填充为 p 步幅为 s 时
 - 在行和列之间插入 $s - 1$ 行或列
 - 将输入填充 $k - p - 1$ (k 是核窗口)
 - 将核矩阵上下、左右翻转
 - 然后做正常卷积（填充0、步幅1）





形状换算

- 输入高（宽）为 n , 核 k , 填充 p , 步幅 s
- 转置卷积: $n' = sn + k - 2p - s$
 - 卷积: $n' = \lfloor (n - k - 2p + s)/s \rfloor \rightarrow n \geq sn' + k - 2p - s$
 - 如果让高宽成倍增加, 那么 $k = 2p + s$



同反卷积的关系

- 数学上的反卷积 (deconvolution) 是指卷积的逆运算
 - 如果 $Y = \text{conv}(X, K)$ ，那么 $X = \text{deconv}(Y, K)$
- 反卷积很少用在深度学习中
 - 我们说的反卷积神经网络指用了转置卷积的神经网络

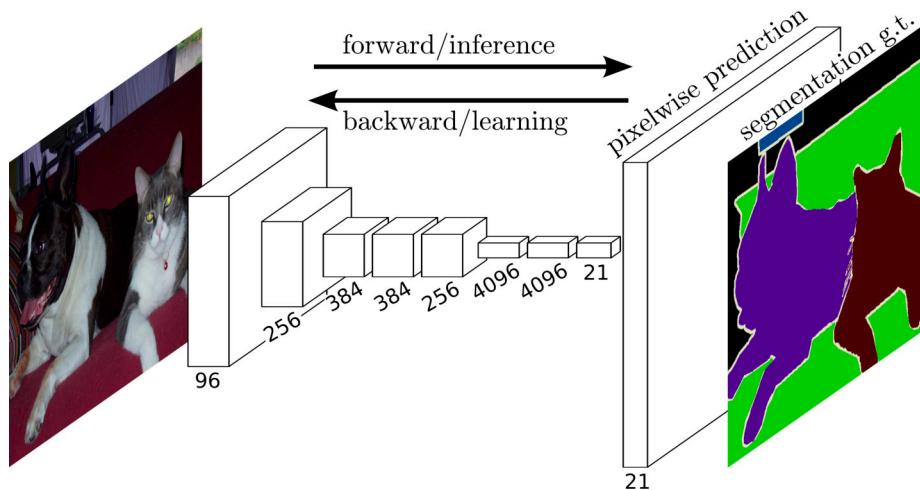


总结

- 转置卷积是一种变化了输入和核的卷积，来得到上采用的目的
- 不等同于数学上的反卷积操作



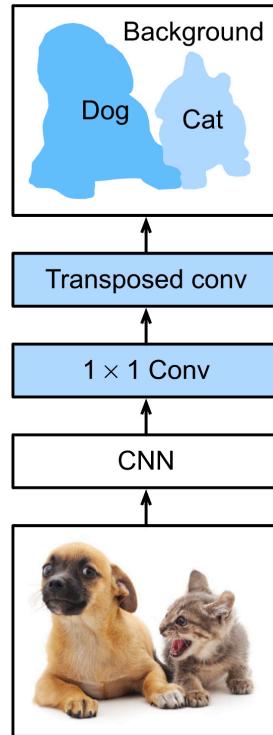
全连接卷积神经网络 (FCN)





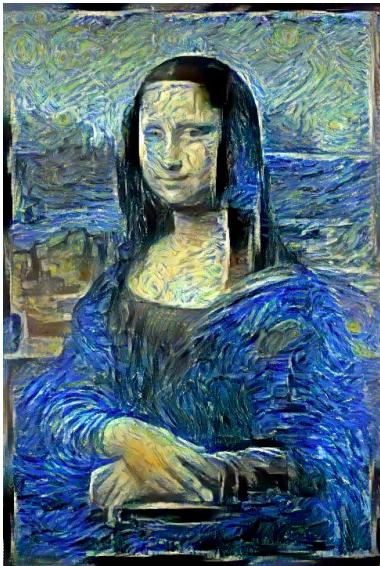
全连接卷积神经网络 (FCN)

- FCN是用来做语义分割的最早的深度神经网络之一
- 它替换用转置卷积层来替换CNN最后的全连接层
 - 从而可以实现每个像素的预测





样式迁移





样式迁移

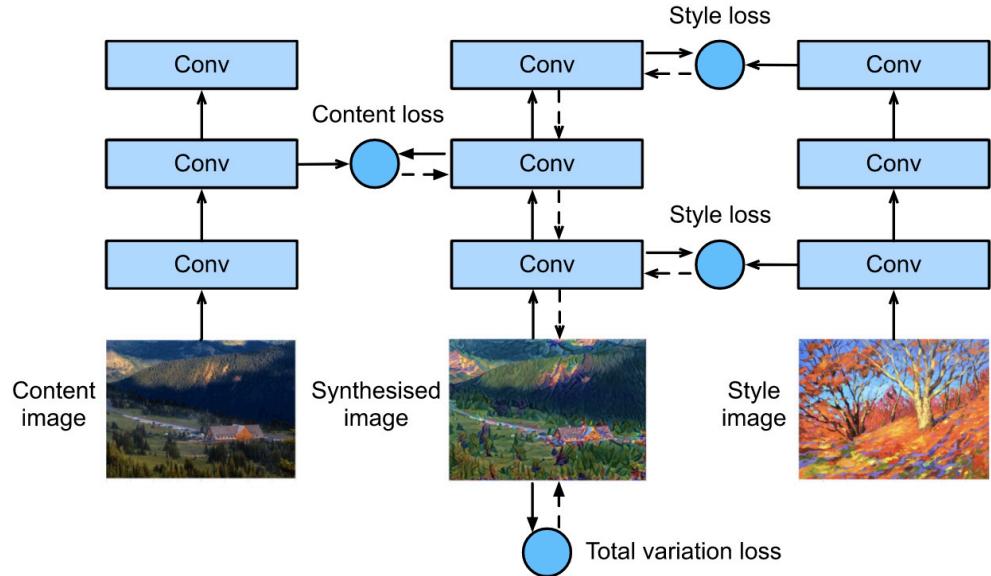
- 将样式图片中的样式迁移到内容图片上，得到合成图片





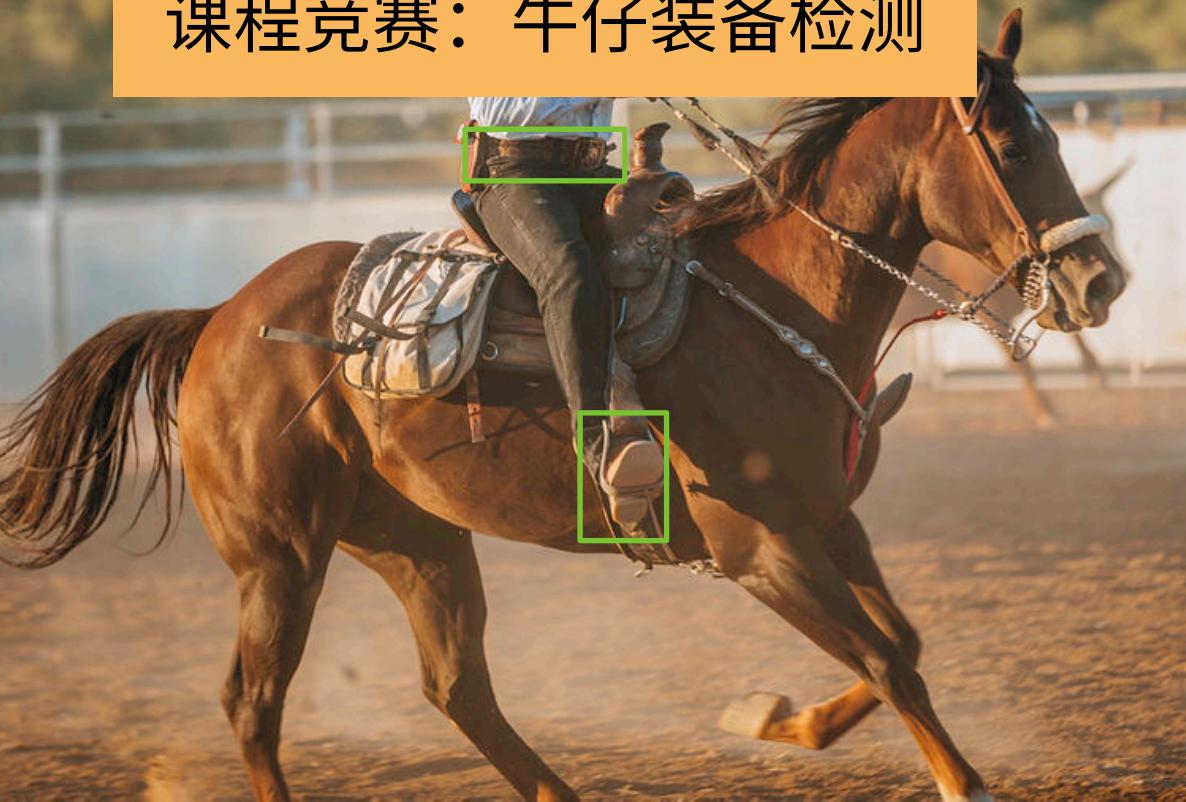
基于CNN的样式迁移

- 奠基性工作





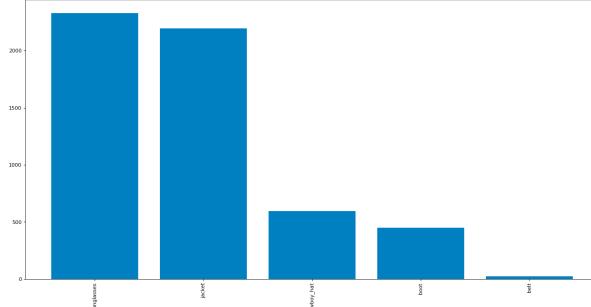
课程竞赛：牛仔装备检测





任务

- 检测牛仔夹克、墨镜、靴子、牛仔帽、腰带
- 6937张训练图片，12660标注框
- 数据使用MS-COCO格式，评测使用mAP
 - 均可直接调用 pycocotools
- 挑战：
类别不平衡





安排

- Kaggle地址：<https://www.kaggle.com/c/cowboyoutfits/>
 - 结果提交请用：<https://competitions.codalab.org/competitions/33573>
- 取得前10名的队伍，并在Kaggle提交notebook将获得签名书
- 时间为三周
 - 公榜：现在—8月6日晚12点
 - 私榜：公榜结束—8月7日中午12点

动手学深度学习 v2

李沐 · AWS



序列模型



Riccardo Vasapolli
Photography



序列数据

- 实际中很多数据是有时序结构的
- 电影的评价随时间变化而变化
 - 拿奖后评分上升，直到奖项被忘记
 - 看了很多好电影后，人们的期望变高
 - 季节性：贺岁片、暑期档
 - 导演、演员的负面报道导致评分变低



序列数据 - 更多例子

- 音乐、语言、文本、和视频都是连续的
 - 标题“狗咬人”远没有“人咬狗”那么令人惊讶
- 大地震发生后，很可能会有几次较小的余震
- 人的互动是连续的，从网上吵架可以看出
- 预测明天的股价要比填补昨天遗失的股价的更困难



统计工具

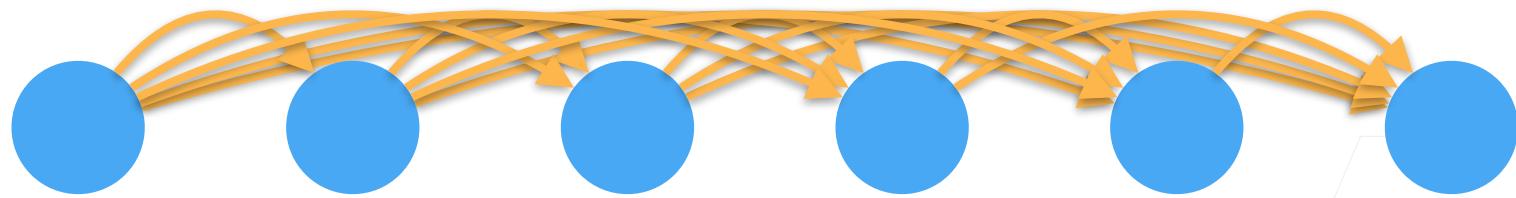
- 在时间 t 观察到 x_t , 那么得到 T 个不独立的随机变量
 $(x_1, \dots, x_T) \sim p(\mathbf{x})$
- 使用条件概率展开

$$p(a, b) = p(a)p(b | a) = p(b)p(a | b)$$

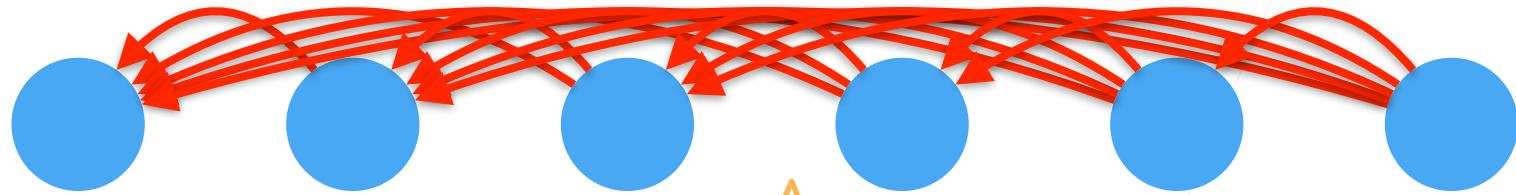


统计工具

$$p(\mathbf{x}) = p(x_1) \cdot p(x_2 | x_1) \cdot p(x_3 | x_1, x_2) \cdot \dots \cdot p(x_T | x_1, \dots, x_{T-1})$$



$$p(\mathbf{x}) = p(x_T) \cdot p(x_{T-1} | x_T) \cdot p(x_{T-2} | x_{T-1}, x_T) \cdot \dots \cdot p(x_1 | x_2, \dots, x_T)$$



物理上不一定可行



序列模型

$$p(\mathbf{x}) = p(x_1) \cdot p(x_2 | x_1) \cdot p(x_3 | x_1, x_2) \cdot \dots \cdot p(x_T | x_1, \dots, x_{T-1})$$



- 对条件概率建模

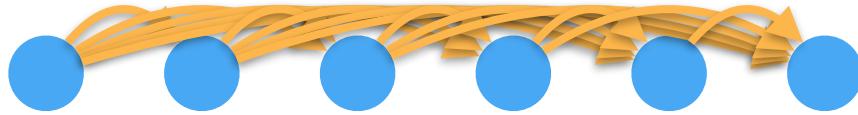
$$p(x_t | x_1, \dots, x_{t-1}) = p(x_t | f(x_1, \dots, x_{t-1}))$$

对见过的数据建模，也称
自回归模型

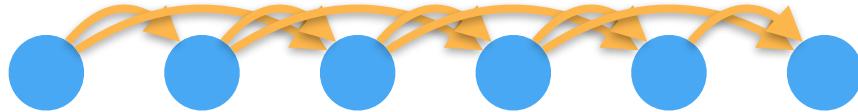


方案 A - 马尔科夫假设

$$p(\mathbf{x}) = p(x_1) \cdot p(x_2 | x_1) \cdot p(x_3 | x_1, x_2) \cdot \dots \cdot p(x_T | x_1, \dots, x_{T-1})$$



- 假设当前数据只跟 τ 个过去数据点相关



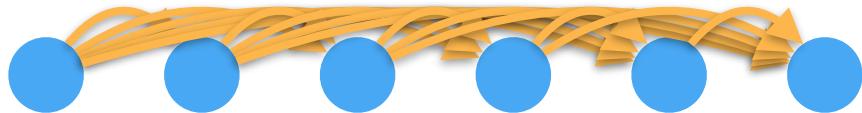
$$p(x_t | x_1, \dots, x_{t-1}) = p(x_t | x_{t-\tau}, \dots, x_{t-1}) = p(x_t | f(x_{t-\tau}, \dots, x_{t-1}))$$

例如在过去数据上训练
一个MLP模型

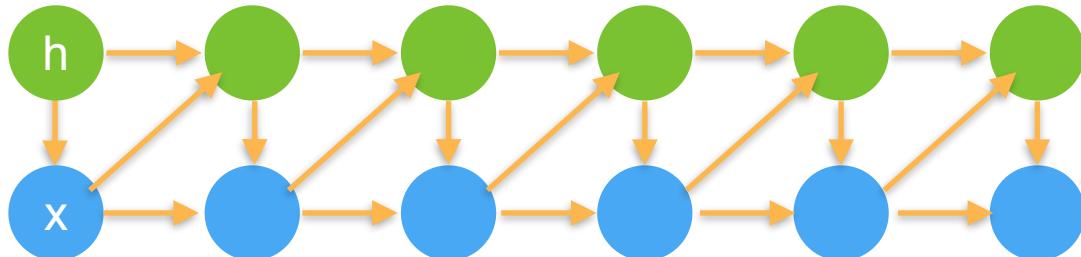


方案 B - 潜变量模型

$$p(\mathbf{x}) = p(x_1) \cdot p(x_2 | x_1) \cdot p(x_3 | x_1, x_2) \cdot \dots \cdot p(x_T | x_1, \dots, x_{T-1})$$



- 引入潜变量 h_t 来表示过去信息 $h_t = f(x_1, \dots, x_{t-1})$
 - 这样 $x_t = p(x_t | h_t)$





总结

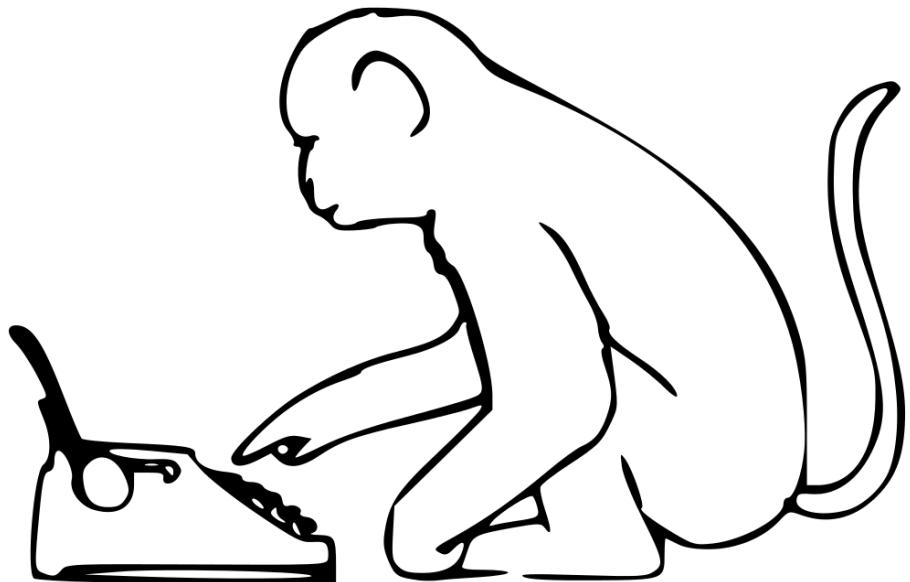
- 时序模型中，当前数据跟之前观察到的数据相关
- 自回归模型使用自身过去数据来预测未来
- 马尔科夫模型假设当前只跟最近少数数据相关，从而简化模型
- 潜变量模型使用潜变量来概括历史信息

动手学深度学习 v2

李沐 · AWS



语言模型





语言模型

- 给定文本序列 x_1, \dots, x_T , 语言模型的目标是估计联合概率 $p(x_1, \dots, x_T)$
- 它的应用包括
 - 做预训练模型 (eg BERT, GPT-3)
 - 生成本文, 给定前面几个词, 不断的使用 $x_t \sim p(x_t | x_1, \dots, x_{t-1})$ 来生成后续文本
 - 判断多个序列中哪个更常见, e.g. “to recognize speech” vs “to wreck a nice beach”



使用计数来建模

- 假设序列长度为2， 我们预测

$$p(x, x') = p(x)p(x' | x) = \frac{n(x)}{n} \frac{n(x, x')}{n(x)}$$

- 这里 n 是总词数， $n(x), n(x, x')$ 是单个单词和连续单词对的出现次数
- 很容易拓展到长为3的情况

$$p(x, x', x'') = p(x)p(x' | x)p(x'' | x, x') = \frac{n(x)}{n} \frac{n(x, x')}{n(x)} \frac{n(x, x', x'')}{n(x, x')}$$



N元语法

- 当序列很长时，因为文本量不够大，很可能 $n(x_1, \dots, x_T) \leq 1$
- 使用马尔科夫假设可以缓解这个问题

$$p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2)p(x_3)p(x_4)$$

- 一元语法：

$$= \frac{n(x_1)}{n} \frac{n(x_2)}{n} \frac{n(x_3)}{n} \frac{n(x_4)}{n}$$

$$p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_3)$$

- 二元语法：

$$= \frac{n(x_1)}{n} \frac{n(x_1, x_2)}{n(x_1)} \frac{n(x_2, x_3)}{n(x_2)} \frac{n(x_3, x_4)}{n(x_3)}$$

- 三元语法： $p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)p(x_4|x_2, x_3)$



总结

- 语言模型估计文本序列的联合概率
- 使用统计方法时常采用n元语法



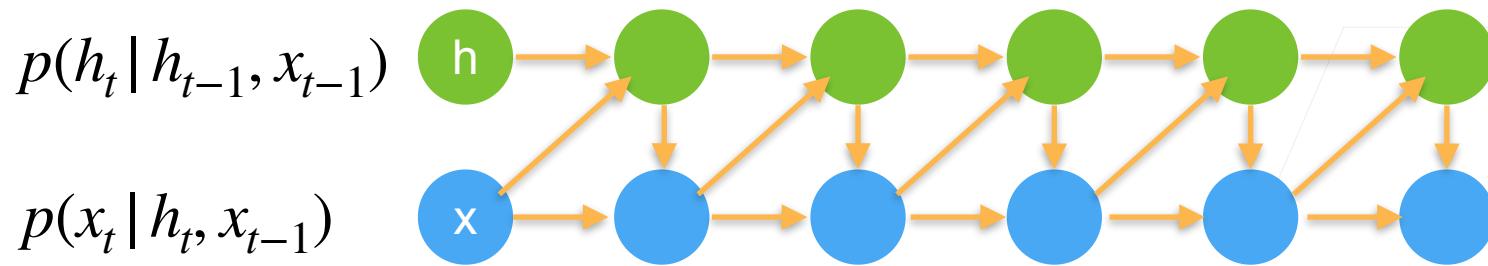
李沐 · AWS

循环神经网络



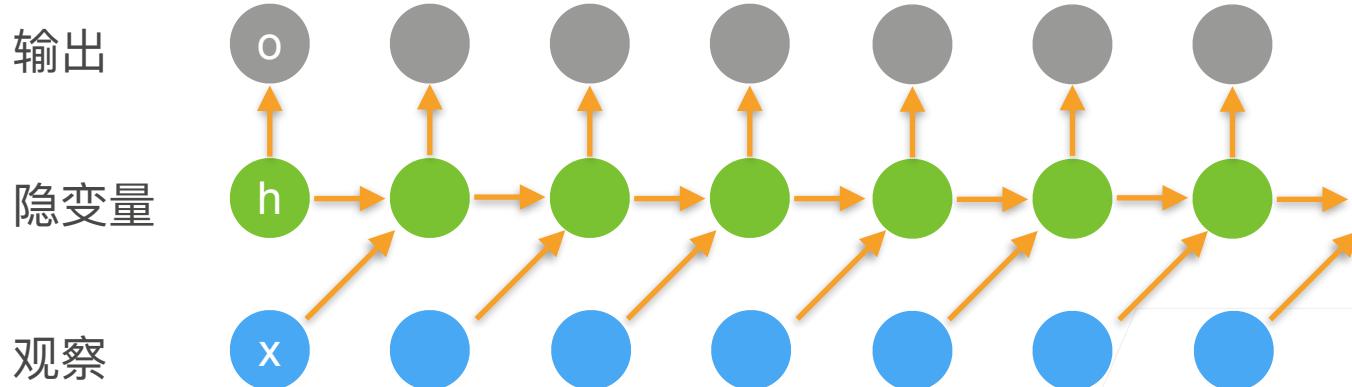
潜变量自回归模型

- 使用潜变量 h_t 总结过去信息





循环神经网络

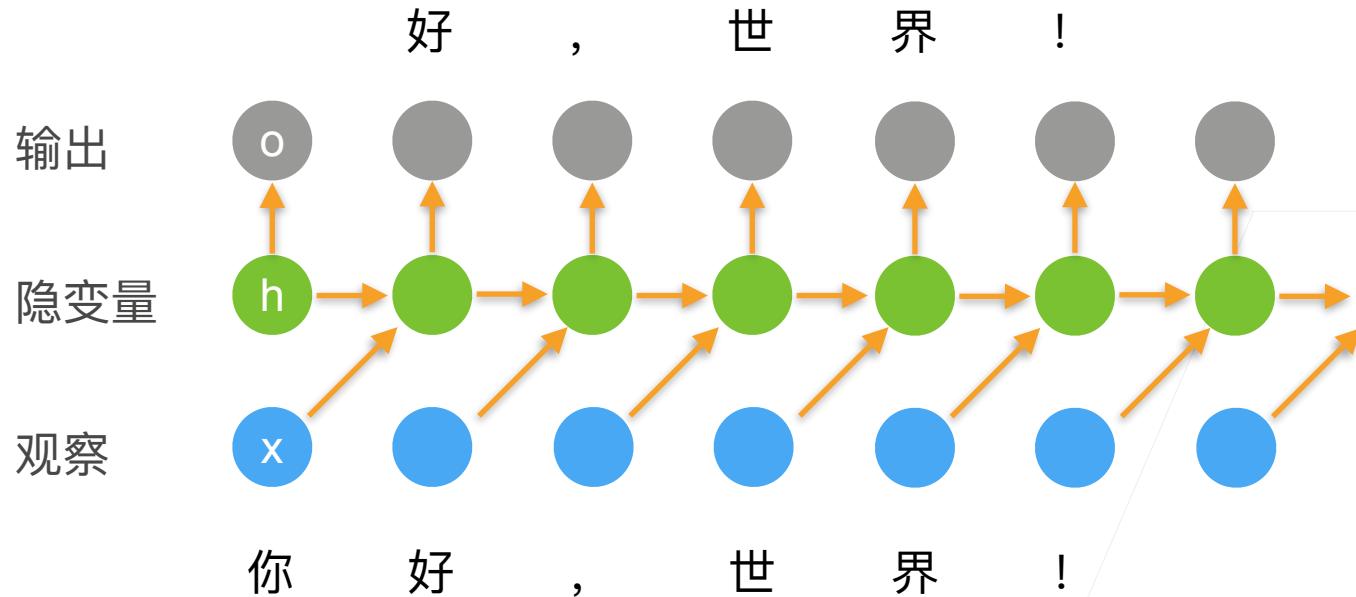


- 更新隐藏状态: $\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_{t-1} + \mathbf{b}_h)$
- 输出: $\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$

去掉这一项就
退化成MLP



使用循环神经网络的语言模型





困惑度 (perplexity)

- 衡量一个语言模型的好坏可以用平均交叉熵

$$\pi = \frac{1}{n} \sum_{i=1}^n -\log p(x_t | x_{t-1}, \dots)$$

p 是语言模型的预测概率， x_t 是真实词

- 历史原因NLP使用困惑度 $\exp(\pi)$ 来衡量，

是平均每次可能选项

- 1表示完美，无穷大是最差情况



梯度裁剪

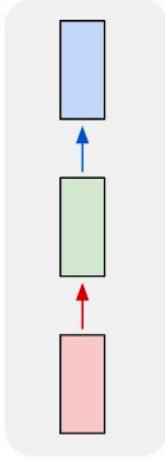
- 迭代中计算这 T 个时间步上的梯度，在反向传播过程中产生长度为 $O(T)$ 的矩阵乘法链，导致数值不稳定
- 梯度裁剪能有效预防梯度爆炸
 - 如果梯度长度超过 θ ，那么拖影回长度 θ

$$\mathbf{g} \leftarrow \min\left(1, \frac{\theta}{\|\mathbf{g}\|}\right) \mathbf{g}$$

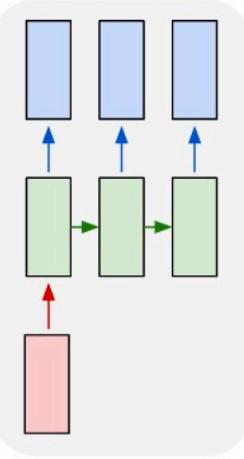


更多的应用 RNNs

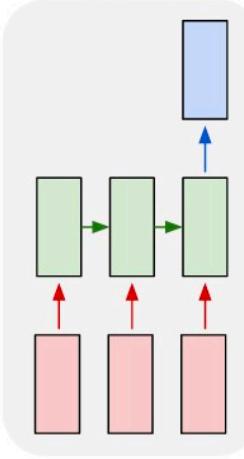
one to one



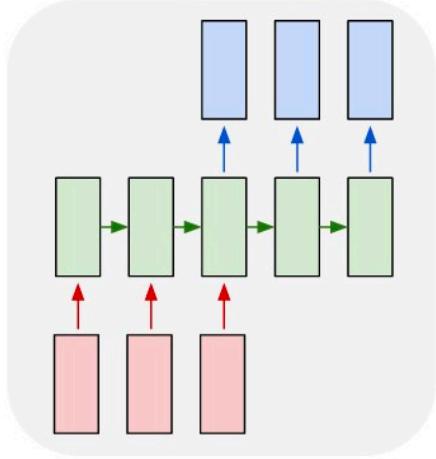
one to many



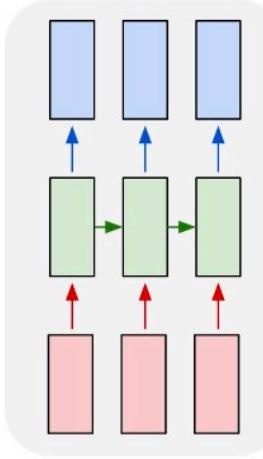
many to one



many to many



many to many



文本生成

文本分类

问答、机器翻译

Tag生成



总结

- 循环神经网络的输出取决于当下输入和前一时间的隐变量
- 应用到语言模型中时，循环神经网络根据当前词预测下一次时刻词
- 通常使用困惑度来衡量语言模型的好坏

动手学深度学习 v2

李沐 · AWS



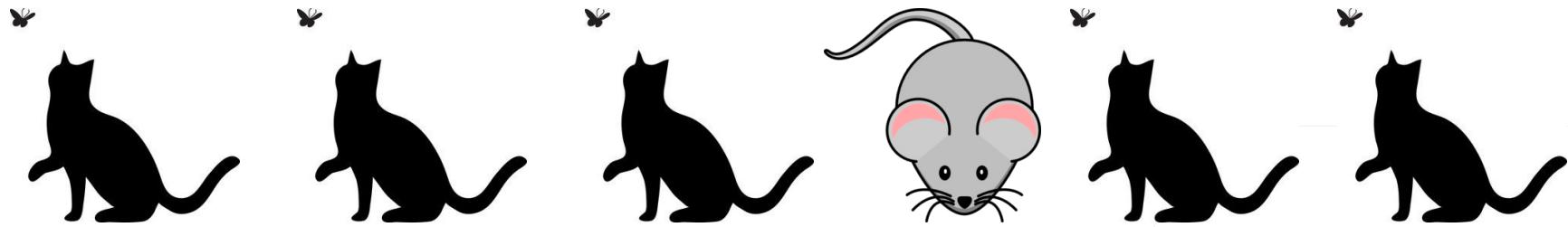
门控循环单元 GRU





关注一个序列

- 不是每个观察值都是同等重要



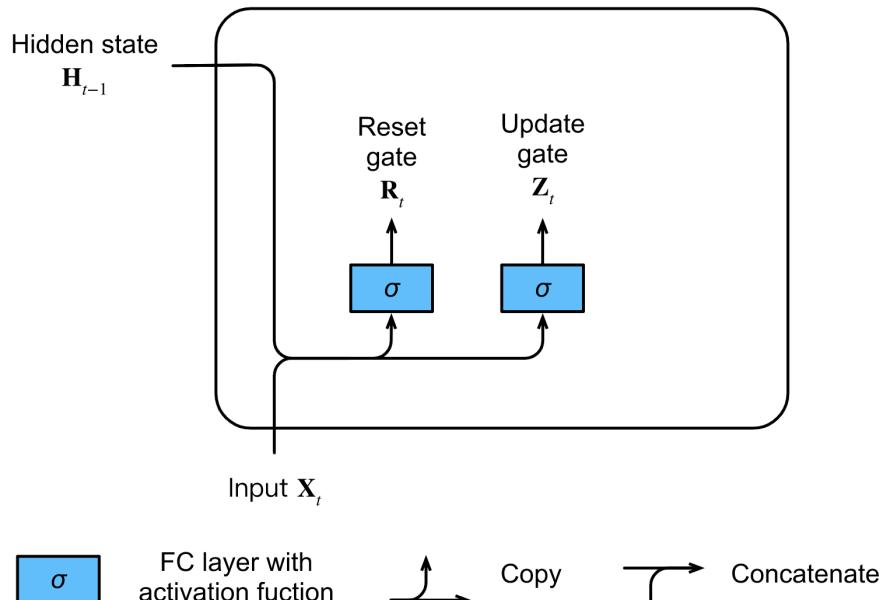
- 想只记住相关的观察需要：
 - 能关注的机制（更新门）
 - 能遗忘的机制（重置门）



门

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$

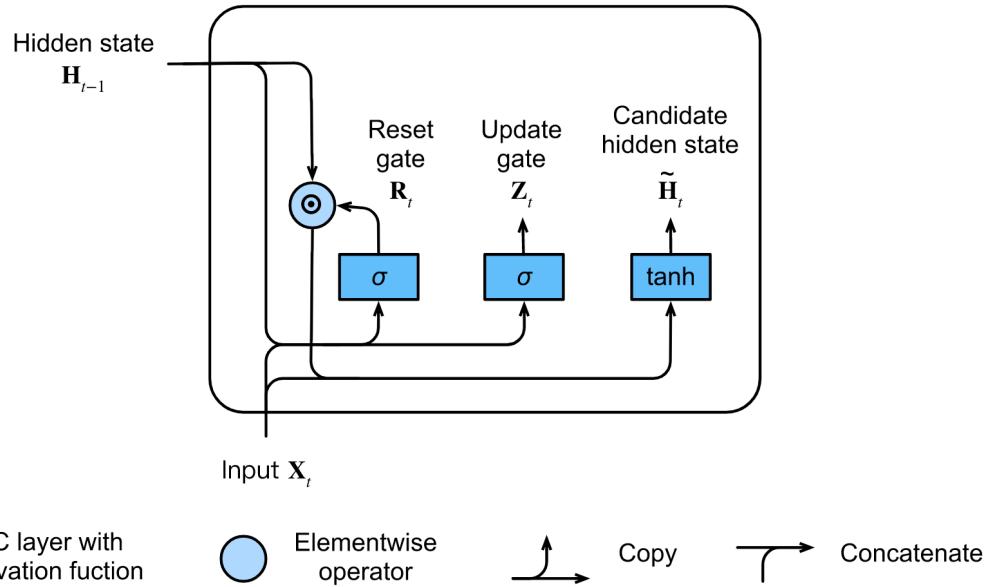
$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$





候选隐状态

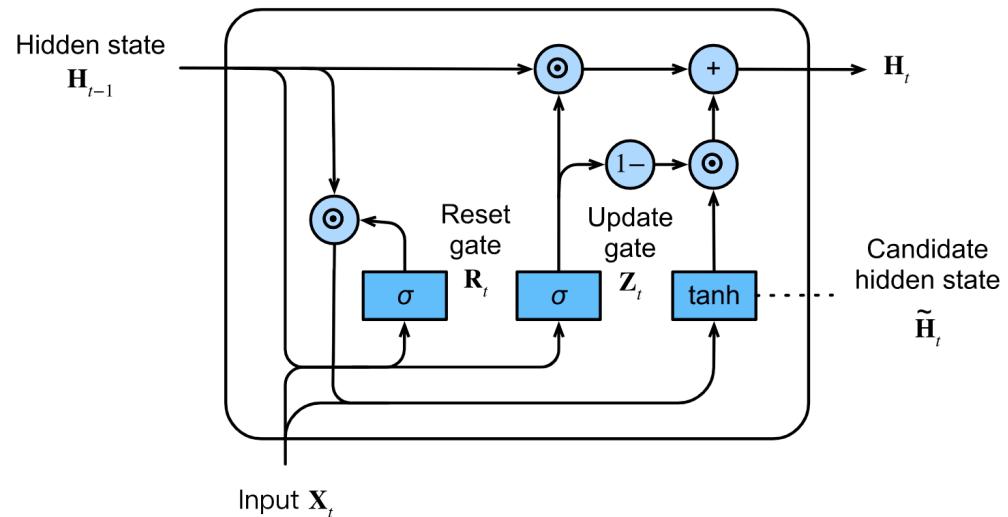
$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h)$$





隐状态

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$



FC layer with
activation function



Elementwise
operator



Copy



Concatenate



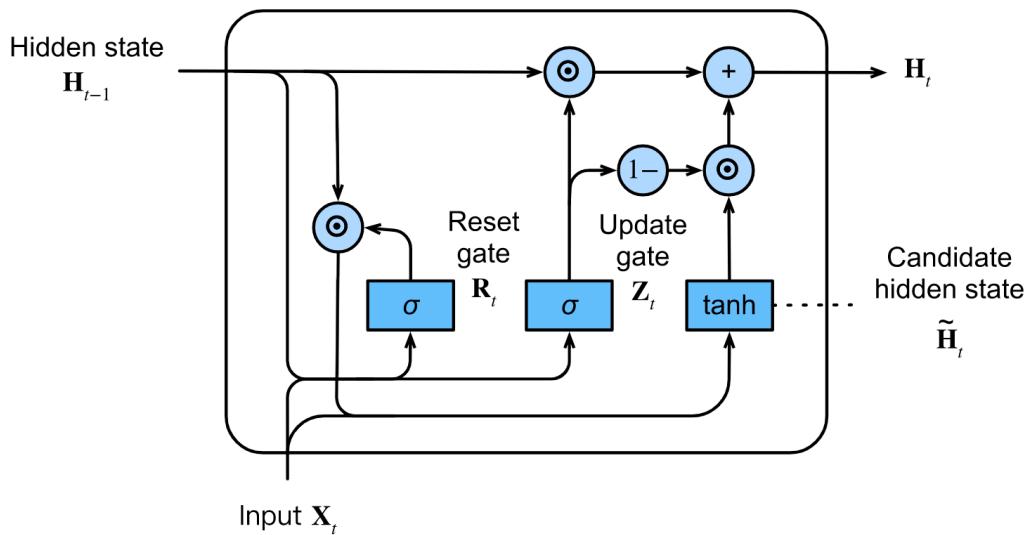
总结

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$

$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$

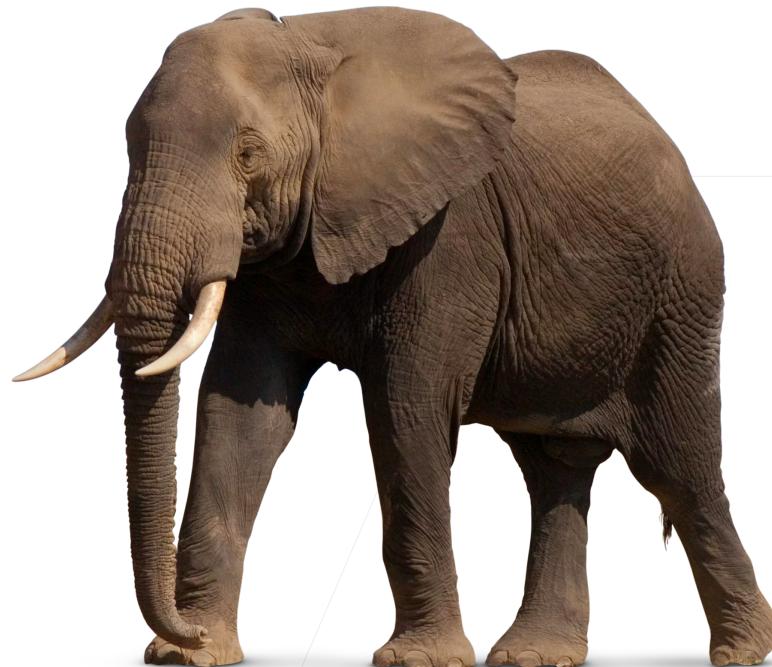
$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h)$$

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$





长短期记忆网络 (LSTM)





长短期记忆网络

- 忘记门：将值朝0减少
- 输入门：决定不是忽略掉输入数据
- 输出门：决定是不是使用隐状态

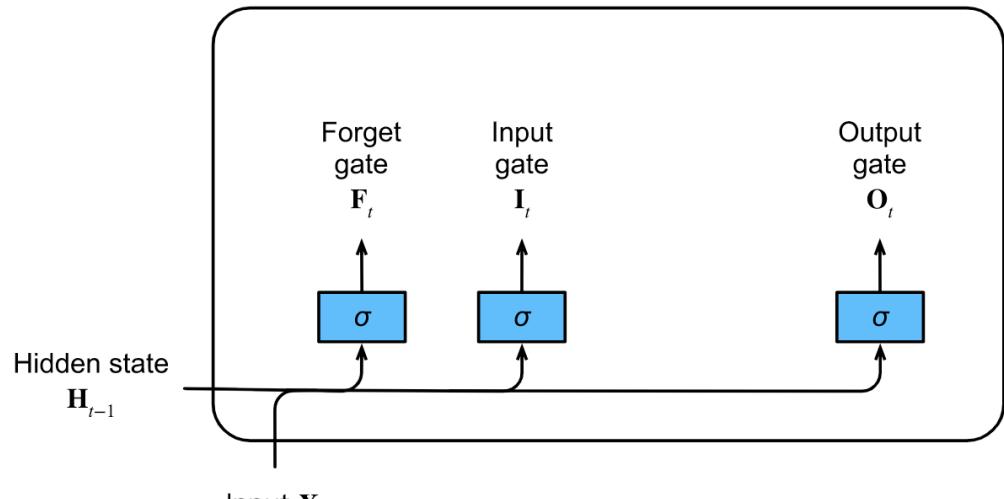


门

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$$



FC layer with
activation function



Copy

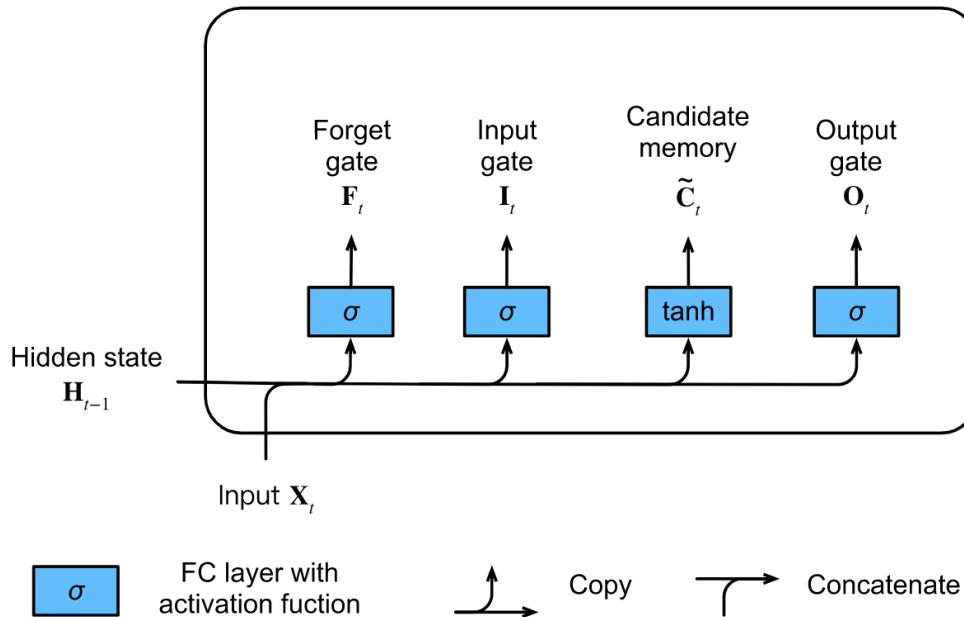


Concatenate



候选记忆单元

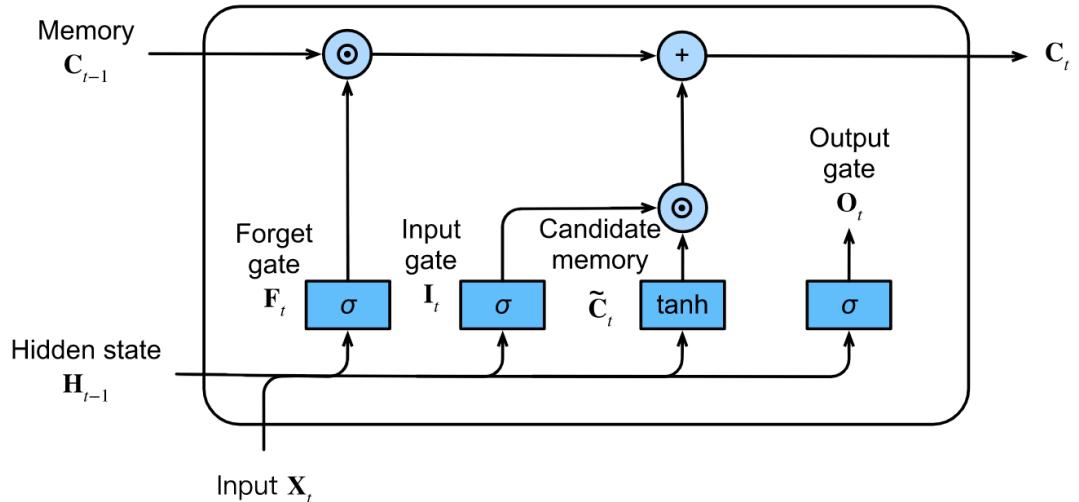
$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$





记忆单元

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$



FC layer with
activation fuction



Elementwise
operator



Copy

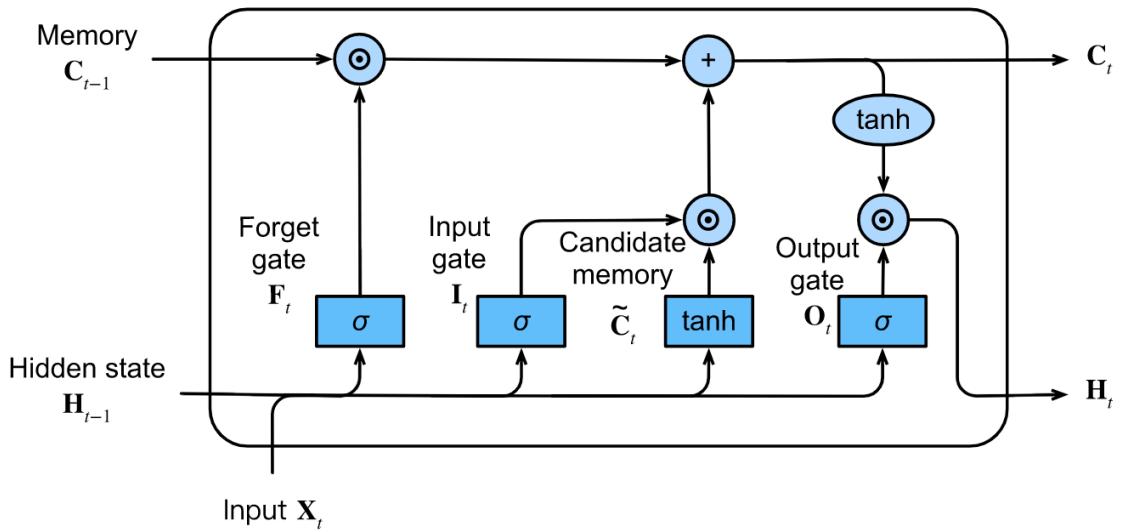


Concatenate



隐状态

$$H_t = O_t \odot \tanh(C_t)$$



FC layer with
activation function



Elementwise
operator



Copy



Concatenate

总结

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$

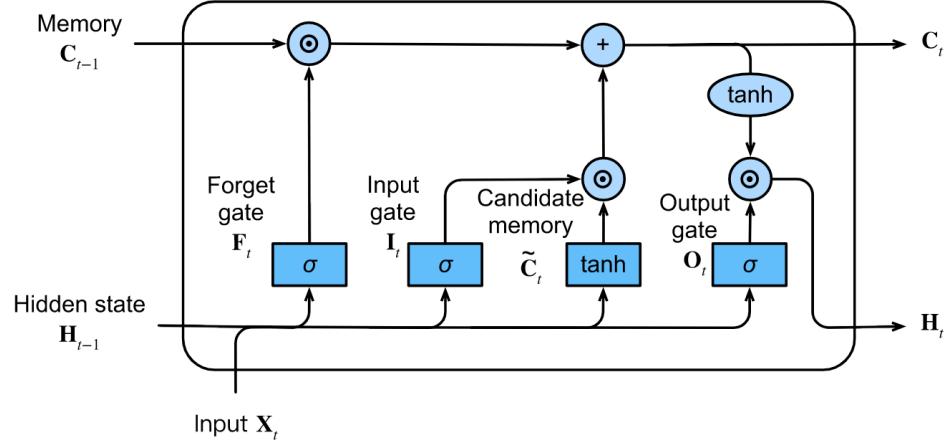
$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$$

$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$

$$H_t = O_t \odot \tanh(C_t)$$



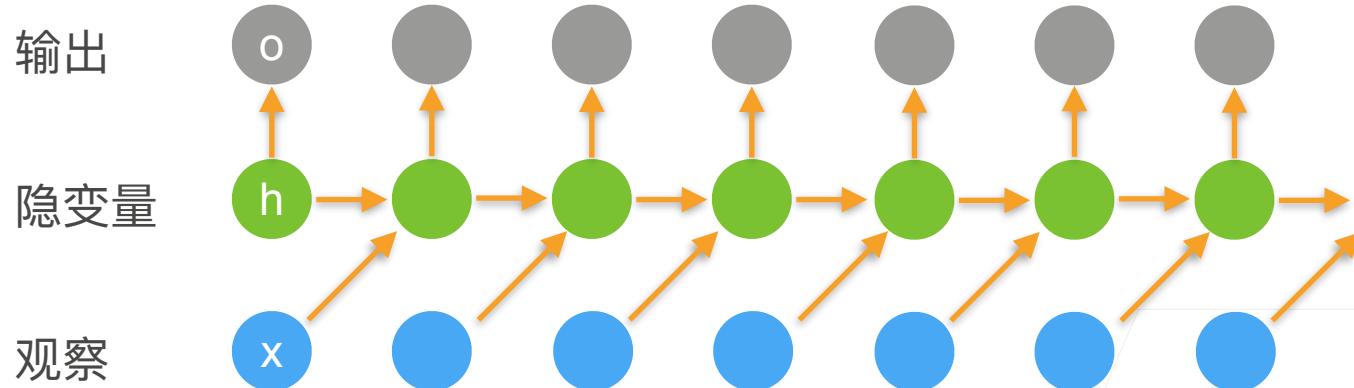


李沐 · AWS

深度循环神经网络



回顾：循环神经网络

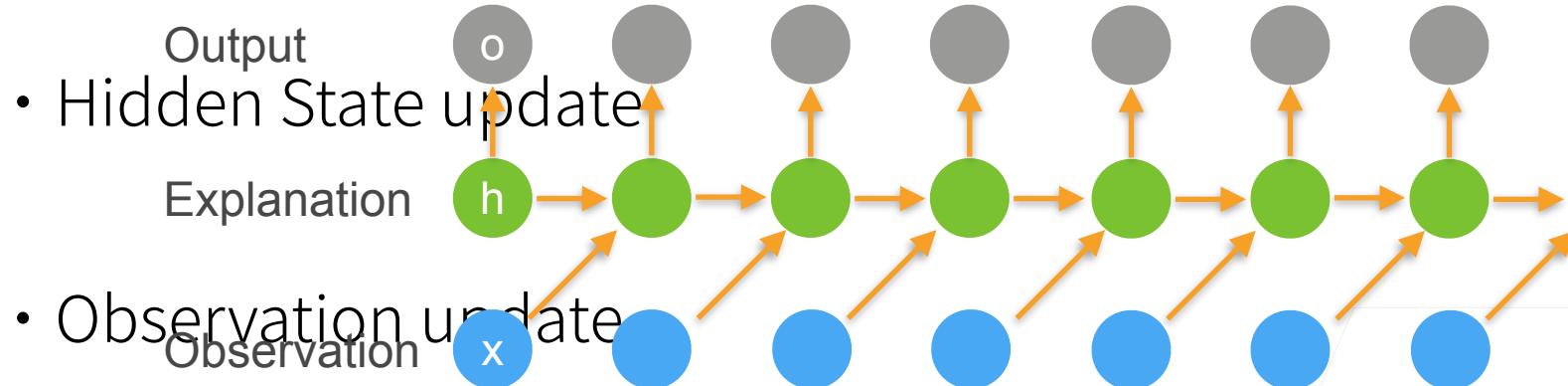


- 更新隐藏状态： $\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_{t-1} + \mathbf{b}_h)$
- 输出： $\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$

如何得到更多的
非线性？



Plan A - Nonlinearity in the units



$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_{t-1} + \mathbf{b}_h)$$

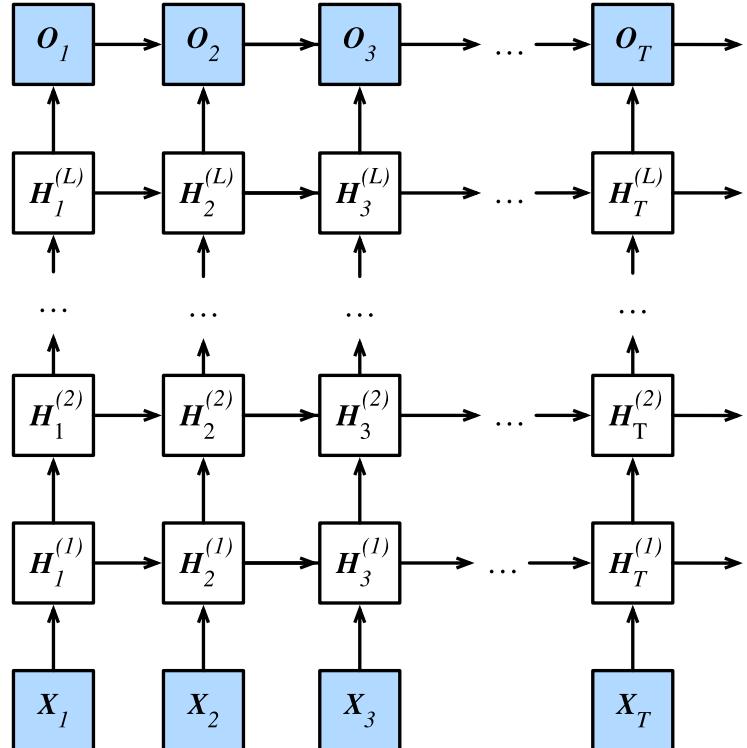
$$\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

Replace with
MLP?



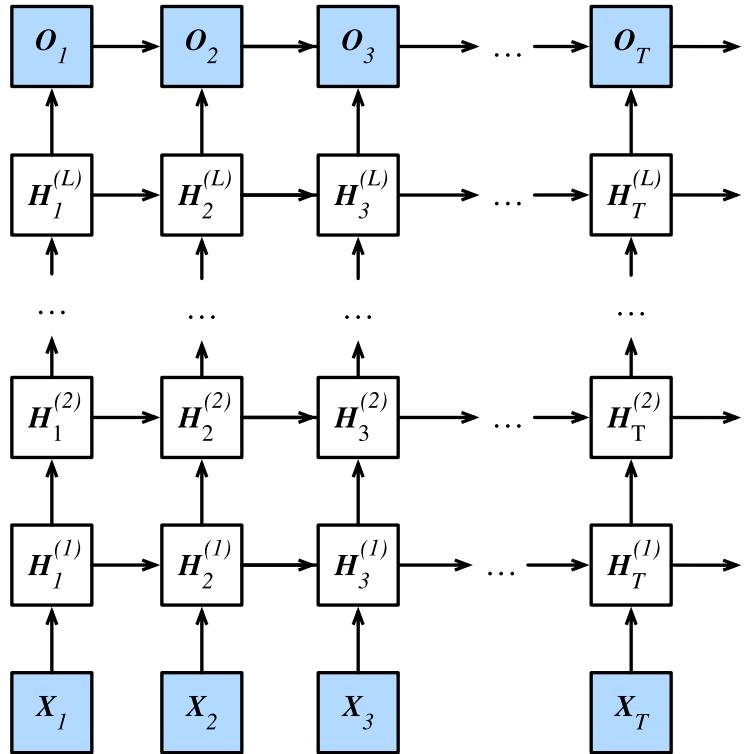
更深

- 浅 RNN
 - 输入
 - 隐层
 - 输出
- 深 RNN
 - 输入
 - 隐层
 - 隐层
 - ...
 - 输出





更深



$$\mathbf{H}_t = f(\mathbf{H}_{t-1}, \mathbf{X}_t)$$

$$\mathbf{O}_t = g(\mathbf{H}_t)$$

$$\mathbf{H}_t^1 = f_1(\mathbf{H}_{t-1}^1, \mathbf{X}_t)$$

$$\mathbf{H}_t^j = f_j(\mathbf{H}_{t-1}^j, \mathbf{H}_t^{j-1})$$

$$\mathbf{O}_t = g(\mathbf{H}_t^L)$$



总结

- 深度循环神经网络使用多个隐藏层来获得更多的非线性性

李沫 · AWS

双向循环神经网络





未来很重要

I am _____

I am _____ very hungry,

I am _____ very hungry, I could eat half a pig.



未来很重要

I am **happy**.

I am **not** very hungry,

I am **very** very hungry, I could eat half a pig.

未来很重要



I am **happy**.

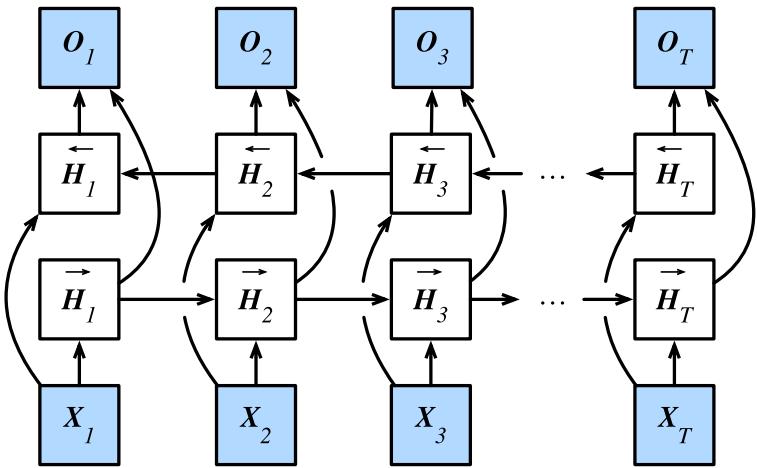
I am **not** very hungry,

I am **very** very hungry, I could eat half a pig.

- 取决于过去和未来的上下文，可以填很不一样的词
- 目前为止RNN只看过去
- 在填空的时候，我们也可以看未来



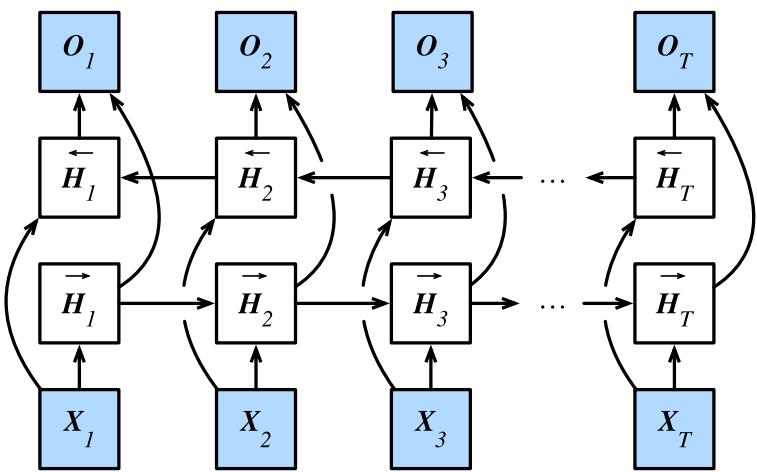
双向 RNN



- 一个前向RNN隐层
- 一个方向RNN隐层
- 合并两个隐状态得到输出



双向 RNN



$$\vec{\mathbf{H}}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh}^{(f)} + \vec{\mathbf{H}}_{t-1} \mathbf{W}_{hh}^{(f)} + \mathbf{b}_h^{(f)}),$$

$$\overleftarrow{\mathbf{H}}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh}^{(b)} + \overleftarrow{\mathbf{H}}_{t+1} \mathbf{W}_{hh}^{(b)} + \mathbf{b}_h^{(b)}),$$

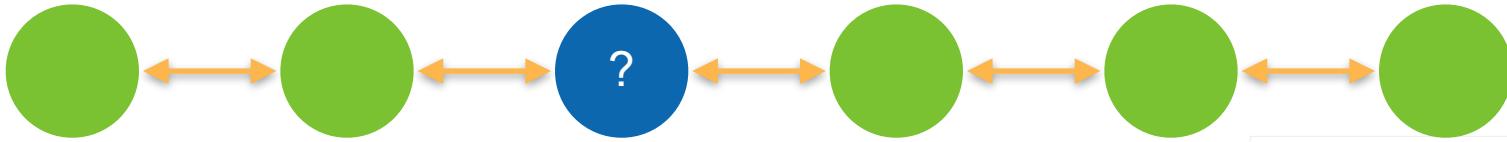
$$\mathbf{H}_t = [\vec{\mathbf{H}}_t, \overleftarrow{\mathbf{H}}_t]$$

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q$$



推理

- 训练：



- 推理：





总结

- 双向循环神经网络通过反向更新的隐藏层来利用方向时间信息
- 通常用来对序列抽取特征、填空，而不是预测未来



编码器-解码器

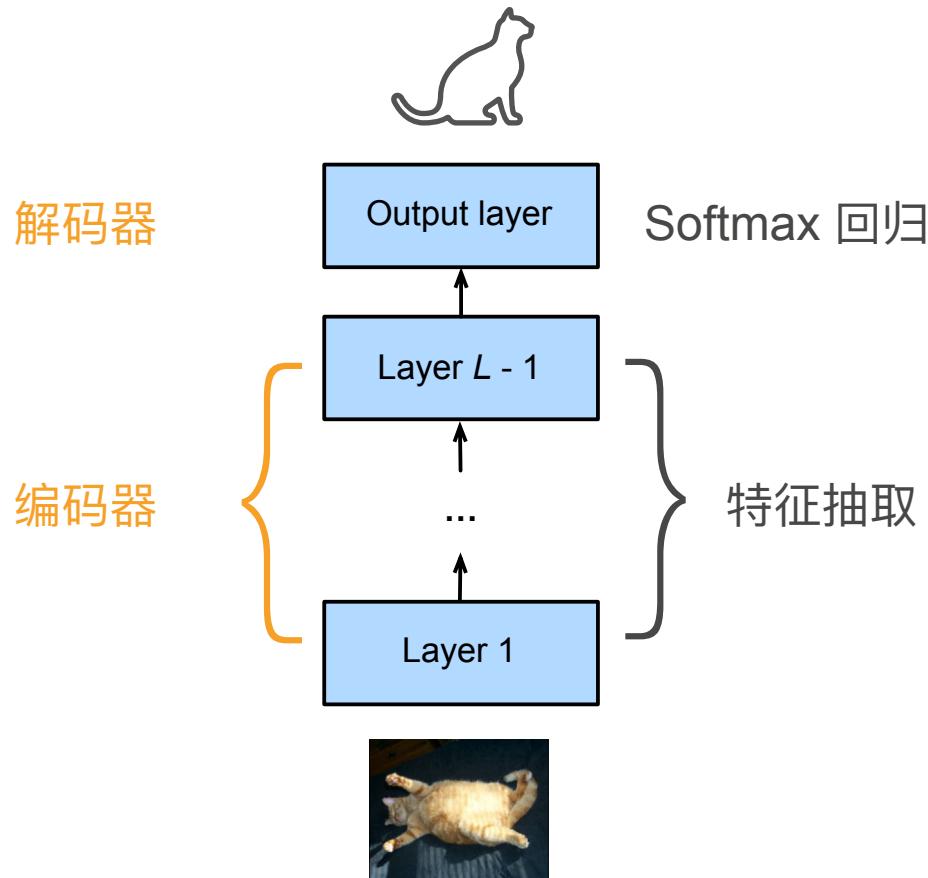
动手学深度学习 v2

李沐 · AWS





重新考察CNN

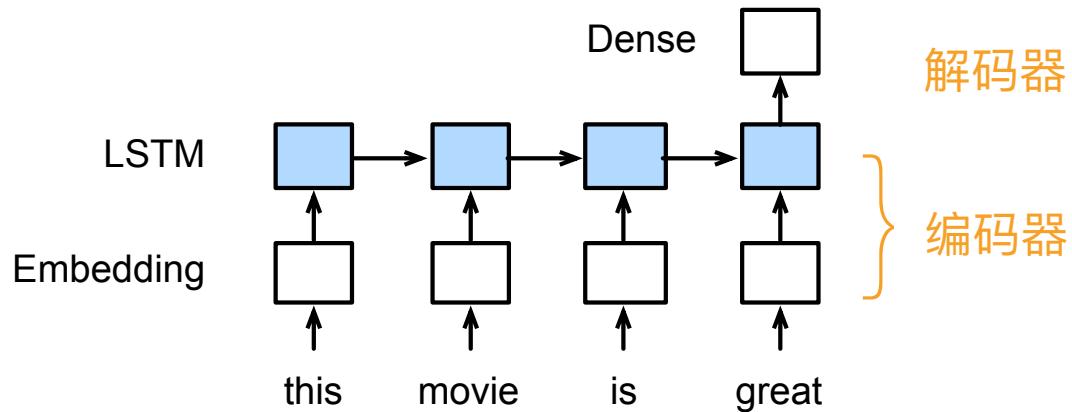


- 编码器：将输入编程成中间表达形式（特征）
- 解码器：将中间表示解码成输出



重新考察RNN

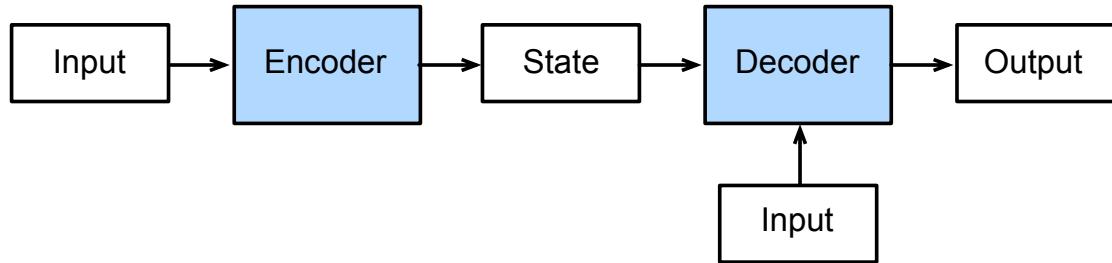
- 编码器：将文本表示成向量
- 解码器：向量表示成输出





编码器-解码器架构

- 一个模型被分为两块：
 - 编码器处理输出
 - 解码器生成输出



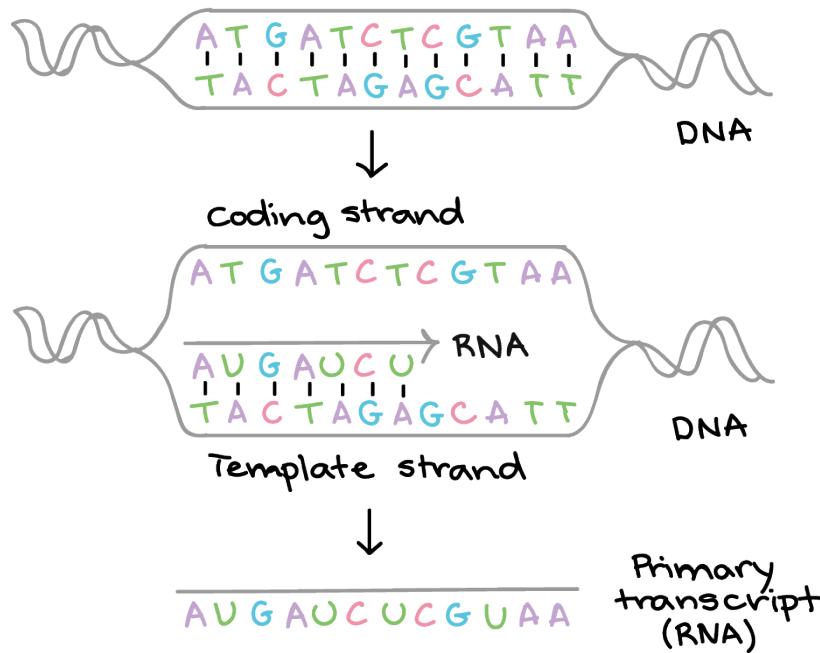
总结



- 使用编码器-解码器架构的模型， 编码器负责表示输入， 解码器负责输出



序列到序列学习 (seq2seq)





机器翻译

- 给定一个源语言的句子，自动翻译成目标语言
- 这两个句子可以有不同的长度

The screenshot shows a machine translation application with the following interface elements:

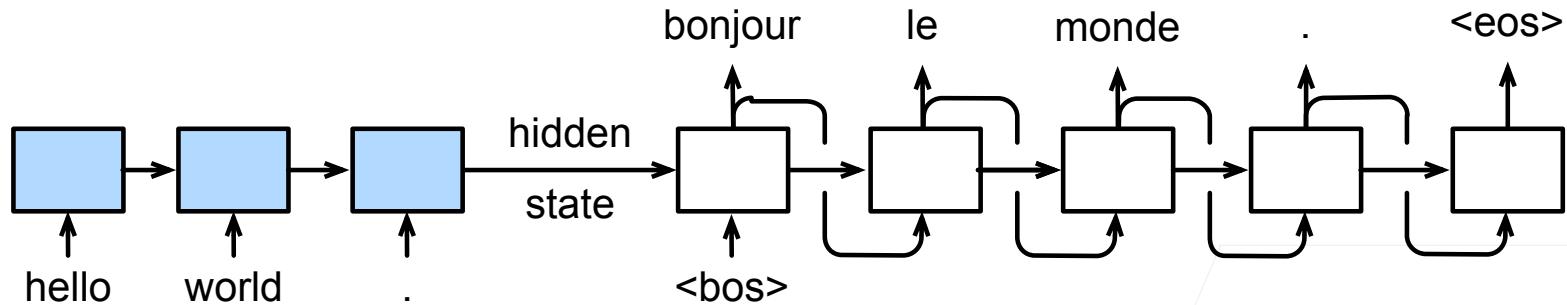
- Source Language:** DETECT LANGUAGE, CHINESE, ENGLISH (highlighted), SPANISH.
- Target Language:** ENGLISH, CHINESE (Simplified) (highlighted), SPANISH.
- Input Text (English):** Following the design principle of the encoder-decoder architecture, the RNN encoder can take a variable-length sequence as the input and transforms it into a fixed-shape hidden state.
- Output Text (Chinese Simplified):** 遵循encoder-decoder架构的设计原则，RNN编码器可以将一个可变长度的序列作为输入，将其转化为固定形状的隐藏状态。
- Output Text (Pinyin):** Zūnxún encoder-decoder jiàgòu de shèjì yuánzé,RNN biānmǎ qì kěyǐ jiāng yīgè kě biàn chángdù de xùliè zuòwéi shùrù, jiāng qí zhuǎnhuà wéi gùding xíngzhuàng de yǐncáng zhuàngtài.
- Interaction Buttons:** Microphone icon, speaker icon, progress bar (183 / 5000), edit icon, and back/forward icons.

Seq2seq



Encoder

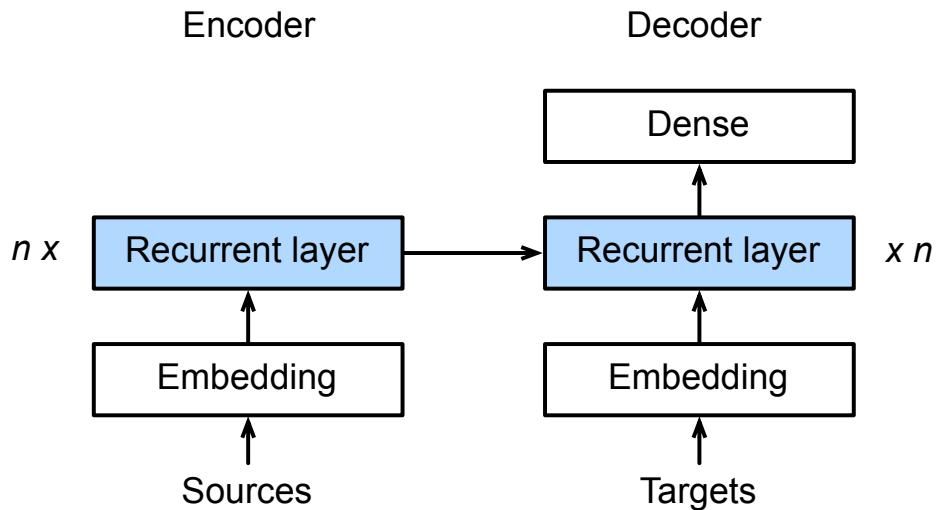
Decoder



- 编码器是一个RNN，读取输入句子
 - 可以是双向
- 解码器使用另外一个RNN来输出



编码器-解码器细节

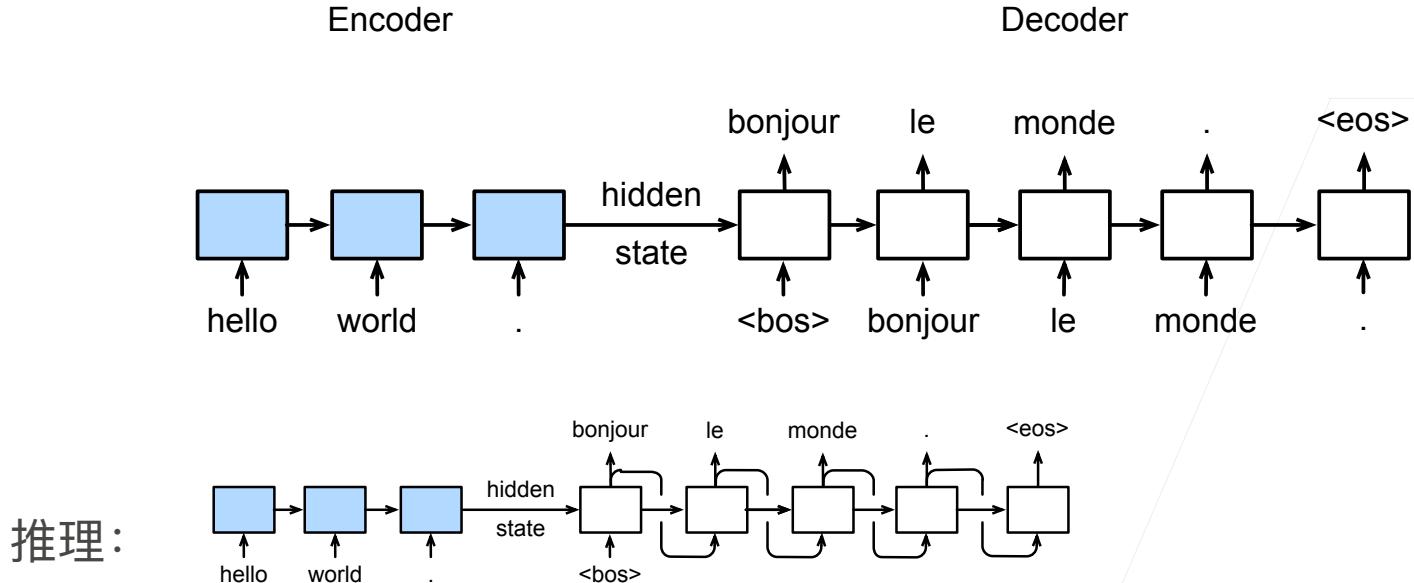


- 编码器是没有输出蹭到RNN
- 编码器最后时间步的隐状态用作解码器的初始隐状态



训练

- 训练时解码器使用目标句子作为输入





衡量生成序列的好坏的BLEU

- p_n 是预测中所有 n-gram 的精度
 - 标签序列 $A \ B \ C \ D \ E \ F$ 和预测序列 $A \ B \ B \ C \ D$, 有
$$p_1 = 4/5, p_2 = 3/4, p_3 = 1/3, p_4 = 0$$
- BLEU 定义

$$\exp \left(\min \left(0, 1 - \frac{\text{len}_{\text{label}}}{\text{len}_{\text{pred}}} \right) \right) \prod_{n=1}^k p_n^{1/2^n}$$

动手学深
惩罚过短的预测

长匹配有高权重



总结

- Seq2seq从一个句子生成另一个句子
- 编码器和解码器都是RNN
- 将编码器最后时间隐状态来初始解码器隐状态来完成信息传递
- 常用BLEU来衡量生成序列的好坏



束搜索





贪心搜索

- 在seq2seq中我们使用了贪心搜索来预测序列
 - 将当前时刻预测概率最大的词输出
- 但贪心很可能不是最优的：

贪心： $0.5 \times 0.4 \times 0.4 \times 0.6 = 0.048$

很好的选项： $0.5 \times 0.3 \times 0.6 \times 0.6 = 0.054$

Time step	1	2	3	4
A	0.5	0.1	0.2	0.0
B	0.2	0.4	0.2	0.2
C	0.2	0.3	0.4	0.2
<eos>	0.1	0.2	0.2	0.6

Time step	1	2	3	4
A	0.5	0.1	0.1	0.1
B	0.2	0.4	0.6	0.2
C	0.2	0.3	0.2	0.1
<eos>	0.1	0.2	0.1	0.6



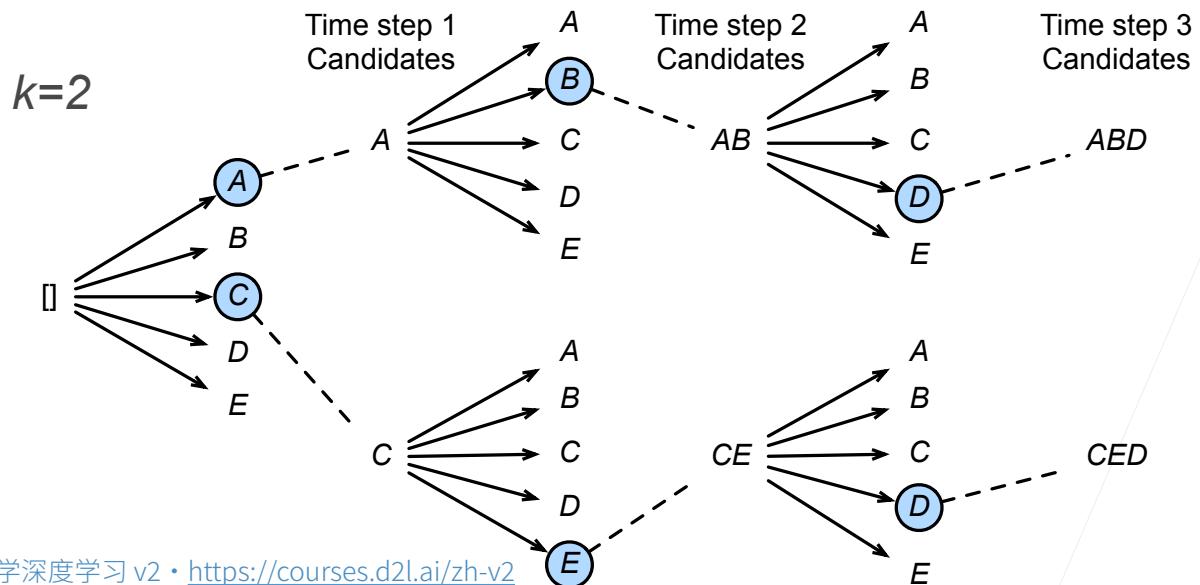
穷举搜索

- 最优算法：对所有可能的序列，计算它的概率，然后选取最好的那个
- 如果输出字典大小为 n ，序列最长为 T ，那么我们需要考察 n^T 个序列
 - $n = 10000, T = 10 : n^T = 10^{40}$
 - 计算上不可行



束搜索

- 保存最好的 k 个候选
- 在每个时刻，对每个候选新加一项 (n 种可能)，在 kn 个选项中选出最好的 k 个





束搜索

- 时间复杂度 $O(knT)$
 - $k = 5, n = 10000, T = 10 : knT = 5 \times 10^5$
- 每个候选的最终分数是：

$$\frac{1}{L^\alpha} \log p(y_1, \dots, y_L) = \frac{1}{L^\alpha} \sum_{t'=1}^L \log p(y_{t'} \mid y_1, \dots, y_{t'-1}, \mathbf{c})$$

- 通常 $\alpha = 0.75$



总结

- 束搜索在每次搜索时保存 k 个最好的候选
 - $k = 1$ 时是贪心搜索
 - $k = n$ 时是穷举搜索



自注意力和位置编码

动手学深度学习 v2

李沐 · AWS

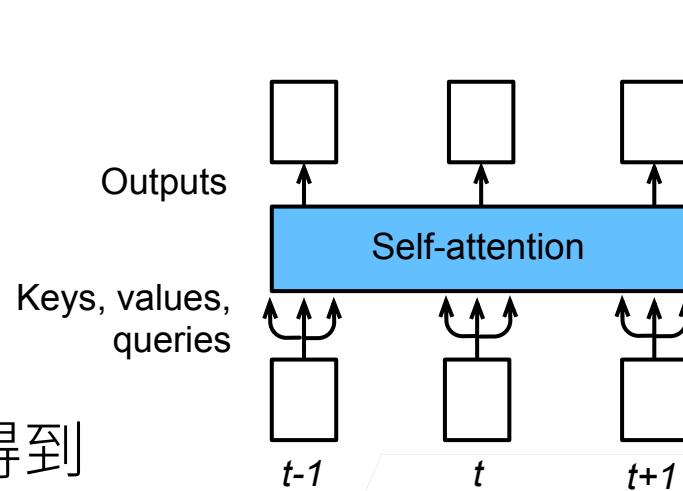




自注意力

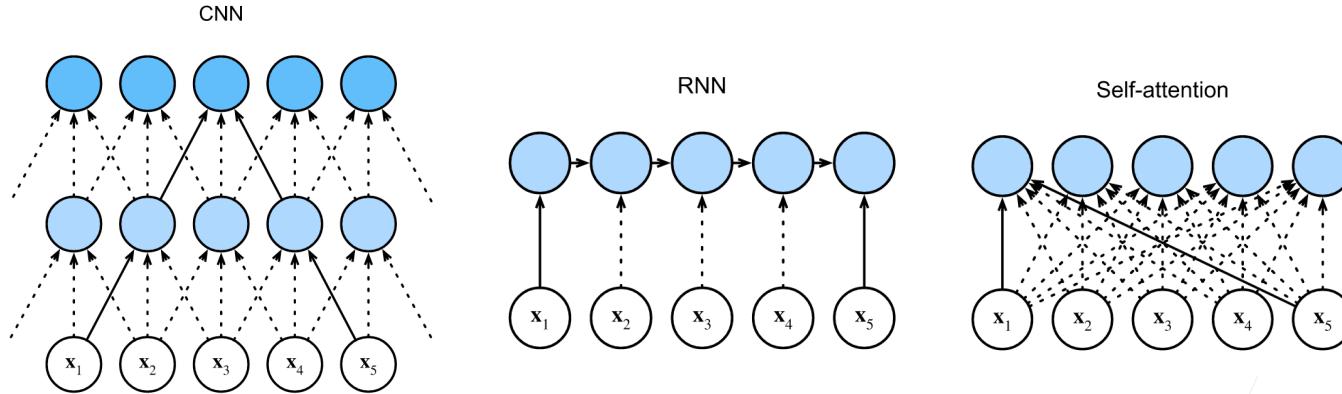
- 给定序列 $\mathbf{x}_1, \dots, \mathbf{x}_n, \forall \mathbf{x}_i \in \mathbb{R}^d$
- 自注意力池化层将 \mathbf{x}_i 当做key, value, query来对序列抽取特征得到 $\mathbf{y}_1, \dots, \mathbf{y}_n$, 这里

$$\mathbf{y}_i = f(\mathbf{x}_i, (\mathbf{x}_1, \mathbf{x}_1), \dots, (\mathbf{x}_n, \mathbf{x}_n)) \in \mathbb{R}^d$$





跟CNN，RNN对比



	CNN	RNN	自注意力
计算复杂度	$O(knd^2)$	$O(nd^2)$	$O(n^2d)$
并行度	$O(n)$	$O(1)$	$O(n)$
最长路径	$O(n/k)$	$O(n)$	$O(1)$



位置编码

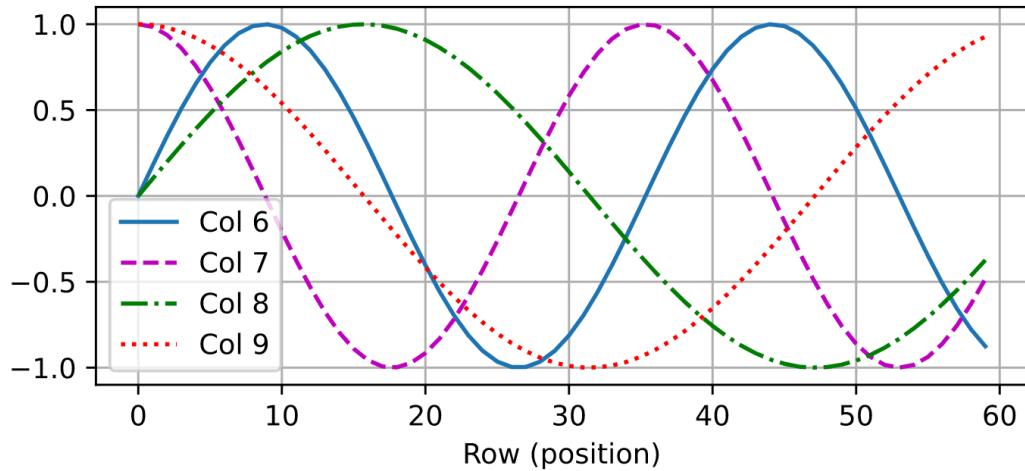
- 跟CNN/RNN不同，自注意力并没有记录位置信息
- 位置编码将位置信息注入到输入里
 - 假设长度为 n 的序列是 $\mathbf{X} \in \mathbb{R}^{n \times d}$ ，那么使用位置编码矩阵 $\mathbf{P} \in \mathbb{R}^{n \times d}$ 来输出 $\mathbf{X} + \mathbf{P}$ 作为自编码输入
- \mathbf{P} 的元素如下计算：

$$p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d}}\right), \quad p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d}}\right)$$



位置编码矩阵

- $\mathbf{P} \in \mathbb{R}^{n \times d}$: $p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d}}\right)$, $p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d}}\right)$



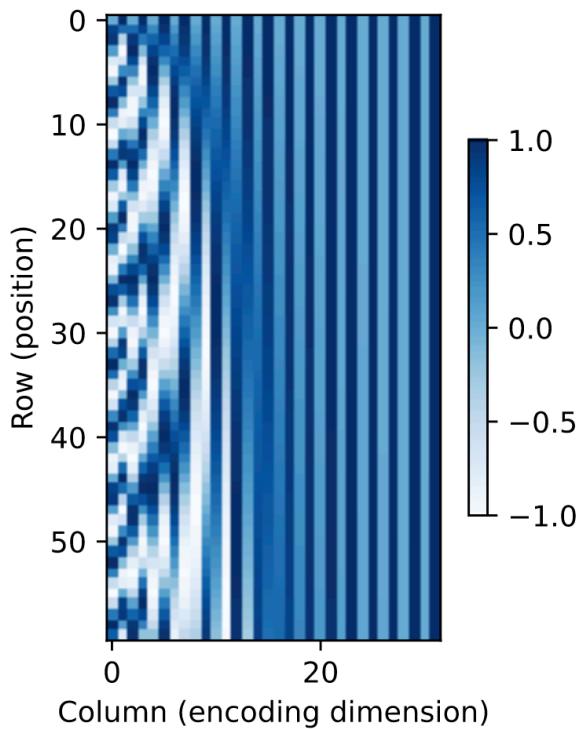


绝对位置信息

计算机使用的二进制编码

```
0 in binary is 000
1 in binary is 001
2 in binary is 010
3 in binary is 011
4 in binary is 100
5 in binary is 101
6 in binary is 110
7 in binary is 111
```

位置编码矩阵





相对位置信息

- 位于 $i+\delta$ 处的位置编码可以线性投影位置 i 处的位置编码来表示
- 记 $\omega_j = 1/10000^{2j/d}$, 那么

$$\begin{bmatrix} \cos(\delta\omega_j) & \sin(\delta\omega_j) \\ -\sin(\delta\omega_j) & \cos(\delta\omega_j) \end{bmatrix} \begin{bmatrix} p_{i,2j} \\ p_{i,2j+1} \end{bmatrix} = \begin{bmatrix} p_{i+\delta,2j} \\ p_{i+\delta,2j+1} \end{bmatrix}$$

投影矩阵
跟 i 无关



总结

- 自注意力池化层将 \mathbf{x}_i 当做key, value, query来对序列抽取特征
- 完全并行、最长序列为1、但对长序列计算复杂度高
- 位置编码在输入中加入位置信息，使得自注意力能够记忆位置信息



动手学深度学习 v2

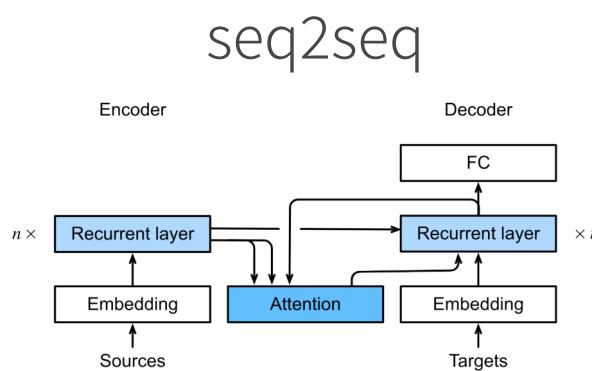
李沐 · AWS





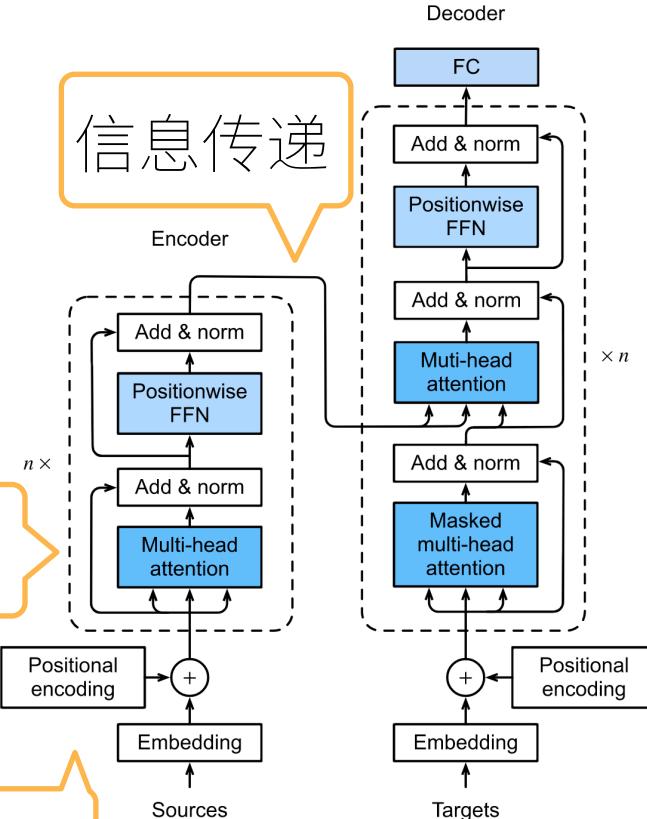
Transformer 架构

- 基于编码器-解码器架构来处理序列对
- 跟使用注意力的seq2seq不同，Transformer是纯基于注意力



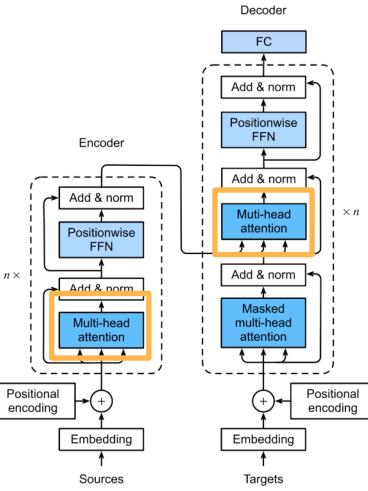
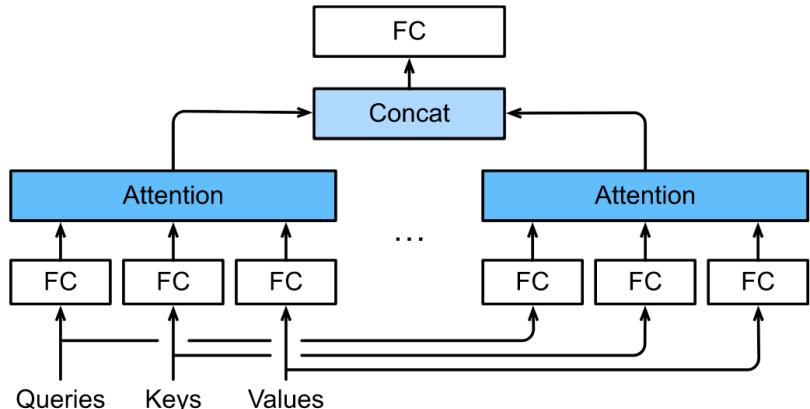
Transformer块

位置编码



多头注意力

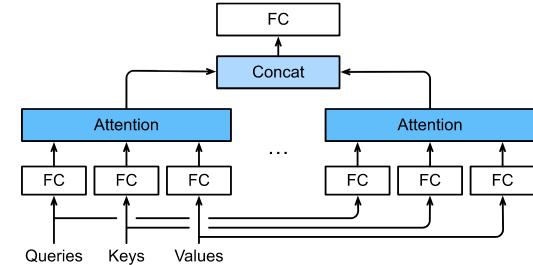
- 对同一key, value, query, 希望抽取不同的信息
 - 例如短距离关系和长距离关系
- 多头注意力使用 h 个独立的注意力池化
 - 合并各个头 (head) 输出得到最终输出



多头注意力

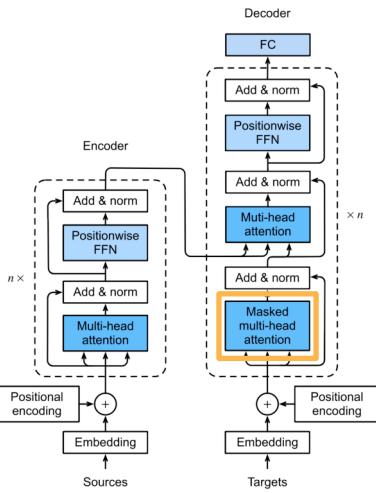
- query $\mathbf{q} \in \mathbb{R}^{d_q}$, key $\mathbf{k} \in \mathbb{R}^{d_k}$, value $\mathbf{v} \in \mathbb{R}^{d_v}$
- 头 i 的可学习参数 $\mathbf{W}_i^{(q)} \in \mathbb{R}^{p_q \times d_q}$, $\mathbf{W}_i^{(k)} \in \mathbb{R}^{p_k \times d_k}$, $\mathbf{W}_i^{(v)} \in \mathbb{R}^{p_v \times d_v}$
- 头 i 的输出 $\mathbf{h}_i = f(\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v}) \in \mathbb{R}^{p_v}$
- 输出的可学习参数 $\mathbf{W}_o \in \mathbb{R}^{p_o \times hp_v}$
- 多头注意力的输出

$$\mathbf{W}_o \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_h \end{bmatrix} \in \mathbb{R}^{p_o}$$



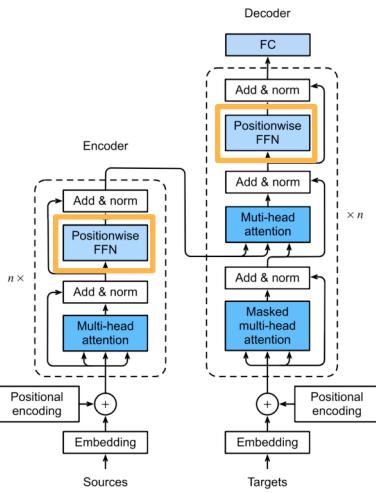
有掩码的多头注意力

- 解码器对序列中一个元素输出时，不应该考虑该元素之后的元素
- 可以通过掩码来实现
 - 也就是计算 \mathbf{x}_i 输出时，假装当前序列长度为 i



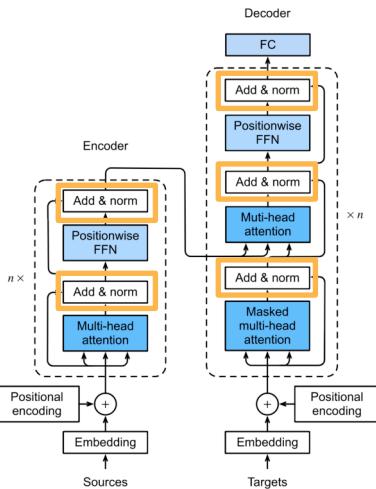
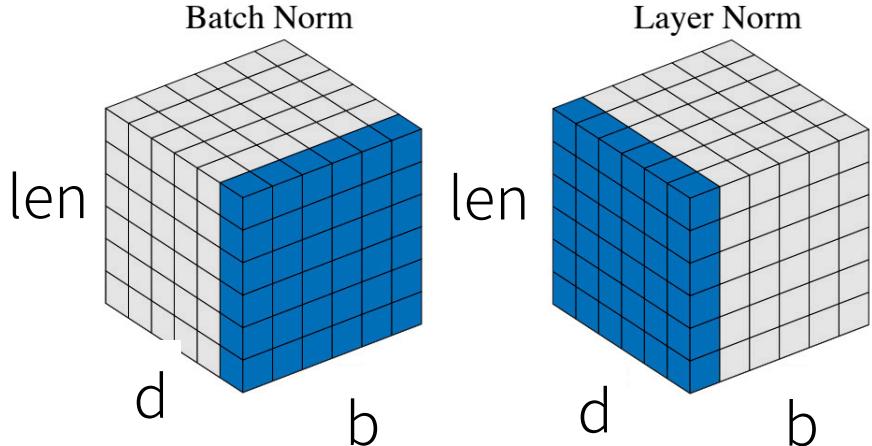
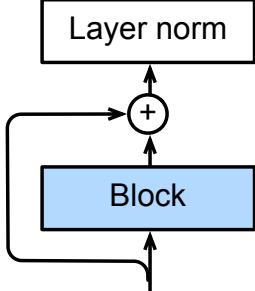
基于位置的前馈网络

- 将输入形状由 (b, n, d) 变换成 (bn, d)
- 作用两个全连接层
- 输出形状由 (bn, d) 变化回 (b, n, d)
- 等价于两层核窗口为1的一维卷积层



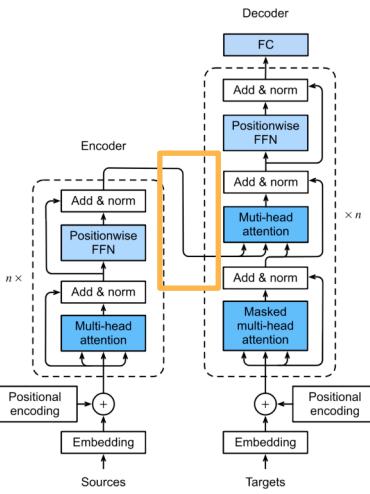
层归一化

- 批量归一化对每个特征/通道里元素进行归一化
 - 不适合序列长度会变的NLP应用
- 层归一化对每个样本里的元素进行归一化



信息传递

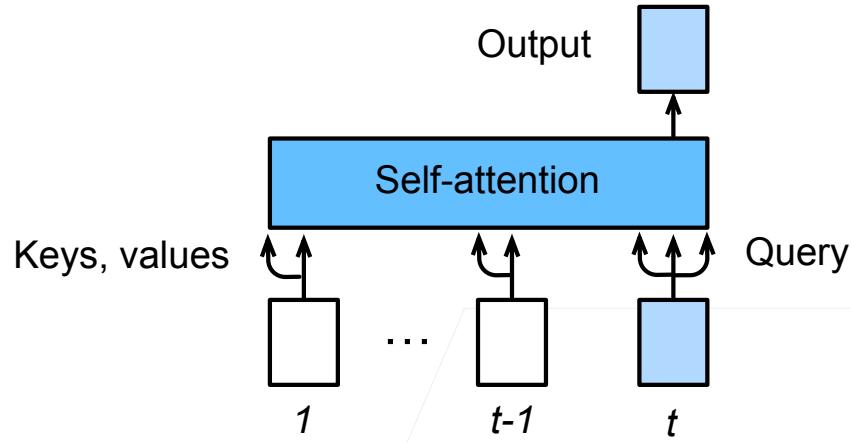
- 编码器中的输出 $\mathbf{y}_1, \dots, \mathbf{y}_n$
- 将其作为解码中第 i 个Transformer块中多头注意力的key和value
 - 它的query来自目标序列
- 意味着编码器和解码器中块的个数和输出维度都是一样的





预测

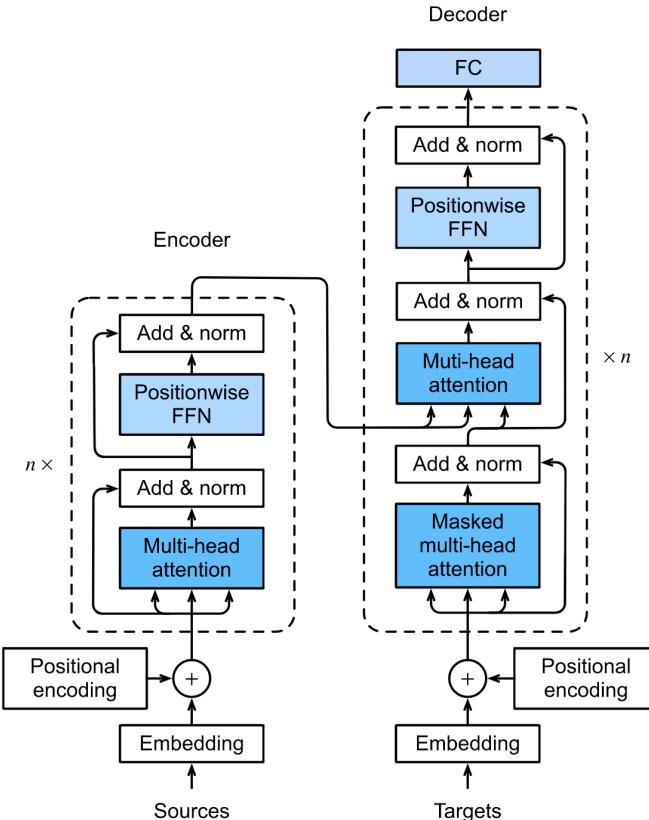
- 预测第 $t + 1$ 个输出时
- 解码器中输入前 t 个预测值
 - 在自注意力中，前 t 个预测值作为key和value，第 t 个预测值还作为query





总结

- Transformer是一个纯使用注意力的编码-解码器
- 编码器和解码器都有 n 个transformer块
- 每个块里使用多头（自）注意力，基于位置的前馈网络，和层归一化



BERT



动手学深度学习 v2
李沐 · AWS





NLP里的迁移学习

- 使用预训练好的模型来抽取词、句子的特征
 - 例如 word2vec 或语言模型
- 不更新预训练好的模型
- 需要构建新的网络来抓取新任务需要的信息
 - Word2vec 忽略了时序信息，
语言模型只看了一个方向

BERT的动机



- 基于微调的NLP模型
- 预训练的模型抽取了足够多的信息
- 新的任务只需要增加一个简单的输出层

NLP



CV



Output layer

Layer $L - 1$

...

Layer 1

Output layer

Layer $L - 1$

...

Layer 1

分类器

特征抽取

I love this movie





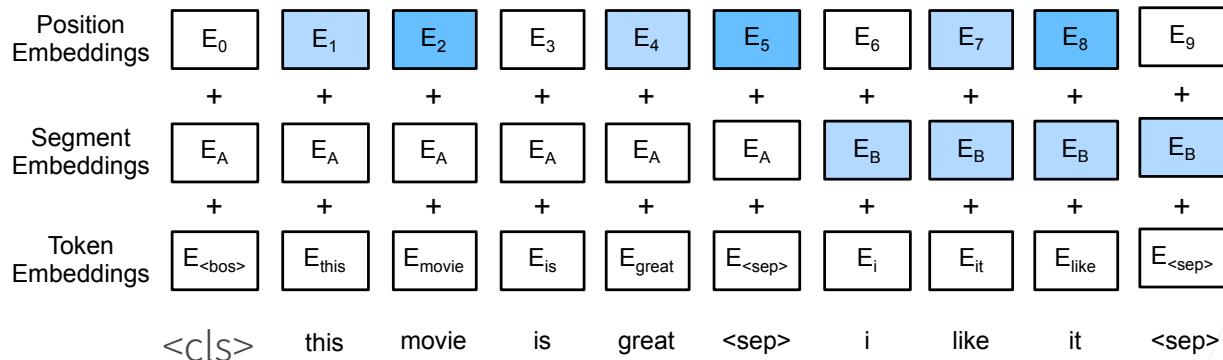
BERT 架构

- 只有编码器的 Transformer
- 两个版本：
 - Base: #blocks = 12, hidden size = 768, #heads = 12, #parameters = 110M
 - Large: #blocks = 24, hidden size = 1024, #heads = 16, #parameter = 340M
- 在大规模数据上训练 > 3B 词



对输入的修改

- 每个样本是一个句子对
- 加入额外的片段嵌入
- 位置编码可学习





预训练任务1：带掩码的语言模型

- Transformer的编码器是双向， 标准语言模型要求单向
- 带掩码的语言模型每次随机（15%概率） 将一些词元换成 <mask>
- 因为微调任务中不出现<mask>
 - 80%概率下， 将选中的词元变成<mask>
 - 10%概率下换成一个随机词元
 - 10%概率下保持原有的词元



预训练任务2：下一句子预测

- 预测一个句子对中两个句子是不是相邻
- 训练样本中：
 - 50%概率选择相邻句子对：<cls> this movie is great <sep> i like it <sep>
 - 50%概率选择随机句子对：<cls> this movie is great <sep> hello world <sep>
- 将<cls>对应的输出放到一个全连接层来预测



总结

- BERT针对微调设计
- 基于Transformer的编码器做了如下修改
 - 模型更大，训练数据更多
 - 输入句子对，片段嵌入，可学习的位置编码
 - 训练时使用两个任务：
 - 带掩码的语言模型
 - 下一个句子预测



BERT微调

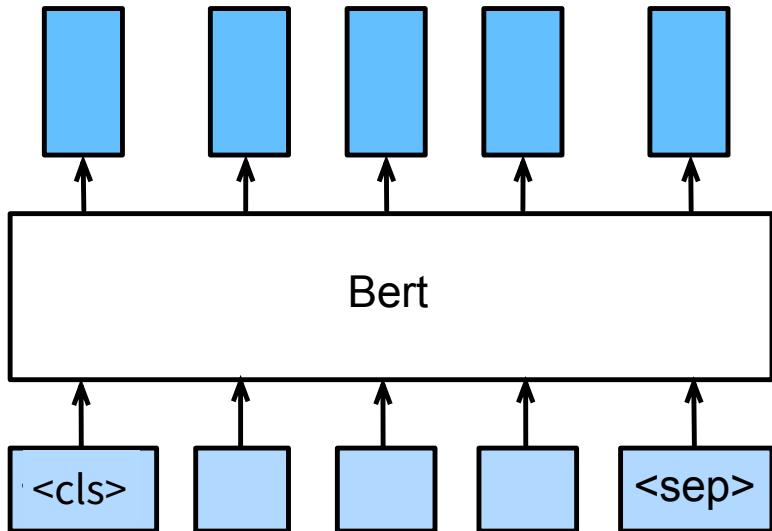
动手学深度学习 v2
李沐 · AWS





微调 Bert

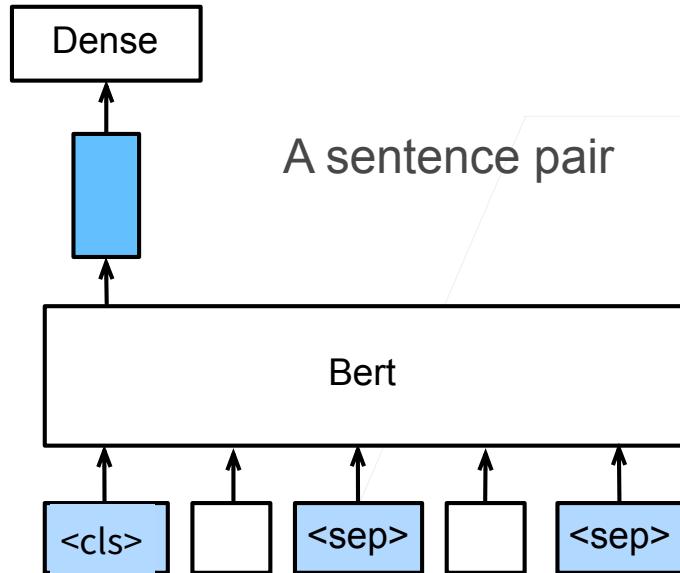
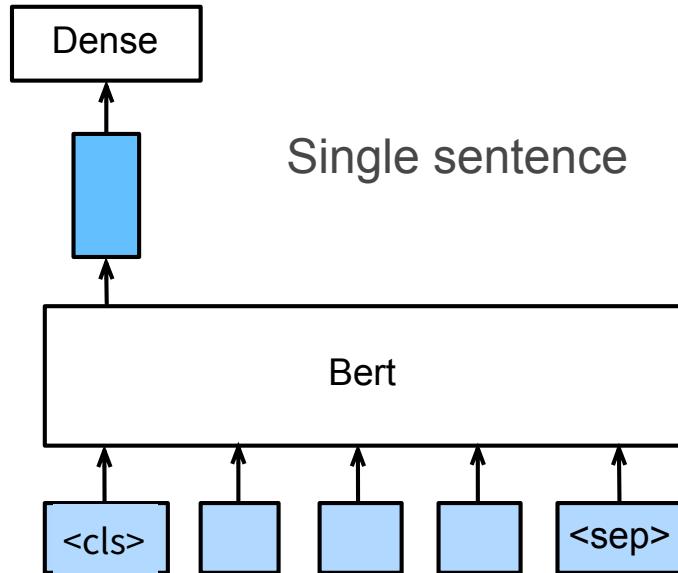
- BERT 对每一个词元返回抽取了上下文信息的特征向量
- 不同的任务使用不用的特性





句子分类

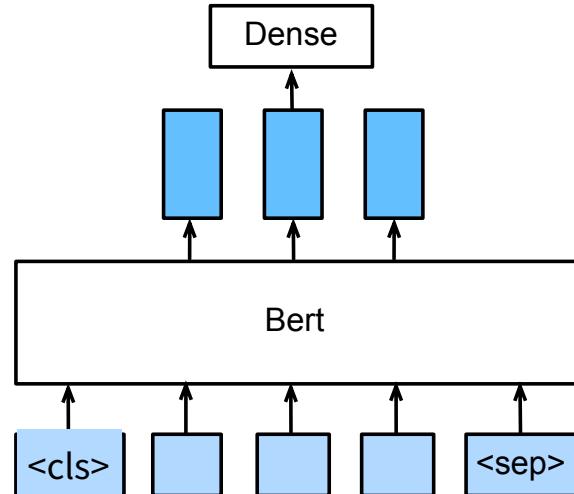
- 将 <cls> 对应的向量输入到全连接层分类





命名实体识别

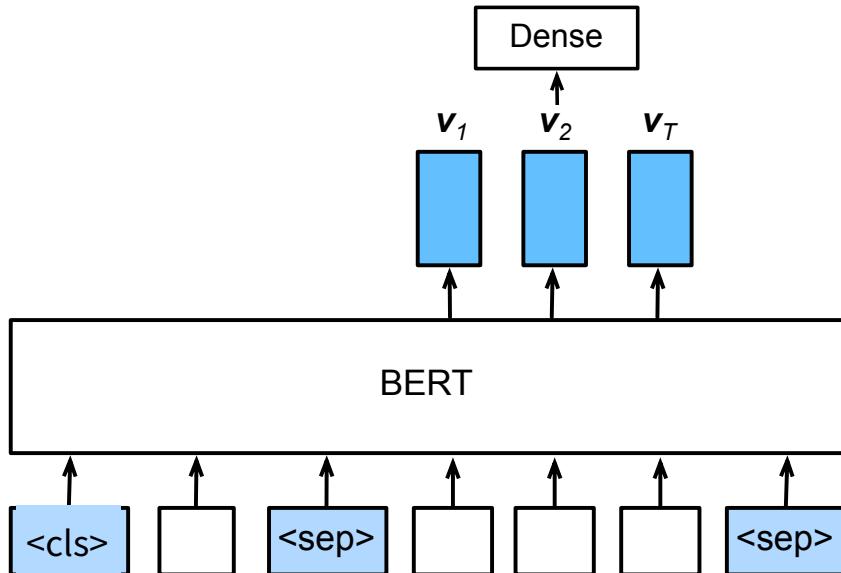
- 识别一个词元是不是命名实体，例如人名、机构、位置
- 将非特殊词元放进全连接层分类





问题回答

- 给定一个问题，和描述文字，找出一个片段作为回答
- 对片段中的每个词元预测它是不是回答的开头或结束





总结

- 即使下游任务各有不同，使用BERT微调时均只需要增加输出层
- 但根据任务的不同，输入的表示，和使用的BERT特征也会不一样

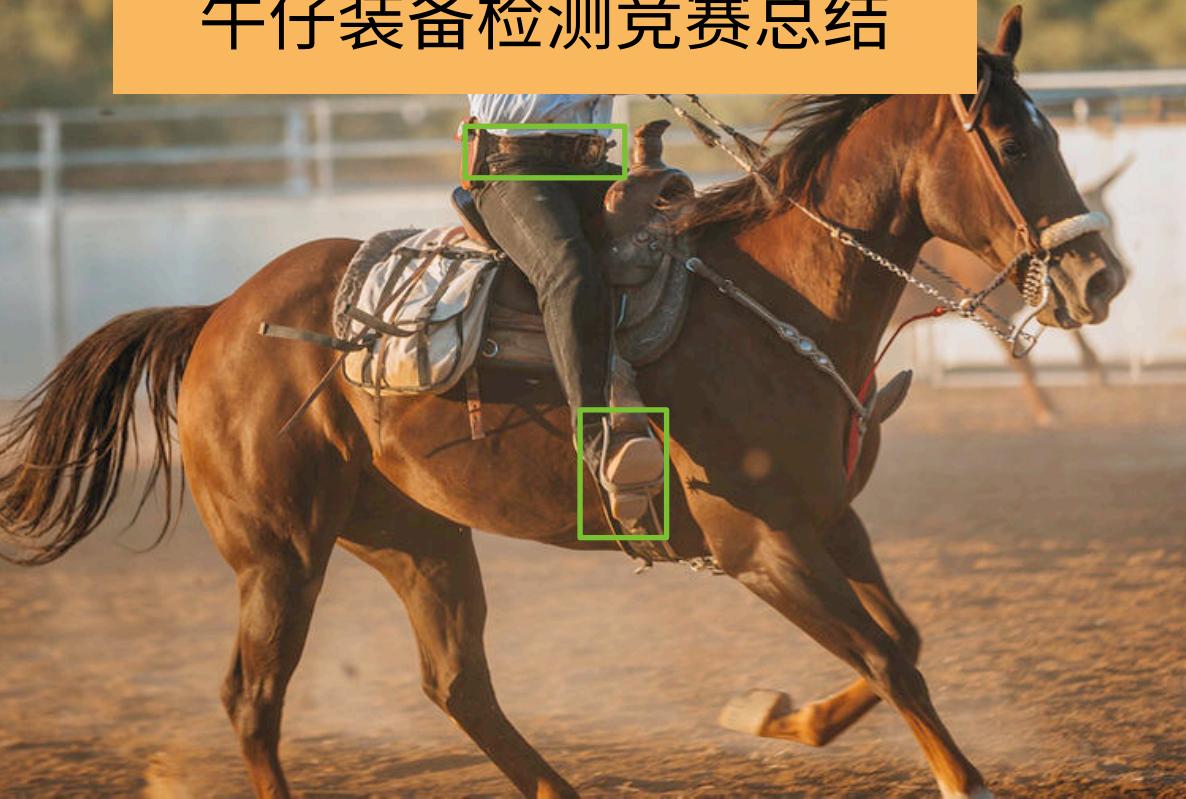


动手学深度学习 v2

李沐 · AWS



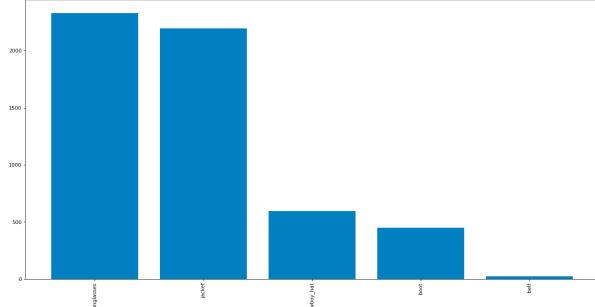
牛仔装备检测竞赛总结





任务回顾

- 检测牛仔夹克、墨镜、靴子、牛仔帽、腰带
- 6937张训练图片，12660标注框
- 数据使用MS-COCO格式，评测使用mAP
 - 均可直接调用 pycocotools
- 挑战：
类别不平衡



结果

- 23名同学提交了300次结果
- 15名同学公布了代码
 - <https://www.kaggle.com/c/cowboyoutfits/code>

InClass Prediction Competition

CowBoy Outfits Detection

Can you train a model to locate cowboy outfits? Accept the challenge of imbalanced training data.

10 days ago

Overview Data Code Discussion Leaderboard Rules New Notebook ...

Search notebooks Filters

All Your Work Shared With You Bookmarks Hotness

Cowboy Outfits Detection Megui_YOLOX Updated 7d ago 1 comment - Cowboy Outfits Detection

[2nd] picos for cowboyoutfits Updated 6d ago 1 comment - Cowboy Outfits Detection +1

yolov5I Updated 10d ago 0 comments - Cowboy Outfits Detection

4th:[Training]YOLOv5+transformer-module+ensemble Updated 7d ago 0 comments - Cowboy Outfits Detection

6th: [Training]cowboy detectron2 faster-rcnn Updated 8d ago 0 comments - Cowboy Outfits Detection

cow_boy_outfits_YOLOv5+k-fold+ensemble Updated 10d ago 1 comment - Cowboy Outfits Detection

torchvision_faster-rcnn_PB_58.9 Updated 10d ago 3 comments - Cowboy Outfits Detection

windows7使用Yolov5训练于牛仔行头检测 Updated 17d ago 2 comments - Cowboy Outfits Detection

用yolov5进行cowboy outfit detection Notebook copied with edits from depaul - Updated 10d ago 0 comments - CowBoy Outfits Detection

迷你fasterRCNN Updated 18d ago 0 comments - CowBoy Outfits Detection

8th:[Training]MMDetection+CascadeRCNN+Weight&Bias Updated 9d ago 3 comments - CowBoy Outfits Detection

[Training]cowboy_object_detection_YOLOv5 Updated 15d ago 15 comments - Cowboy Outfits Detection +1

Cowboy_outfits_starter_kit Updated 1m ago 0 comments - Cowboy Outfits Detection

[Rank 8th] Challenge with imbalanced training data Updated 7d ago 3 comments - CowBoy Outfits Detection +1

6th: [inference]cowboy detectron2 faster-rcnn Updated 10d ago 0 comments - CowBoy Outfits Detection +1

faster-rcnn resnet50 Updated 12d ago 2 comments - Cowboy Outfits Detection +1

How does the coco eval work? Updated 13d ago 1 Comment - Cowboy Outfits Detection +1

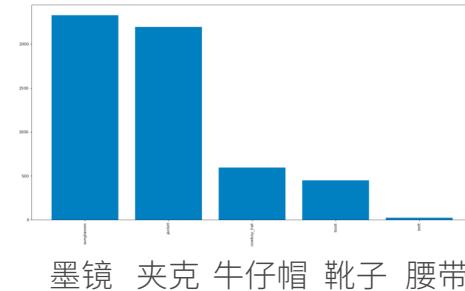
[Training]cowboy_object_detection_using_yolov5rddc Notebook copied with edits from Xinhao Wang - Updated 10d ago 0 comments - CowBoy Outfits Detection +1

Jeans detect Notebook copied with edits from Joshua Z. Zhang - Updated 5d ago 0 comments - CowBoy Outfits Detection +2

- 请Final Phase前10并提交了代码的同学将微信、Kaggle、CodaLab ID发给mli@amazon.com

数据重采样

- 当有类别样本严重不足时，可以人工干预提升它们对模型的影响力
- 最简答将不足的类别样本复制多次
- 在随机采样小批量时对每个类使用不同采样频率
- 在计算损失时增大不足类别样本的权重
- 有同学使用了 SMOTE
 - 在不足类样本的中选择相近对做差值





模型

- YOLOX: YOLOv3 + anchor free
- YOLOv5: YOLOv3 Pytorch版本的改进版
 - YOLOv4和YOLOv5均是社区改进版，命名有争议
- Detectron2: Faster RCNN
- 大都采用了多模型、k则融合



总结

- 目标检测代码实现复杂，训练代价大，上手仍以找到容易上手的库为主
- 因为超参数多，一般需要较长时间探索

SOMMET
3 VALLÉ
3 230m



优化算法



动手学深度学习 v2

李沐 · AWS





优化问题

- 一般形式

$$\text{minimize } f(\mathbf{x}) \quad \text{subject to } \mathbf{x} \in C$$

- 目标函数 $f: \mathbb{R}^n \rightarrow \mathbb{R}$
- 限制集合例子

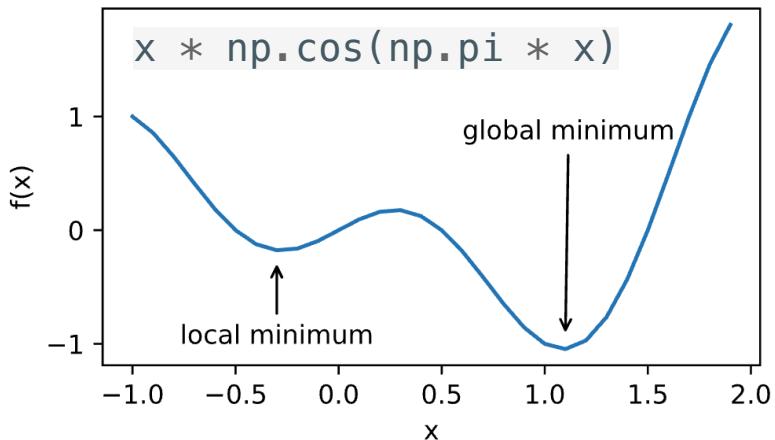
$$C = \{\mathbf{x} \mid h_1(\mathbf{x}) = 0, \dots, h_m(\mathbf{x}) = 0, g_1(\mathbf{x}) \leq 0, \dots, g_r(\mathbf{x}) \leq 0\}$$

- 如果 $C = \mathbb{R}^n$ 那就是不受限



局部最小 vs 全局最小

- 全局最小 \mathbf{x}^* : $f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in C$
- 局部最小 \mathbf{x}^* : 存在 ε , 使得 $f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} : \|\mathbf{x} - \mathbf{x}^*\| \leq \varepsilon$
- 使用迭代优化算法来求解, 一般只能保证找到局部最小值

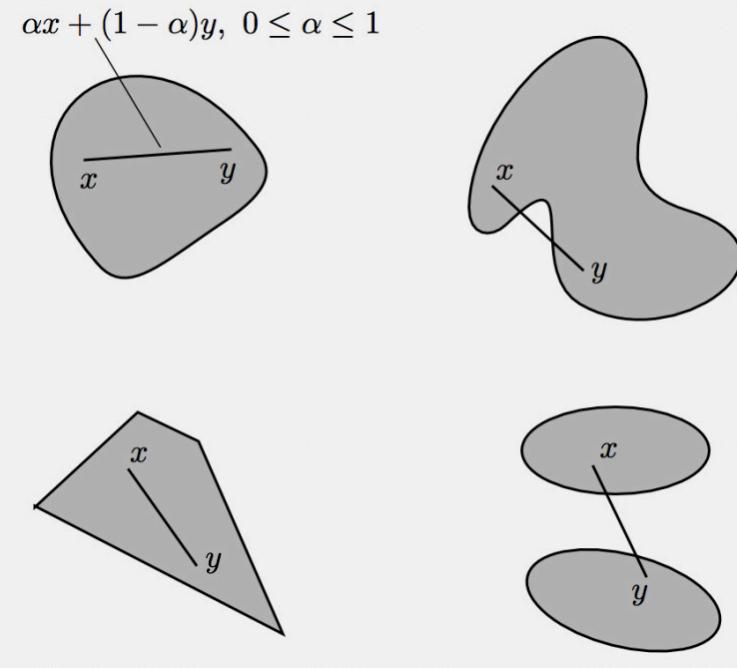


凸集

- 一个 \mathbb{R}^n 的子集 C 是凸当且仅当

$$\alpha \mathbf{x} + (1 - \alpha) \mathbf{y} \in C$$

$$\forall \alpha \in [0,1] \quad \forall \mathbf{x}, \mathbf{y} \in C$$



凸函数

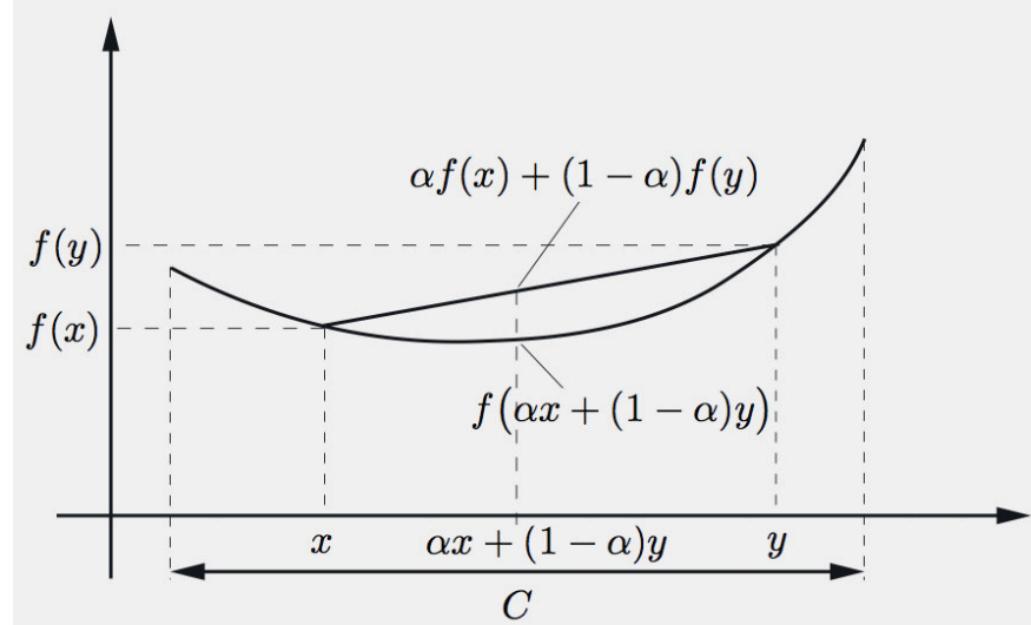
- 函数 $f: C \rightarrow \mathbb{R}$ 是凸的

当且仅当

$$\begin{aligned} f(\alpha \mathbf{x} + (1 - \alpha)\mathbf{y}) \\ \leq \alpha f(\mathbf{x}) + (1 - \alpha)f(\mathbf{y}) \end{aligned}$$

$$\forall \alpha \in [0,1] \quad \forall \mathbf{x}, \mathbf{y} \in C$$

- 如果 $\mathbf{x} \neq \mathbf{y}, \alpha \in (0,1)$ 时不等式严格成立，
那么叫严格凸函数





凸函数优化

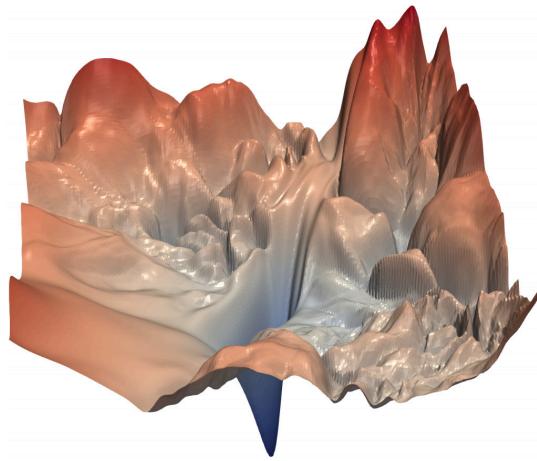
- 如果代价函数 f 是凸的，且限制集合 C 是凸的，那么就是凸优化问题，那么局部最小一定是全局最小
- 严格凸优化问题有唯一的全局最小





凸和非凸例子

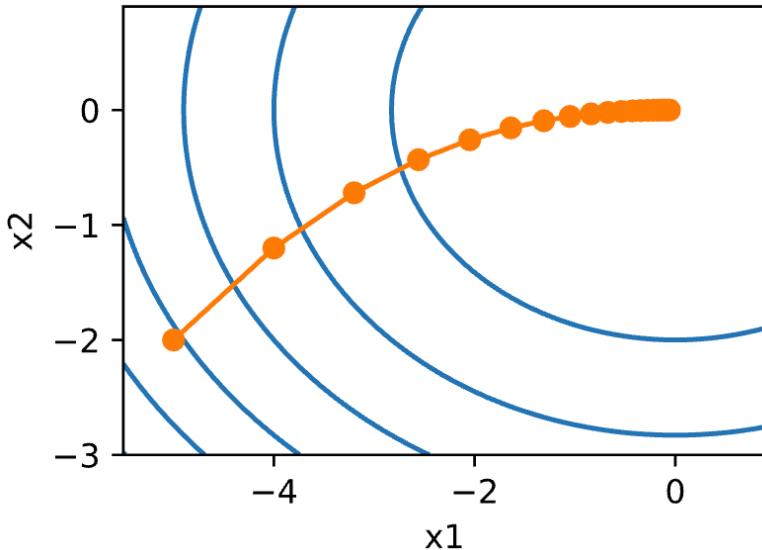
-
- 线性回归 $f(\mathbf{x}) = \|\mathbf{W}\mathbf{x} - \mathbf{b}\|_2^2$
- Softmax 回归
- 非凸：其他
 - MLP, CNN, RNN, attention, ...





梯度下降

- 最简单的迭代求解算法
- 选取开始点 \mathbf{x}_0
- 对 $t = 1, \dots, T$
 - $\mathbf{x}_t = \mathbf{x}_{t-1} - \eta \nabla f(\mathbf{x}_{t-1})$
- η 叫做学习率





随机梯度下降

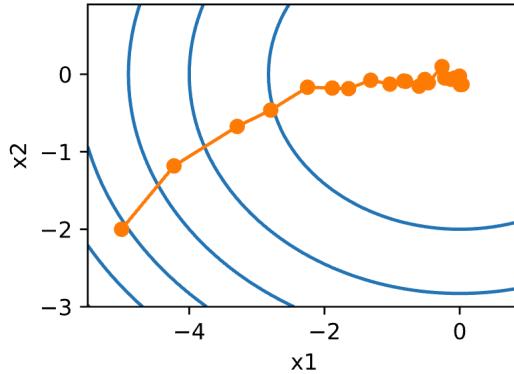
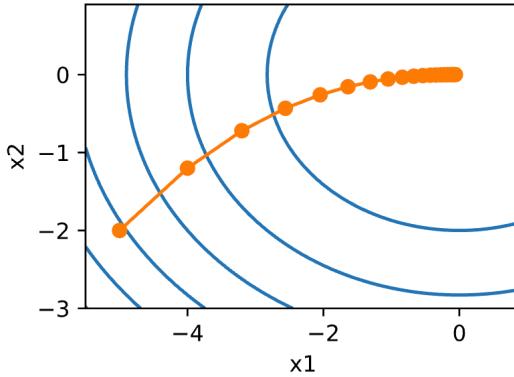
- 有 n 个样本时，计算

$$f(\mathbf{x}) = \frac{1}{n} \sum_{i=0}^n \ell_i(\mathbf{x})$$
 的导数太贵

- 随机梯度下降在时间 t 随机选项样本 t_i 来近似 $f(x)$

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \eta_t \nabla \ell_{t_i}(\mathbf{x}_{t-1})$$

$$\mathbb{E} \left[\nabla \ell_{t_i}(\mathbf{x}) \right] = \mathbb{E} [\nabla f(\mathbf{x})]$$





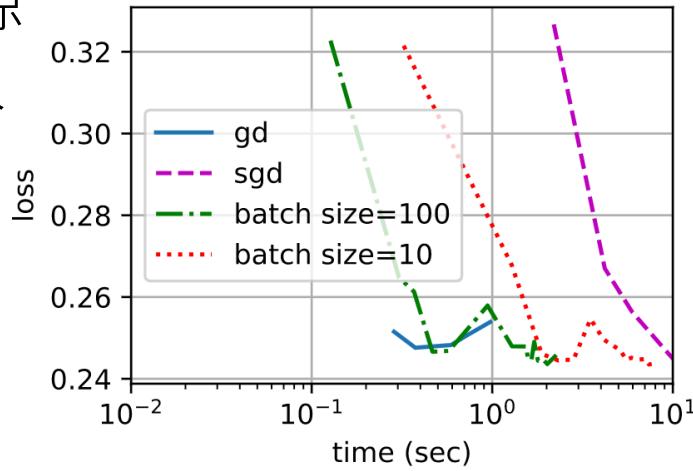
小批量随机梯度下降

- 计算单样本的梯度难完全利用硬件资源
- 小批量随机梯度下降在时间 t 采样一个随机子集 $I_t \subset \{1, \dots, n\}$ 使得 $|I_t| = b$

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \frac{\eta_t}{b} \sum_{i \in I_t} \nabla \ell_i(\mathbf{x}_{t-1})$$

- 同样，这是一个无偏的近似，但降低了方差

$$\mathbb{E}\left[\frac{1}{b} \sum_{i \in I_t} \nabla \ell_i(\mathbf{x})\right] = \nabla f(\mathbf{x})$$



冲量法

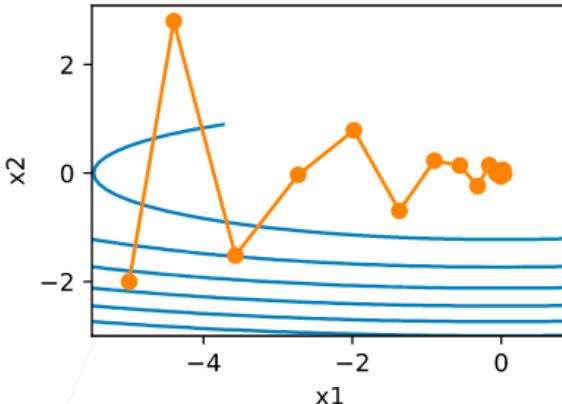
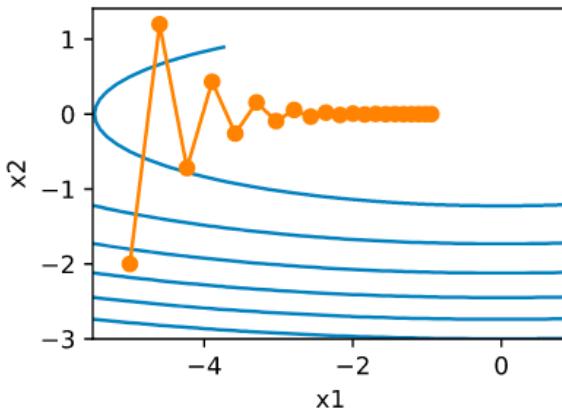


- 冲量法使用平滑过的梯度对权重更新

$$\mathbf{g}_t = \frac{1}{b} \sum_{i \in I_t} \nabla \ell_i(\mathbf{x}_{t-1})$$

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + \mathbf{g}_t \quad \mathbf{w}_t = \mathbf{w}_{t-1} - \eta \mathbf{v}_t$$

梯度平滑： $\mathbf{v}_t = \mathbf{g}_t + \beta \mathbf{g}_{t-1} + \beta^2 \mathbf{g}_{t-2} + \beta^3 \mathbf{g}_{t-3} + \dots$



- β 常见取值 [0.5, 0.9, 0.95, 0.99]

Adam



- 记录 $\mathbf{v}_t = \beta_1 \mathbf{v}_{t-1} + (1 - \beta_1) \mathbf{g}_t$ 通常 $\beta_1 = 0.9$
 - 展开 $\mathbf{v}_t = (1 - \beta_1)(\mathbf{g}_t + \beta_1 \mathbf{g}_{t-1} + \beta_1^2 \mathbf{g}_{t-2} + \beta_1^3 \mathbf{g}_{t-3} + \dots)$
 - 因为 $\sum_{i=0}^{\infty} \beta_1^i = \frac{1}{1 - \beta_1}$, 所以权重和为1
 - 由于 $\mathbf{v}_0 = 0$, 且 $\sum_{i=0}^t \beta_1^i = \frac{1 - \beta_1^t}{1 - \beta_1}$,
- 修正 $\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_1^t}$

Adam



- 类似记录 $\mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$, 通常 $\beta_2 = 0.999$, 且修正

$$\hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1 - \beta_2^t}$$

- 计算重新调整后的梯度 $\mathbf{g}'_t = \frac{\hat{\mathbf{v}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon}$
- 最后更新 $\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \mathbf{g}'_t$



总结

- 深度学习模型大多是非凸
- 小批量随机梯度下降是最常用的优化算法
- 冲量对梯度做平滑
- Adam对梯度做平滑，且对梯度各个维度值做重新调整



动手学深度学习 v2
李沐 · AWS



课程总结和进阶学习



数据

- 从3月到8月一共进行了34节课，共51小时
- 讲了1055页幻灯片（一半是代码），回答了934个问题
- 直播参与同学稳定在一千人以上
 - 倒数第二节课仍有1001，平均观看1.1小时
- 视频录像在B站观看125万次，1万弹幕+评论，收到了608.95贝壳

直播设置 v3





大纲回顾

• 深度学习基础

- 线性回归
- Softmax回归
- 多层感知机
- 模型选择
- 过拟合、欠拟合
- 权重衰退
- Dropout
- 数值稳定性
- 模型初始化和激活函数

• 卷积神经网络

- LeNet
- AlexNet
- VGG
- NiN
- GoogleNet
- ResNet

• 计算机视觉

- 图片增广
- 微调
- R-CNN, SSD, YOLO
- FCN
- 样式迁移

• 循环神经网络

- RNN, GRU, LSTM
- 深层、双向RNN
- Seq2Seq

• 注意力机制

- Seq2seq+attention
- Transformer
- BERT

• 性能

- CPU、GPU、ASIC
- 使用（多）GPU
- 分布式

• 应用

- 房价预测
- 图片分类
- 物体检测
- 语义分割
- BERT
- 样式迁移
- 语言模型
- 机器翻译
- 自然语言推理



还有很多应用、模型没有讲到

- 计算机视觉
 - 人脸识别
 - 体态识别
 - 无人驾驶
 - 图片合成
 - 超分辨率
 - 医学图片
- 自然语言处理
 - 文本分类
 - 文本合成
 - 文本摘要
 - 实体命名识别
- 图神经网络
- 语音识别
- 时序数据
- 玩游戏
- 代码生成
- 音乐
- 推荐系统

斯坦福2021秋季新课：实用机器学习



- 课程地址：<https://c.d2l.ai/stanford-cs329p>
- 关注机器学习落地的技术，是本课的补充

- | | | | |
|--------|--------------------------------------|----------|--------|
| • 基础建模 | • 不正确的假设 | • 性能调优 | • 模型之外 |
| • 数据收集 | • Covariate/
concept/label 偏
移 | • AutoML | • 模型部署 |
| • 数据处理 | • IID以外的数据 (时
序、图) | • 模型蒸馏 | • 公平 |
| • 模型验证 | | • 多模块数据 | |
| • 模型融合 | | | |



个人进阶建议 – 知识

- 可以看朋友圈分享，但更要主动获取信息、建立知识库
- 学会读论文
 - 经典论文需要读懂每一句话
 - 结合代码来了解细节（例如 <https://paperswithcode.com>）
 - 可以看openreview上的评（吐）论（槽）
- 对读过的论文做整理
 - <https://www.bilibili.com/video/BV1nA41157y4>

个人进阶建议 – 实践



- 可以参加竞赛，但注意竞赛跟研究和工业界落地都不一样
 - 竞赛：调最好的参、模型融合
 - 研究：新的模型、调还不错的参
 - 工业界：将应用表达成机器学习问题、收集数据
- 多研究开源代码，跟开发者多交流，积极贡献
 - 你可以从修文档开始

感谢



- 感谢同学们观看、评论、点赞、提问题、参与竞赛
- 感谢机器之心，特别是石东乐、任淑航、李亚洲
- 感谢家里领导和两小兽神
- 最后请大家花5分钟填写课程问卷帮助我们改进
 - <https://jinshuju.net/f/ruh8yz>



下次再见！



第69节 BERT



动手学深度学习
PyTorch版
李沐 · AWS

