



Introduction to Computer Vision Notes

授课教师：王鹤

作者：林晓疏, 梁宇桐, 徐靖

组织：PKU EECS

版本：2025 Spring

声明：请勿用于个人学习外其他用途！



ElegantLATEX Program

个人笔记，如有谬误，欢迎指正！

联系方式：2200012917@stu.pku.edu.cn

目录

第 1 章 Images as Functions	1
1.1 Image Gradient	1
1.2 Filters	2
1.3 2D Discrete Filter	3
1.4 Non-linear Filtering	3
第 2 章 Edge Detection	4
2.1 What is an Edge?	4
2.2 Criteria for Optimal Edge Detection	4
2.3 Non-Maximal Suppression (NMS)	4
2.4 A Simplified Version of NMS	5
2.5 Hysteresis Thresholding	5
第 3 章 Keypoint Detection	6
3.1 The Basic Idea of Harris Corner	6
3.2 Harris Corner	6
3.3 equivariant V.S. invariant	7
3.4 How to prove Harris detector is equivariant?	7
3.5 How to do NMS with corner-response-function?	8
3.6 Scale Invariant Detectors	8
第 4 章 Line Fitting	9
4.1 Least Square Method	9
4.2 RANSAC	9
4.3 RANSAC calculation	9
4.4 Hough Transform	10
第 5 章 Convolutional Neural Network	11
5.1 CNN and MaxPooling	11
5.2 Summary of CNN-based classification networks	11
5.3 Pooling Layer affects the parameter number	11
5.4 Comparison of MLP and CNN	11
第 6 章 CNN Training	13
6.1 Data Preparation	13
6.2 Weight Initialization	13
6.3 Start Optimization	13
6.4 Learning Rate Schedule	15
第 7 章 CNN Training improvement	16
7.1 Underfitting	16
7.2 Some Questions	18
7.3 Overfitting	20

7.4	Summary of Mitigating Overfitting	22
第 8 章 Classification		23
8.1	Nearest Neighbour Classifier	23
8.2	Using CNN for image Classification	23
8.3	Cross Entropy Loss V.S. Accuracy	24
第 9 章 CNNs for Image Classification		25
9.1	Reception Field	25
第 10 章 Segmentation		26
第 11 章 3D Vision		28
11.1	Intrinsics	28
11.2	Extinsics	29
11.3	weak perspective	30
第 12 章 Camera Calibration		32
12.1	Some Problems with Camera	33
第 13 章 Single View Geometry		34
13.1	Transformation in \mathbb{R}^2	34
13.2	Vanishing Points	34
第 14 章 Epipolar Geometry		37
14.1	Epipolar Constrain	37
第 15 章 3D Data		41
15.1	Sensors	41
15.2	Multiple 3D representation	41
15.3	Mesh	42
15.4	Point Cloud	42
15.5	CD vs EMD	43
15.6	Implicit Representation	43
15.7	Deep SDF	44
15.8	NeRF, 3DGS, Convolution on Mesh/Graph	45
第 16 章 3D Deep Learning		46
16.1	Sparse Conv	47
16.2	PointNet++ V.S. Conv	48
16.3	设计一个二维 PointNet	48
第 17 章 Sequential Modeling		49
17.1	Recurrent Neural Network (RNN)	49
17.2	Vanilla RNN	50
17.3	RNN tradeoff	51
17.4	Multilayer RNNs	51
17.5	Vanilla RNN Gradient Flow	52

17.6 LSTM	53
17.7 Gated Recurrent Unit (GRU)	53
17.8 Summary	54
第 18 章 Video analysis	55
18.1 Problems: Videos are Big	55
18.2 Video Classification: Single-Frame CNN	55
18.3 Video Classification: Late Fusion with FC	55
18.4 Video Classification: Late Fusion with Pooling	55
18.5 Video Classification: Early Fusion	56
18.6 Video Classification: 3D CNN	56
18.7 Comparison	56
18.8 C3D: The VGG of 3D CNNs	56
18.9 Modeling Long-Term Temporal Structure	57
18.10 Recurrent convolutional Network	57
第 19 章 Transformer	59
19.1 Attention	59
19.2 Transformer	63
19.3 Self-Supervised Learning (Pre-training)	63
19.4 Large Multi-modal Models	63
第 20 章 Object Detection and Instance Segmentation	65
20.1 任务简介	65
20.2 Object Detection: Single Object	65
20.3 Region-based CNN	66
20.4 Fast R-CNN	67
20.5 Faster R-CNN	67
20.6 two-stage detector and one-stage detector	68
20.7 Evaluation Metrics	69
20.8 YOLO	69
20.9 End-to-End Object Detection with Transformers (DETR)	70
20.10 考什么	70
20.11 Mask R-CNN	70
20.12 3D Object Detection and Instance Segmentaiton	71
第 21 章 Generative Models	72
21.1 Tractable Density Model	72
21.2 VAE	72
21.3 GAN	76
21.4 Diffusion Model	78
第 22 章 Pose and Motion	79
22.1 Beyond Detection: Pose	79
22.2 Euler Angle	79
22.3 Axis Angle	79
22.4 Quaternion	79

第 23 章 Instance-Level 6D Object Pose Estimation	81
23.1 Beyond Object Pose	81
第 24 章 Motion	83
第 25 章 Embodied AI	84
25.1 Embodied AI	84
25.2 Object Grasping	85
25.3 Object Manipulation	85
25.4 Locomotion and Navigation	86
25.5 Embodied Multimodal Large Model	86
第 26 章 Summary of Computer Vision	88
第 A 章 Condition Number	89
附录 B Transformation in \mathbb{R}^n	90
附录 C DOF and rank in essential matrix and fundamental matrix	91
附录 D QR Decomposition	92

第 1 章 Images as Functions

An Image as a function f from \mathbb{R}^2 to \mathbb{R}^M :

- $f(x, y)$ gives the intensity at position (x, y)
- Defined over a rectangle with a finite range: $f : [a, b] \times [c, d] \rightarrow [0, 255]$
- Pixel values:
 - **Grayscale/Intensity:** $[0, 255]$
 - **Color (RGB):** $[R, G, B]$

1.0.0.0.1 Note: Images are usually digital (**discrete**).

1.1 Image Gradient

- Image gradient:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$$

- Gradient magnitude:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2}$$

- Gradient direction: Points in the direction of most rapid intensity change.

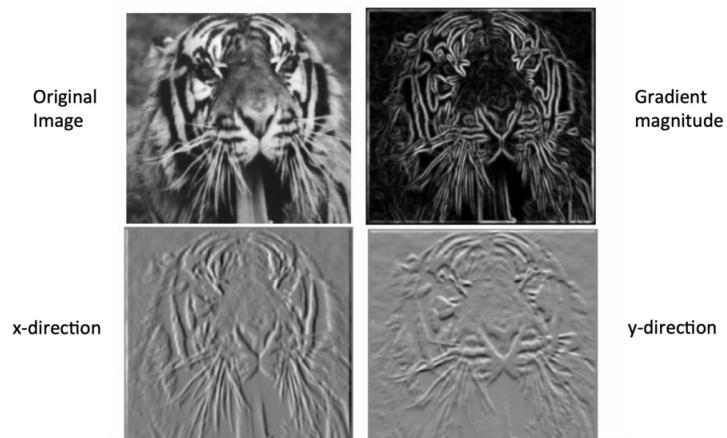


图 1.1: Visualization of image gradient.

1.2 Filters

- **Filtering:** Process to generate a new image by combining original pixel values.

$$f[n] \rightarrow \text{System } \mathcal{G} \rightarrow h[n], \quad h = \mathcal{G}(f), \quad h[n] = \mathcal{G}(f)[n]$$

- **Linear filtering:** A system \mathcal{G} is linear if:

$$\mathcal{G}(\alpha f + \beta g) = \alpha \mathcal{G}(f) + \beta \mathcal{G}(g)$$

- **Discrete convolution:**

$$(f * g)[n] = \sum_{k=-\infty}^{\infty} f[k] \cdot g[n-k]$$

- Key properties:

- **Derivative Theorem:**

$$\frac{d}{dt}(f * g) = f * g'$$

- **Convolution Theorem:**

$$\mathcal{F}(f * g) = \mathcal{F}(f) \cdot \mathcal{F}(g) \quad \Rightarrow \quad h = \mathcal{F}^{-1}(\mathcal{F}(f) \cdot \mathcal{F}(g))$$

Discrete signal

Continuous signal

Convolution	$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$	$(f * g)(x) = \int_{t=-\infty}^{\infty} f(t)g(x-t)dt$
-------------	---	---

Fourier Transform	$\mathcal{F}(f)[n] = \sum_{m=0}^{M-1} f[m] \exp(-\frac{i2\pi}{M} mnt)$	$\mathcal{F}(f) = \int_{t=-\infty}^{\infty} f(t) \exp(-i2\pi\omega t) dt$
-------------------	--	---

图 1.2: Fourier transform visualization.

1.3 2D Discrete Filter

- Example: 2D moving average over a 3×3 window:

$$h[i, j] = \frac{1}{9} \sum_{k=-1}^1 \sum_{l=-1}^1 f[i+k, j+l]$$

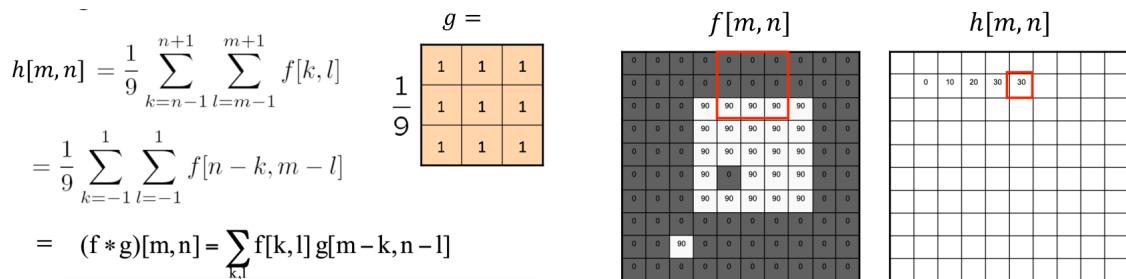


图 1.3: Filtering examples.

1.4 Non-linear Filtering

- Example: **Binarization via Thresholding**:

$$h(x, y) = \begin{cases} 255 & \text{if } f(x, y) > T \\ 0 & \text{otherwise} \end{cases}$$

$$h[m, n] = \begin{cases} 1, & f[n, m] > \tau \\ 0, & \text{otherwise.} \end{cases}$$



图 1.4: Binarization example.

第 2 章 Edge Detection

2.1 What is an Edge?

“边缘”是图像中的一个区域，在这个区域中，沿着图像的一个方向，像素强度值(或者说对比度)发生了“显著”的变化，而在其正交方向上，像素强度值(或对比度)几乎没有变化。

2.2 Criteria for Optimal Edge Detection

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \quad (2.1)$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2.2)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (2.3)$$

Precision 和 Recall 都代表着你检测出的真正边缘所占比例，但是 Precision 的分母是你检测出的边缘，Recall 的分母是真正的边缘。

2.3 Non-Maximal Suppression (NMS)

非最大值抑制，顾名思义，就是抑制非最大值，这里的最大值指的是梯度的局部最大值。

在计算出了所有点的梯度之后，会有很多像素的梯度大于设定的阈值，而我们希望最后得出的边缘像素真的看起来像一条线而不是一块区域，所以 NMS 的目的是为了抑制那些不是边缘的像素，只保留那些是边缘的像素。

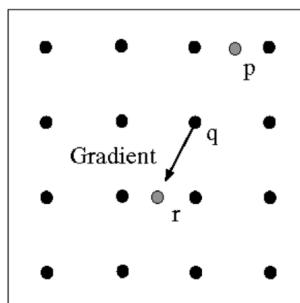


图 2.1: NMS 示意图

对于一个边缘像素的候选点，我们认为它是边缘当：它比它梯度方向的两个点 $q + \nabla q$ 和 $q - \nabla q$ 的梯度值大，也就是这个点的梯度大小是局部最大值的时候。

计算这个点梯度方向的点的梯度值可以使用双线性插值法，就是把这个点周围的四个点的梯度按照横纵距离反比加权。

当然，NMS 是一个思想而不是针对边缘检测的算法，比如对于 keypoint detection, object detection (like YOLO) 都可以使用 NMS，实现的思路都很类似，使用一个打分函数看这个备选点 (bounding box) 是不是比跟它相邻(冲突)的点 (bounding box) 好，如果是就保留，否则就抑制。

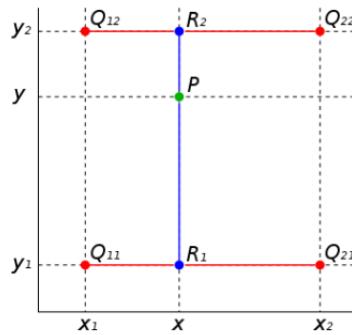


图 2.2: 双线性插值

2.4 A Simplified Version of NMS

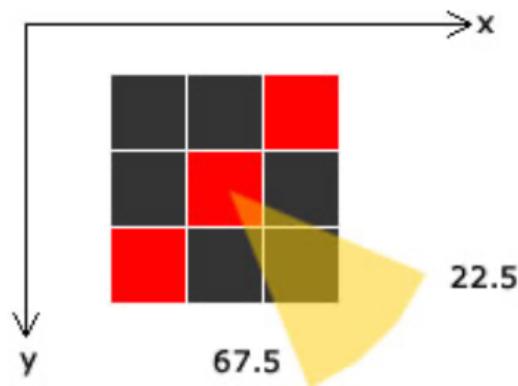


图 2.3: 简化版本的双线性插值

一个 NMS 的简化版本是把双线性插值省去, 直接让这个像素的梯度大于它梯度方向的那两个相邻像素的梯度.

2.5 Hysteresis Thresholding

使用高阈值 (maxVal) 开始边缘曲线, 使用低阈值 (minVal) 继续它们.

- Pixels with gradient magnitudes $> \text{maxVal}$ should be reserved.
- Pixels with gradient magnitudes $< \text{minVal}$ should be removed.

How to decide maxVal and minVal? Examples:

- $\text{maxVal} = 0.3 \times \text{average magnitude of the pixels that pass NMS}$
- $\text{minVal} = 0.1 \times \text{average magnitude of the pixels that pass NMS}$

第 3 章 Keypoint Detection

3.1 The Basic Idea of Harris Corner

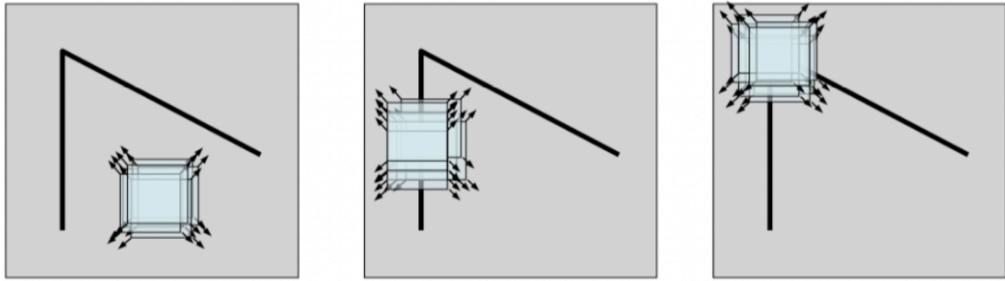


图 3.1: 移动窗口

Move a window and explore intensity changes within the window.

Corner: significant change in all directions.

3.2 Harris Corner

一个 window, 给定它的移动方向 (u, v) :

$$\begin{aligned} E(u, v) &= \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2 \\ &\approx \sum_{x,y} w(x, y)[I(x, y) + uI_x + vI_y - I(x, y)]^2 \\ &= \sum_{x,y} w(x, y)[uI_x + vI_y]^2 \\ &= w * \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \\ &= \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} w * I_x^2 & w * I_x I_y \\ w * I_x I_y & w * I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \\ &= \begin{bmatrix} u & v \end{bmatrix} R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R \begin{bmatrix} u \\ v \end{bmatrix} \\ &= \lambda_1 u_R^2 + \lambda_2 v_R^2 \end{aligned} \tag{3.1}$$

根据这两个特征值的大小可以判断这个点是不是角点.

这个点是角点一般需要满足:

- $\lambda_1, \lambda_2 > b$
- $\frac{1}{k} < \frac{\lambda_1}{\lambda_2} < k$

一个快速的判断公式:

$$\begin{aligned} \theta &= \frac{1}{2}(\lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2) + \frac{1}{2}(\lambda_1 \lambda_2 - 2t) \\ &= \lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2 - t \\ &= \det(R) - \alpha \text{Trace}(R)^2 - t \end{aligned} \tag{3.2}$$

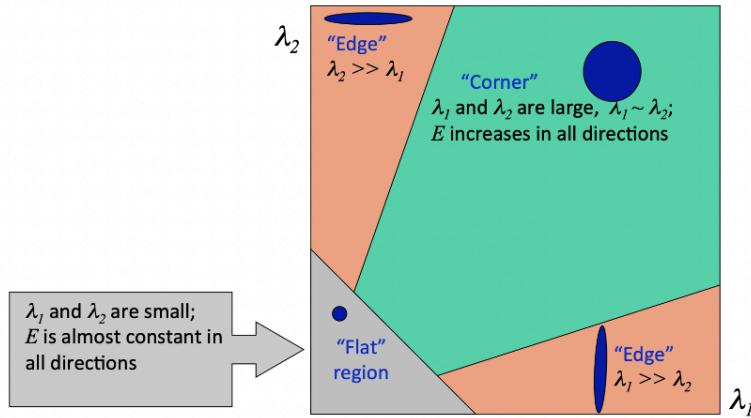


图 3.2: 特征值大小和这个点的是什么种类的点的关系

其中 $\alpha \in [0.04, 0.06]$, $t \in [0.01, 0.03]$.

Harris Corner 对平移和图像旋转是 equivariant 的, 对规模不是 equivariant 的.

3.3 equivariant V.S. invariant

等变 (equivariant): $F(TX) = T(F(x))$, 对于 translation 和 rotation 是等变的.

不变 (invariant): $F(T(X)) = F(X)$, 也就是对于不同位置导出的角点还是那样, 所以其实我们想要的是等变, 也就是对于不同位置导出的角点做了同样的变化.

3.4 How to prove Harris detector is equivariant?

只要说明角点检测函数也是 equivariant 即可.

角点检测函数包括了求导和卷积两个操作, 显然求导是 equivariant 的, 因为导数会随着 trans 和 rot 做相同的变化.

很有趣的是卷积也是 equivariant 的: 当你的 filter function 是各向同性的, 那么这个卷积就是 equivariant 的; 但是如果是一个椭圆形的 window, 那这个卷积就不是 equivariant 的了.



图 3.3: Illumination invariant

这个高光说明不是环境 Illumination Invariant 的

3.5 How to do NMS with corner-response-function?

一个简单的想法:

先给出一个阈值, 把所有 response 排序, 成为一个 list, 从上到下按顺序把这个 pixel 周围的大于阈值的踢出 list. 这个跟之前的 NMS 区别在于之前需要一条边, 现在只需要一个点, 那么现在比之前踢出的像素点更多.

3.6 Scale Invariant Detectors

Harris-Laplacian, SIFT (Lowe)

第 4 章 Line Fitting

4.1 Least Square Method

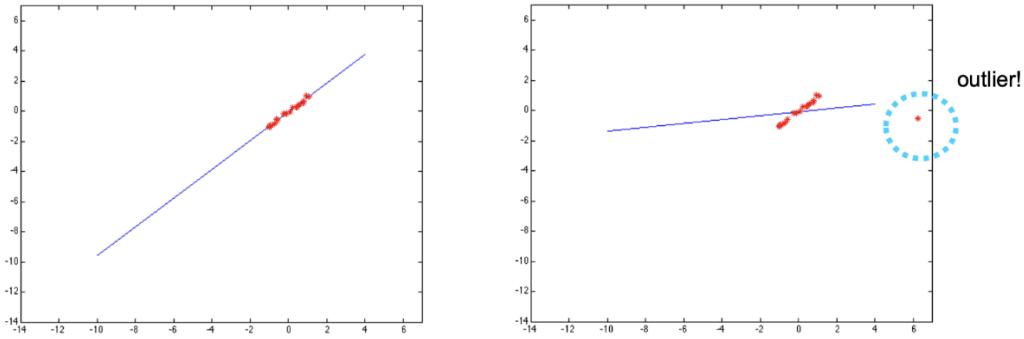


图 4.1: Least Square Method is not robust to outliers

Robust to small noises but sensitive to outliers.

4.2 RANSAC

RANSAC: RANdom SAmple Consensus

Idea: we need to find a line that has the largest supporters (or inliers)

RANSAC loop:

假设这个直线(平面)需要两个(n 个)点来确定。

1. 随机选择 k 组能确定这个直线的点, 也就是在所有点里面选出一个 $k \times 2$ 的矩阵
2. 对每一组点计算出一条直线
3. 对每一组点的直线计算出所有点到这条直线的距离, 如果小于阈值, 则认为这个点是这条直线的 inlier
4. 找到最大的 inlier 数量的直线, 如果大于阈值, 则认为这条直线是最优的
5. 对这个最优的直线, 用这个直线所有的 inlier 重新计算一次直线

4.3 RANSAC calculation

假设我们有所有 inliner 占比为 w 的先验知识, 同时希望有不低于 p 的概率能够找到一个最优的直线, 那么我们需要多少次迭代呢?

$$\Pr[\text{一组点全部是 inliner}] = w^n \quad (4.1)$$

如果一组点中有一个点是 outlier, 那么我们称这组点 fail.

$$\Pr[k \text{ 组点全部 fail}] = (1 - w^n)^k \quad (4.2)$$

我们希望 k 组点全部 fail 的概率小于 $1 - p$.

$$(1 - w^n)^k < 1 - p \Rightarrow k > \frac{\log(1 - p)}{\log(1 - w^n)} \quad (4.3)$$

4.4 Hough Transform

其实就是把一条直线从实际空间的表示转换到参数空间的表示. 但是如果存在垂直的直线, 可能需要考虑使用极坐标来作为参数空间.

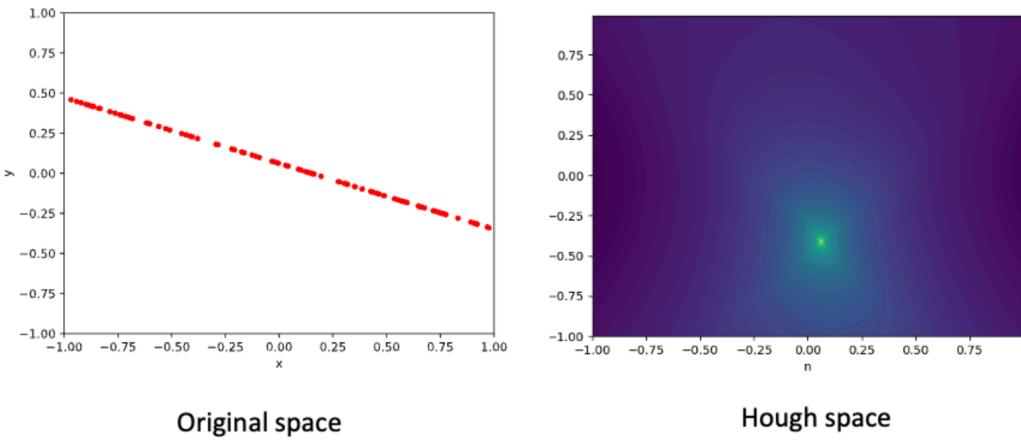


图 4.2: Hough Transform w/o Noise

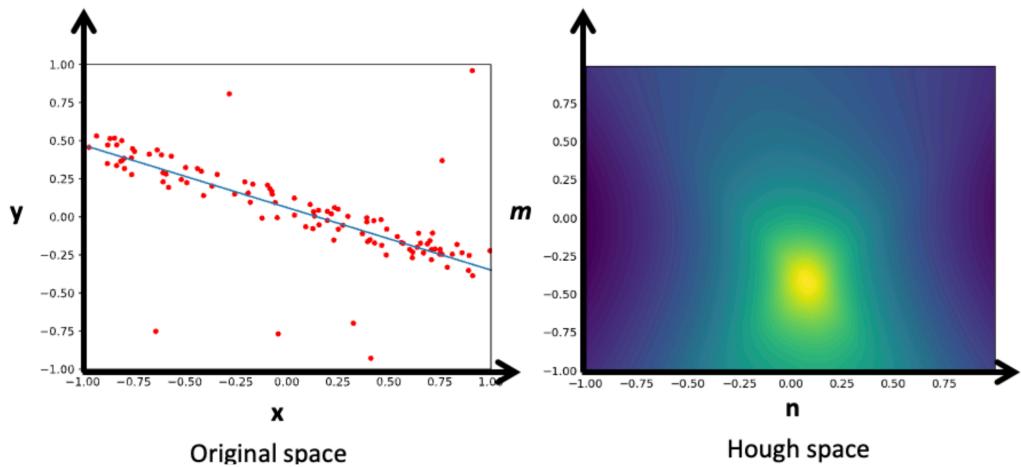


图 4.3: Hough Transform w/ Noise and Outliers

第 5 章 Convolutional Neural Network

5.1 CNN and MaxPooling

假定输入是 $W_1 \times H_1 \times C$ 的矩阵, 可以看作一张图片的宽度, 高度, 通道数. 卷积层需要四个超参数: filter 的数量 K 和大小 F , 步长 S 和零填充参数 P . 经过卷积层后, 原本的输入变成 $W_2 \times H_2 \times K$, 其中

$$\begin{aligned} W_2 &= \frac{W_1 - F + 2P}{S} + 1 \\ H_2 &= \frac{H_1 - F + 2P}{S} + 1 \end{aligned} \tag{5.1}$$

一共需要 $F^2CK + K$ 个参数, 其中额外的 K 是每层的 bias.

选用卷积核, 可以降低 feature map 的大小. 如果直接将图片进行 flatten, 将会导致大量信息的丢失.

所谓池化操作, 就是一种降采样, 可以降低图片的大小. 比如采用 2×2 的大小进行最大值池化, 就是将每个 2×2 范围内最大的像素的值作为池化后这个像素的取值, 将原来的图片缩减到四分之一大小. 池化可以增大感受野. 虽然可以通过步长大于一的卷积来代替池化, 但是池化不需要参数, 更容易优化.

假定输入是 $W_1 \times H_1 \times C$ 的矩阵, 则池化层需要两个超参数: 大小 F 和步长 S . 池化结果是产生 $W_2 \times H_2 \times K$ 的矩阵, 其中

$$\begin{aligned} W_2 &= \frac{W_1 - F}{S} + 1 \\ H_2 &= \frac{H_1 - F}{S} + 1 \end{aligned} \tag{5.2}$$

池化层无需参数.

5.2 Summary of CNN-based classification networks

ConvNets 堆叠了卷积层, 池化层和全连接层, 倾向于使用更小的卷积核和更深的网络结构, 尽量避免使用池化层和全连接层(只使用 Conv 层). 其网络结构大致如下:

$[(\text{Conv-ReLu}) * N - \text{Pool?}] * M - (\text{FC-ReLu}) * K - \text{SoftMax}$

其中 N 一般不超过 5, M 比较大, $0 \leq K \leq 2$. 但最近的网络如 ResNet, GoogleNet 等也开始突破这些范围.

5.3 Pooling Layer affects the parameter number

三层 3×3 比 7×7 多了两个 relu, 非线性性质更好并且参数变少

每 MaxPooling 一次, 长宽减半, Channel 数量变成两倍

$Param = k^2C^2$, 参数变成四倍

$Mem = mnC$, 显存变成二分之一

5.4 Comparison of MLP and CNN

如果输入为 $W_1 \times H_1 \times C$ 的矩阵, 输出 $W_2 \times H_2 \times K$ 的矩阵, 那么 FC 需要 $W_1 W_2 H_1 H_2 C K$ 个参数, 一层卷积核大小为 F 的 CNN 需要 $F^2 C K$ 个参数. 后者一般比前者小得多.

对于一维情形, $h \in \mathbb{R}^m, x \in \mathbb{R}^n$, 则 $y = h * x$ 可以被表示为矩阵乘法:

$$y = h * x = \begin{pmatrix} h_1 & 0 & \cdots & 0 & 0 \\ h_2 & h_1 & & \vdots & \vdots \\ h_3 & h_2 & \cdots & 0 & 0 \\ \vdots & h_3 & \cdots & h_1 & 0 \\ h_{m-1} & \vdots & \ddots & h_2 & h_1 \\ h_m & h_{m-1} & & \vdots & h_2 \\ 0 & h_m & \ddots & h_{m-2} & \vdots \\ 0 & 0 & \cdots & h_{m-1} & h_{m-2} \\ \vdots & \vdots & & h_m & h_{m-1} \\ \vdots & \vdots & \cdots & 0 & h_m \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \quad (5.3)$$

左侧的矩阵被称为 Toeplitz 矩阵. 由于深度学习当中的卷积参数是学习而来, 故不需要像传统的卷积操作一样进行翻转. 二维情形的卷积则是一个 double block circulant matrix.

由于我们选取较小的卷积核, 所以前后层网络之间的连接更为稀疏. 而且有更加明显的 Parameter Sharing 效应, 即对每块区域采取相同的参数进行处理.

FC 和 CNN 两者哪个表达能力更强呢?

很显然是 FC 更强, 因为 FC 的表达范围是 CNN 的超集. 但事实上, 使用 Conv 的结果远好于 FC/MLP. 那么问题出在哪里呢?

一个显而易见的答案是, FC 需要的参数量过于庞大了, 一层可能需要上亿甚至更多的参数, 这使得其非常难以被优化. 另外一个非常重要的原因是, 它并没有突出我们在视觉任务中目标的特点, 即等变性 (equivariance).

我们知道, 在目标分类等任务当中, 将图片进行轻微平移, 旋转, 改变亮度等操作, 并不会影响结果. 但是, 它们都改变了输入的几乎每一个值. 而即使移动了一个像素, FC 的输出都将天差地别. 换言之, 我们要求 FC 将在它看来输出完全不同的图像归为一类, 这样的矩阵是非常难以寻找的, 会在优化过程中产生极大的困难.

那么 CNN 的表现如何呢? 我们先解释上文 equivariance 的含义. 其一般定义为:

$$S_A[\phi(X)] = \phi[T_A(X)] \quad (5.4)$$

这里 A 指代某种操作, 而 T_A, S_A 分别代表 $X, \phi(X)$ 空间下的变换. 例如将 A 看作左移一像素, 忽略边界就有 $T[\phi(X)] = \phi[T(X)]$. 不变性是等变性在 $S_A = I$ 的特殊情形.

我们需要指出的是, 在这里 Parameter Sharing 就等同于 Equivariance to Translation. 我们用同样的参数处理每一个局部, 那么忽略掉边界后, 2D Conv 就是等变的.

举个例子, 当处理图像的时候, 在第一个卷积层进行边的探测是非常有用的, 而同样的边或多或少地会出现在图片的其他位置, 所以在整个图片范围内进行参数共享是非常可行的操作. 换言之, 对于反映同种信息的 pixel 的识别, 在不同位置应该是相同的, 这种人类视觉给出的先验知识才是 Conv 的根本思路.

但需要强调的是, Conv 并不是万能的, 一个 layer 作为一个函数, 其需要具有何种性质与目标的性质高度相关. 我们使用 CNN 正是因为当前我们的目标具有这种不变性. 但有时你需要位置相关的处理, 比如统计一张照片上的绵羊数量, 那显然就不能将不同地方的绵羊做相同处理. 这方面的工作就是 semi-Conv 的工作, 发表于 CVPR 2018. 同样, 它也不具有图像缩放, 图像旋转等情形下的不变性, 需要引入其他机制进行处理.

第 6 章 CNN Training

6.1 Data Preparation

不仅是在 CV 当中, 在诸如数据科学等学科之中, 对数据进行分析之前通常要进行预处理 (processing). 以处理 CIFAR-10 的 $32 \times 32 \times 3$ 数据集, 我们有逐通道减去均值 (VGG) 或逐通道减去均值然后除以标准差 (ResNet).

标准化有什么作用呢? 例如, 如果神经元的输入都是正数, 那么权重的导数就会都是正的或者负的. 而非标准化的数据可能具有很大的数值, 对于权重矩阵的微小变化非常敏感, 难以训练.

6.2 Weight Initialization

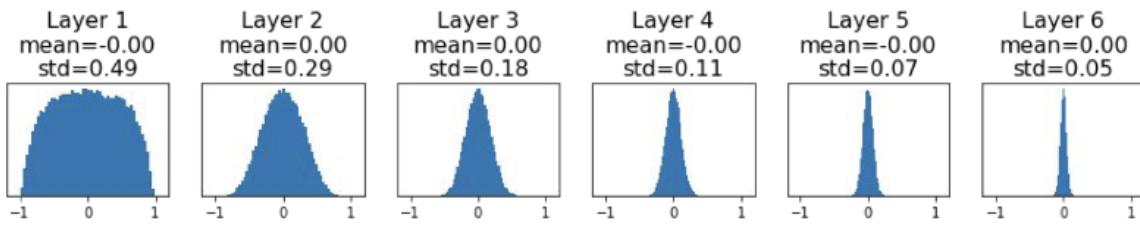


图 6.1: 梯度方差示意图

权重矩阵的初始化并不是一件随意的事情, 它代表了我们优化的起点. 一个最简单的想法是: 用一些小的零均值随机数, 比如调用 $0.01 * np.random.randn(Din, Dout)$ 生成参数矩阵. 但是这样做, 由于参数都是一些服从高斯分布的随机数, 这会导致梯度的方差逐层递减, 最后梯度非常小, 训练困难, 如图所示6.1.

一个优化方法是每层都进行标准化, 同时令参数的方差为 $\frac{1}{D_{in}}$.

数学推导:

假设 $y = x_1w_1 + \dots + x_nw_n$, 我们希望 $Var(y) = Var(x_i)$, 这样可以保证梯度的方差不会随着层数的增加而变小. 那么:

$$\begin{aligned} Var(y) &= Var(x_1w_1 + \dots + x_nw_n) \\ &= Var(x_1w_1) + \dots + Var(x_nw_n) \\ &= Var(x_1)Var(w_1) + \dots + Var(x_n)Var(w_n) \\ &= nVar(x)Var(w) \\ \Rightarrow Var(w) &= \frac{1}{n} \end{aligned} \tag{6.1}$$

在使用 ReLu 时, 可以视为去掉一半神经元, 参数方差为 $\sqrt{\frac{2}{D_{in}}}$.

6.3 Start Optimization

前面我们提到过用 SGD 的方法进行训练, 但在矩阵的 Condition Number¹ 较大的时候, SGD 的效果仍然不够好. 对 SGD 来说, 山谷和鞍点仍然是巨大的麻烦. 所谓山谷, 就是类似右图的地形²:

如果梯度下降进入如图所示的地形, 则由于山谷两侧的梯度指向谷底, 因此如果山谷中比较平缓, 则很可能在两侧山壁来回打转, 或需要很长时间才能到达谷底的最小值.

¹有关矩阵 Condition Number 的简单叙述, 请见附录.

²本小节的配图和叙述参考了 @ 郑思座的[这篇文章](#). 此文还介绍了牛顿动量法和自然梯度法.

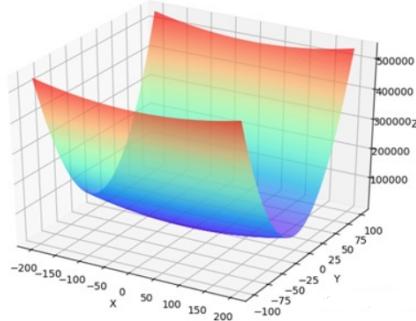


图 6.2: 山谷示意图

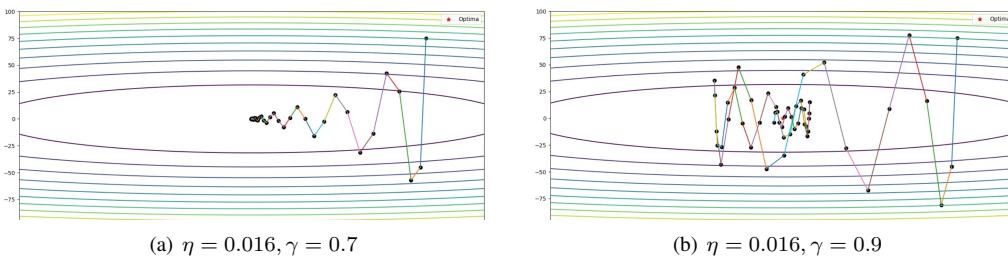
因此,在 SGD 的基础上我们可以添加动量方法 (Momentum)³, 所谓动量方法, 就是对历史梯度进行记录, 并以一定权重影响当前的梯度.

具体来说, 结合物理学上的动量思想, 在梯度下降问题中引入动量项 m 和折扣因子 γ , 公式变为

$$\begin{aligned} m &\leftarrow \gamma m + \eta \nabla_{\theta} \mathcal{L}(\theta) \\ \theta &\leftarrow \theta - m \end{aligned} \tag{6.2}$$

其中, γ 表示历史梯度的影响力, γ 越大, 历史梯度对现在的影响也越大. 直观上来说, 要是当前时刻的梯度与历史梯度方向趋近, 这种趋势会在当前时刻加强, 否则当前时刻的梯度方向减弱.

我们用 SGD+Momentum 来对图6.2进行梯度下降. 下面分别是两个不同参数情形时的俯视图. 如果不加 momentum, 则可以想见点将会以近乎垂直于等高线的方向来回震荡. 左图中每个较长的梯度下降之后, 跟着的一段较短并且一定程度矫正了方向, 这正是历史梯度的作用. 而右图则是 γ 较大的情形, 此时历史梯度的权重过大, 积重难返, 优化效果要差一些.



再来考察另一个麻烦: 鞍点. 存在鞍点意味着此处的 Hessian Matrix 有正负特征值, 是不定的, 因而随机梯度下降很可能无法找到下降的方向. 由于深度学习实践中普遍维度较高, 比如在 1000 维当中只有两个维度不是 local minimum, 那么 SGD 也很难跳出这里. 这也是偶尔会看到 loss 长时间不动而突然间骤减的可能原因.

若用 G_t 表示第 t 轮迭代的动量, $g_t = \eta \nabla_{\theta} \mathcal{L}(\theta)$ 表示第 t 轮迭代的更新量, 当 $t \rightarrow \infty$, $G_{\infty} = \frac{g_0}{1-\gamma}$, 该式的含义是如果梯度保持不变, 最终的更新速度会是梯度项乘以学习率的 $\frac{1}{1-\gamma}$ 倍. 举例来说, $\gamma = 0.9$ 时, 动量算法最终的更新速度是普通梯度下降法的 10 倍, 意味着在穿越“平原”和“平缓山谷”从而摆脱局部最小值时, 动量法更有优势.

在书写代码时, 我们通常这样写:

$$\begin{aligned} v_{t+1} &\leftarrow \rho v_t + \nabla f(x_t) \\ x_{t+1} &\leftarrow x_t - \alpha v_{t+1} \end{aligned} \tag{6.3}$$

也就是将 velocity 项作为梯度的运行时均值. 一般取 $\rho = 0.9$ 或 $\rho = 0.99$.

Adam 就是一种应用了 Momentum 的优化方法. 具体如下图.

³为什么动量优化 work? 可以参考这个可视化可交互网站: [Why Momentum Really Works](#)

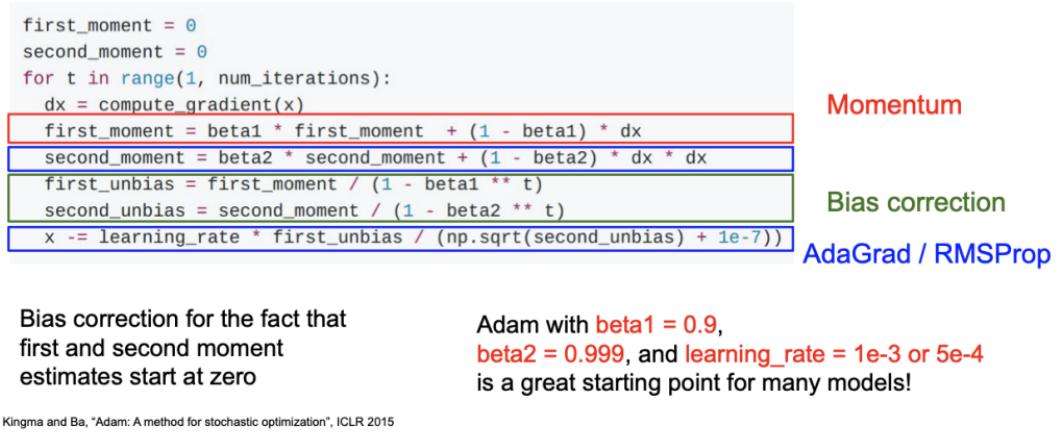


图 6.3: Adam

6.4 Learning Rate Schedule

在神经网络的训练过程中,如何设定学习率的变化策略也是很重要的,因为越靠近最小值,其对学习率的取值可能越敏感,需要更加精细地取值. 我们先来看一张不同大小的学习率对于 loss function 的影响. 由此可以看出,好的学习率就应

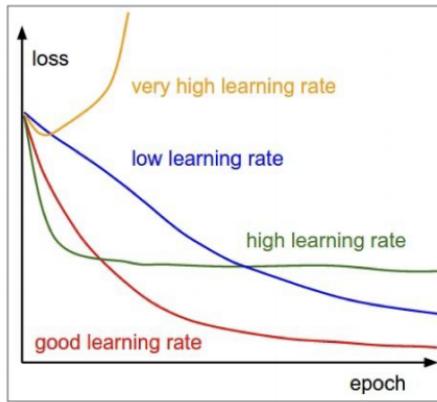


图 6.4: 不同学习率对损失函数的影响

较低的学习率会使得训练所需时间较长,而过高的学习率会限制最终的结果. 如果学习率非常高,可能 loss 反而会上升⁴.

一种策略是,在固定轮数之后,缩减学习率. 比如 ResNet 在每 30 个 epoch⁵后将学习率乘以 0.1. 除此之外还有多种形式,如余弦,线性递减,除以轮数的平方根等.

除了一定轮数之后的递减策略,在训练伊始,也有“热身”的递增策略. 过高的初始学习率对优化会产生不利影响,但在约前 5000 次迭代中线性增长学习率可以避免这种影响. 另外,有一个经验定律: 如果将 batch 的大小增长了 N ,那么学习率也变为 N 倍.

总之,对于学习率策略而言,Adam+ 默认学习率在多数情形下都是一个默认较好的选择,甚至它对于常数学习率也常常运行得不错. 而 SGD+Momentum 的组合可以取得比 Adam 更好的效果,但恐怕要在学习率策略上花费更多头发. (余弦函数是个不错的较少参数的策略.)

⁴这里老师在课上说不一定,并提问:对于 classification 问题,也会上升吗? 有想法的同学欢迎联系笔者,我将在此添加您的真知灼见. 两年之后的笔者对这个问题也没有想法 orz

⁵此处王鹤老师详细解释了 iteration 和 epoch 两个概念的区别. 一次 iteration, 就是跑完一次 mini-batch 的过程. 而 epoch 的含义则更加灵活. 它可指代:1. 训练完整个 dataset 称为一个 epoch. 例如将 3200 张图片分为 100 个 mini-batch, 则 1 epoch = 100 iteration. 2. 每个 epoch 结束保存一次模型. 3. epoch 作为绘制 curve 的单元. 4. 进行 model validation 的单元.

第 7 章 CNN Training improvement

7.1 Underfitting

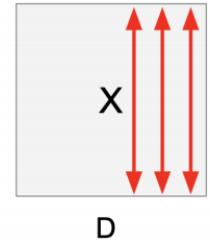
Batch Normalization

所谓 Batch Normalization, 中文译名为“批标准化”. 其基本操作是在进入激活函数层之前, 将数据逐通道进行标准化. 假设输入为 n 个数据, 通道数为 D , 如右图.

那么我们需要计算

$$\begin{aligned}\mu_j &= \frac{1}{N} \sum_{i=1}^N x_{i,j} \\ \sigma_j^2 &= \frac{1}{N} (x_{i,j} - \mu_j)^2 \\ \hat{x}_{i,j} &= \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}\end{aligned}\tag{7.1}$$

Input: $x : N \times D$



其中 ϵ 是为防止方差为 0 而添加的小量. 这一步是标准化的过程, 此层还需要进行一步处理:

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j\tag{7.2}$$

图 7.1: Input of Batch Normalization

我们需要学习 γ, β 这两个 D 维参数.

那么, 为什么要进行 BN 操作呢?

在机器学习领域有个很重要的假设: 独立同分布假设, 就是假设训练数据和测试数据是满足相同分布的, 这是通过训练数据获得的模型能够在测试集获得好的效果的一个基本保障. 那 BatchNorm 的作用是什么呢? BatchNorm 的作用就是在深度神经网络训练过程中使得每一层神经网络的输入保持相同分布.

对于深度学习这种包含很多隐层的网络结构, 在训练过程中, 因为各层参数不停在变化, 所以每个隐层都会面临 covariate shift 的问题, 也就是在训练过程中, 隐层的输入分布经常发生变化, 这就是所谓的 “Internal Covariate Shift”, Internal 指的是深层网络的隐层, 是发生在网络内部的事情.

BatchNorm 的基本思想就是: 能不能让每个隐层节点的激活输入分布固定下来呢? 这样就避免了 “Internal Covariate Shift” 问题了.

之前的研究表明如果在图像处理中对输入图像进行白化操作的话——所谓白化, 就是对输入数据分布进行 PCA, 并将各个分量变换到 0 均值, 单位方差的正态分布——那么神经网络会较快收敛. 但是白化操作较难进行, 而且操作不可导. 而 BN 可以理解为对深层神经网络每个隐层神经元的激活值做简化版本的白化操作.

BN 的基本思想其实相当直观: 因为深层神经网络在做非线性变换前的激活输入值随着网络深度加深或者在训练过程中, 其分布逐渐发生偏移或者变动, 之所以训练收敛慢, 一般是整体分布逐渐往非线性函数的取值区间的上下限两端靠近 (对于 Sigmoid 函数来说, 意味着激活输入值是大的负值或正值), 所以这导致反向传播时低层神经网络的梯度消失, 这是训练深层神经网络收敛越来越慢的本质原因, 而 BN 就是通过一定的规范化手段, 把每层神经网络任意神经元这个输入值的分布强行拉回到均值为 0 方差为 1 的标准正态分布, 其实就是把越来越偏的分布强制拉回比较标准的分布, 这样使得激活输入值落在非线性函数对输入比较敏感的区域, 这样输入的小变化就会导致损失函数较大的变化, 意思是这样让梯度变大, 避免梯度消失问题产生, 而且梯度变大意味着学习收敛速度快, 能大大加快训练速度.

对于每个隐层神经元, 把逐渐向非线性函数映射后向取值区间极限饱和区靠拢的输入分布强制拉回到均值为 0 方差为 1 的比较标准的正态分布, 使得非线性变换函数的输入值落入对输入比较敏感的区域, 以此避免梯度消失问题.

以上关于 BN 基本思想的叙述来自参考文献 [BN]. 但是在 BN 原论文 [BNpaper] 当中, 作者也提出了只进

行 7.1 的问题: 它可能限制模型的表达能力. 以 Sigmoid 为例, 标准化将所有的输入集中于 0 附近, 而这里正是 Sigmoid 的线性近似区域, 这样 Sigmoid 层的作用就和线性层类似了. 因此 BN 需要再加一项平移和拉伸, 并把这两项参数作为学习对象.

总之, BatchNorm 的提出试图通过标准化解决深层神经网络当中内部协变量漂移的问题, 将数据限制在一定范围内, 并通过平移和拉伸操作减少此操作对模型表达能力的影响. 但是, 后来有人提出了新的看法: BatchNorm 对 Internal Covariate Shift 的减缓微乎其微, 它是通过平滑化 loss landscape 来起到作用的.

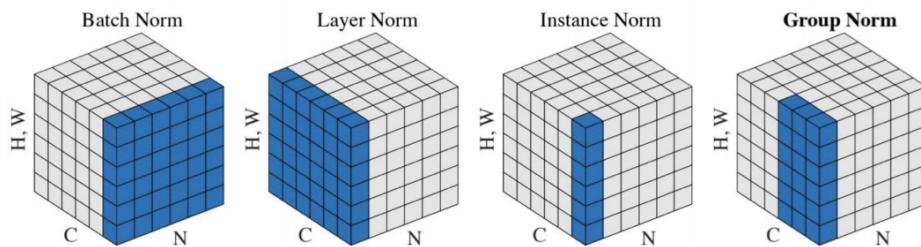
但 BatchNorm 有一点需要额外注意: 那就是在测试集上运行时, BatchNorm 层并不是计算输入的期望方差, 而是调用在训练时储存的 μ, σ . 具体来说, 按照以下方式迭代每次运算得到的 μ_{rms} :

$$\begin{cases} \mu'_{rms} = \rho\mu_{rms} + (1 - \rho)\mu_t \\ \sigma'_{rms} = \rho\sigma_{rms} + (1 - \rho)\sigma_t \end{cases} \quad (7.3)$$

BatchNorm 通常位于 FC 层或 Conv 层之后, 非线性层之前. 它有如下的优点:

1. 使得深层网络更加容易训练, 因为它可以减缓梯度消失.
2. 允许使用更大的学习率, 并且收敛更快. 这是因为如果不加 BatchNorm, 则深层网络容易产生级联效应, 学习率如果比较大, 则前面的 layer 发生变化将会引起后面的层的剧烈变化, 极不稳定.
3. 网络对于参数的初始化要求降低, 因为会进行标准化操作.
4. 训练的时候使用 BatchNorm 具有正则化的特性: 这是因为进行了噪声注入, BatchNorm 在每个批次上计算均值和方差, 这意味着每个批次的统计特征都会略有不同, 这种基于小批量的统计计算引入了噪声, 类似于数据增强或 Dropout 的效果. 这种噪声可以帮助模型避免过拟合, 在一定程度上增强了模型的泛化能力.
5. 测试时不花费开销, 可与 Conv 一同使用

但是同时需要注意: BatchNorm 在训练和测试时行为不同! 这是 Bug 的一个常见原因. 此外, 一旦使用了 BatchNorm, 那么 Batch 就不能太小, 否则会使 μ, σ 变得极不稳定, 这可能会导致模型在训练集和测试集上的性能的巨大差异.



Wu and He, "Group Normalization", ECCV 2018

图 7.2: Normalization Techniques

这样的特性使得人们自然而然提出了一个问题: 我们能否绕过 BatchSize 的限制? 如果可行, 那么就不会存在训练和测试的差异.

图 7.2 当中的 C, N 分别代表通道数和样本数, H, W 等维度因为地位相同, 被压缩为一个维度, 可以视为图片的宽和高. 左一逐通道对所有样本求平均值, 正是 BatchNorm. 要想脱离 BatchSize 的限制, 就必须不沿 N 方向处理.

左二是对 C, H, W 三个维度求平均, 这其实蕴含了所有数据服从同一分布的假设, 但实际上可能并非如此. 比如, 偏绿色的图片在 G 通道上的 μ 更高, 这样做会破坏不同 channel 的差异性, 效果比 BatchNorm 更差, 因此极少应用于 CV 领域, 但在 NLP 领域应用广泛.

右二其实就对应了 BatchSize=1 的情形, 其问题前文已述. 虽然其存在巨大的不稳定性, 但其 μ, σ 表征了图片风格, 因此在 Style Transfer 中常用.

右一由何恺明等人提出的 GroupNorm, 它将 channel 分组进行计算, 相当于对图 2 和图 3 进行了插值.

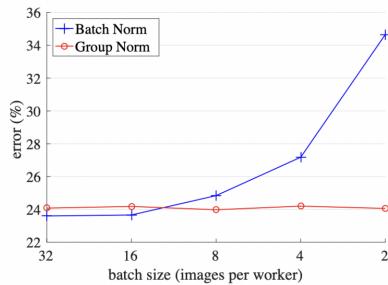


图 7.3: classification error vs batch sizes

图 7.3 是 He 在论文中给出的一个 ResNet-50 模型的训练结果, 可以看出在 BatchSize 较大的时候, BatchNorm 表现略优于 GroupNorm, 但随着 BatchSize 减小, 前者的错误率快速增长, 而后者的错误率则没有明显变化. 当训练集的 BatchSize 不得不很小或 Batch 之间差异极大时, 应该选用 GroupNorm.

7.2 Some Questions

问题: 为什么 BatchNorm 不是对 N 上的一维数组进行 norm?

这是因为会破坏 translation invariance, 也就是平移一格变化很大.

问题: 后面三个 Norm 需要进行 moving average 吗?

跟 BN 不同, 后面三个在 train 和 test 上都是一样的, 没有 moving avg mean, 因为没有对 batch size 做 norm.

为了防止 test 的时候 batch size 过小, BN 需要在 test 的时候使用 train 的 moving avg mean. 而后面三个 norm 没有这个问题, 因为它们不是对 batch size 做 norm 的.

ResNet or Skip Links

介绍 BatchNorm 之后, CNN 网络结构可修改如下:

$[(\text{Conv}-\text{BN}-\text{ReLU}) * N - \text{Pool?}] * M - (\text{FC}-\text{BN}-\text{ReLU}) * K - \text{FC} - \text{SoftMax}$

注意最后的 FC-SoftMax 之间无需添加 BN. 那么, 层数是越深越好吗? 我们来看下图.



图 7.4: 两种深度的神经网络的测试误差 (左) 和训练误差 (右)

56 层神经网络在测试集上误差大于 20 层, 我们可以解释为是过拟合导致, 但其在训练集上的误差竟然也更大, 这就说明更深的神经网络表现不佳并不是过拟合导致. 那么在 CNN 变深的过程中, 究竟产生了什么问题呢?

现在的事实是: 更深的模型具有更强的表达能力 (即更多参数), 但是表现却更差. 我们提出一个假设: 可能是

因为深层的神经网络更加难以进行优化。这个时候我们应该控制变量找出原因。¹ 一种让深层网络至少和浅层网络一样好的方法是，将浅层网络加上一些恒等变换的层，“变成”深层网络。从这种意义上讲，更深的网络不应该比浅的差，只要中间的一些层学习成恒等变换即可。这可能预示着，对于深层网络而言，“恒等”是一个较难学习的特征。这或许是因为深层网络引入太多非线性表达，在高维的参数空间中难以学习到恒等吧。²

既然如此，那么一种自然的思路就是：构造天然的恒等。假设神经网络非线性层的输入输出维度一致，那么可以将要拟合的函数分成两个部分：

$$\mathbf{z}^{(l)} = \mathcal{H}(\mathbf{a}^{(l-1)}) = \mathbf{a}^{(l-1)} + \mathcal{F}(\mathbf{a}^{(l-1)}) \quad (7.4)$$

其中 $\mathcal{F}(\cdot)$ 是残差函数。在网络高层，学习一个恒等映射 $\mathcal{H}(\mathbf{a}^{(l-1)}) \rightarrow \mathbf{a}^{(l-1)}$ 即等价于令残差部分趋近于 0，即 $\mathcal{F}(\mathbf{a}^{(l-1)}) \rightarrow \mathbf{0}$ 。

残差单元可以以跳层连接的形式实现，即将单元的输入直接与单元输出加在一起，然后在激活。因此残差网络可以轻松地用主流的自动微分深度学习框架实现，直接使用 BP 算法更新参数。

实验表明，残差网络很好地解决了深度神经网络的退化问题，并在 ImageNet 和 CIFAR-10 等图像任务上取得了非常好的结果，同等层数的前提下残差网络也收敛得更快。这使得前馈神经网络可以采用更深的设计。除此之外，去除个别神经网络层，残差网络的表现不会受到显著影响，这与传统的前馈神经网络大相径庭。

那么，残差神经网络为什么有效呢？

从梯度反向传播的角度，Skip Link 提供了一条梯度反向传播的旁路，使得梯度可以更加顺畅地传递到浅层。我们考虑式 7.4 描述的层，假设残差块不采取任何激活函数，即

$$\mathbf{a}^{(l)} = \mathbf{z}^{(l)} \quad (7.5)$$

考虑 $l_1 > l_2$ 两个层，递归地进行展开，有

$$\mathbf{a}^{(l_2)} = \mathbf{a}^{(l_1)} + \sum_{i=l_1}^{l_2-1} \mathcal{F}(\mathbf{a}^{(i)}) \quad (7.6)$$

在前向传播时，输入信号可以从任意低层直接传播到高层。由于包含了一个天然的恒等映射，一定程度上可以解决网络退化问题。这样，最终的损失函数 \mathcal{L} 对某低层输出的梯度可以展开为

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(l_1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(l_2)}} + \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(l_2)}} \frac{\partial}{\partial \mathbf{a}^{(l_1)}} \sum_{i=l_1}^{l_2-1} \mathcal{F}(\mathbf{a}^{(i)}) \quad (7.7)$$

上式说明，反向传播时，错误信号可以不经过任何中间权重矩阵变换直接传播到低层，一定程度上可以缓解梯度弥散问题（即便中间层矩阵权重很小，梯度也基本不会消失）。综上，可以认为残差连接使得信息前后向传播更加顺畅。

在文章 [ResidualNetworksAreExponentialEnsembles] 中，作者提出了另一种观点，认为残差神经网络是由一系列路径集合组装成的模型。

Andreas Veit 等人展开了几组实验，在测试时，删去残差网络的部分网络层（即丢弃一部分路径）、或交换某些网络模块的顺序（改变网络的结构，丢弃一部分路径的同时引入新路径）。实验结果表明，网络的表现与正确网络路径数平滑相关（在路径变化时，网络表现没有剧烈变化），这表明残差网络展开后的路径具有一定的独立性和冗余性，使得残差网络表现得像一个集成模型（ensemble）。作者还通过实验表明，残差网络中主要在训练中贡献了梯度的是那些相对较短的路径。

¹ 在接下来将较浅层的输出通过一条旁路直连深层的操作，老师称之为“寻找一种中间状态”，即将浅层网络加上恒等变换层“视为”深层网络，再将两个进行叠加。

² 本小节余下的内容取自 [WhyResNetWorks]。

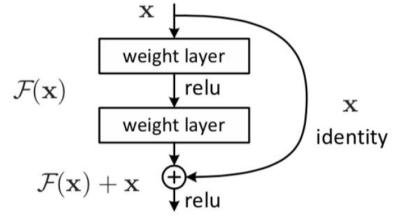


图 7.5：残差神经网络单元

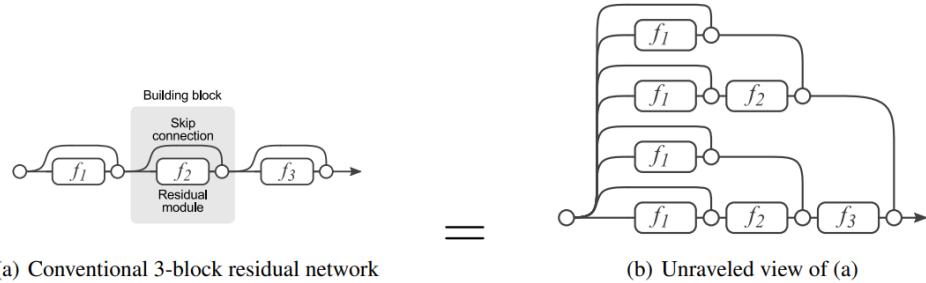


Figure 1: Residual Networks are conventionally shown as (a), which is a natural representation of Equation (1). When we expand this formulation to Equation (6), we obtain an *unraveled* view of a 3-block residual network (b). Circular nodes represent additions. From this view, it is apparent that residual networks have $O(2^n)$ implicit paths connecting input and output and that adding a block doubles the number of paths.

图 7.6: 文章给出的配图

7.3 Overfitting

过拟合产生的原因是 `data` 和 `model` 之间的不平等。例如分类问题，我们给网络输入的数据，对于机器来说就是某个概念的“定义”。但同类事物有共性也有差别，定义应当只取其共性，但神经网络并不一定能做到这一点。所以，减弱 `model` 增强 `data` 的手段，有助于打破这种不平衡。

首先我们来看看在神经网络训练过程中常见的 generalization gap:

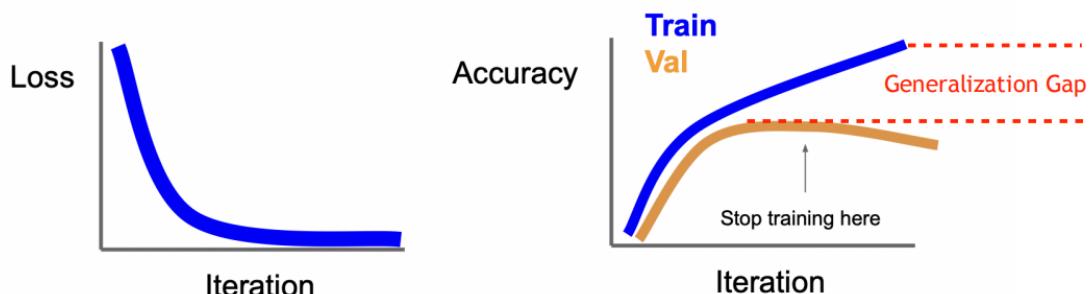


图 7.7: generalization gap 示意图

所谓 generalization gap, 就是指模型在同分布的训练集和测试集上表现出的准确率差异. 这里右图略有瑕疵, 我们说的 gap 应该是指同一轮测试之后两个集合上表现的差异. 另外需要注意的是: 在训练时即使 loss 一直在减小, 也并不说明在测试集上的准确率在提升, 因为可能有过拟合的问题.

对一个过拟合的模型来说, 它包含了多于区分数据所必须的数量的参数. 为减少过拟合, 我们可以从数据和模型两方面着手处理. 数据方面, 一种自然的想法是, 如果训练集数据充足而且包含较多各种无关特征, 那么模型更不容易过拟合. 最简单的方法是增加数据量, 但有时这是代价很大的, 我们可以对现有的数据进行处理, 获取更多数据. 而在模型方面, 我们将采取一些限制模型复杂度的处理. 数据方面包括 Data Augmentation 等, 而后者我们会提及正则化与 Dropout.

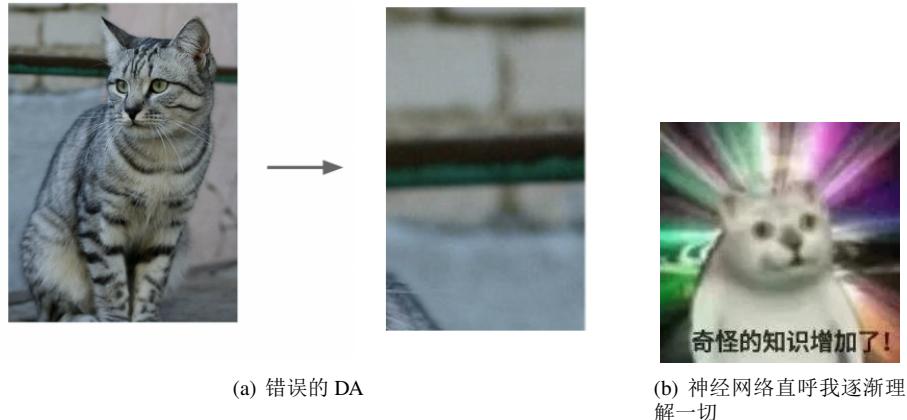
Data Augmentation

所谓数据增强 (data augmentation), 就是一种利用已有数据人工生成新数据的手段. 常用的手段有翻转 (flip), 旋转 (rotation), 缩放 (scale), 裁剪 (crop), 移位 (translation) 和高斯噪声³ (Gaussian Noise) 等.

³当神经网络试图学习可能无用的高頻特征(大量出現的模式)时,通常会发生过度拟合。具有零均值的高斯噪声基本上在所有频率中具有数据

例如,如果我们的网络需要识别猫的图片,那么在训练时,我们可以将一张图片水平翻转,这当然还是一只猫.于是我们轻松将数据量翻了个倍.

但是!要注意的是,你必须保证数据增强之后,你所希望学习的特征并没有发生改变.这隐含着一种对称性,而诺特定理指出每种对称性对应一种守恒量;我们不难看出数据增强的依赖的对称性对应的守恒量就是标签...(胡言乱语)即每一只猫是左右对称的.相对应地,我们很少对猫图片进行上下翻转.所以,进行正确的数据增强是很重要的.比如下面这种反面教材:



总之,DA 不能太强也不能太弱,很多时候其程度还是需要人工评判.

在空间位置方面的数据增强手段有缩放,裁剪,翻转,填充,旋转,移位,仿射变换(affine transformation)等.色彩方面,可以调整亮度(brightness),对比度(contrast),饱和度(saturation),色调(hue)等.此外还可以应用GAN/Reinforcement Learning 完成数据增强.

Regularization

所谓正则化,就是限制模型复杂度的一种手段.具体做法就是在损失函数当中添加一项正则项:

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{DataLoss: \text{模型预测需要拟合训练集}} + \underbrace{\lambda R(W)}_{Regularization: \text{防止模型在训练集上表现太好}} \quad (7.8)$$

举个例子,对于 7 个散点,我们可以拟合成一条直线,也可以用六次函数经过每一个点,但后者常常不是我们想要的,而正则项的添加就是对模型训练程度的一个限制.设想一个模型是六次函数,我们如何限制其表达近似线性的数据?只需将常数项和一次项系数之外的系数限制在 0 附近即可.

这种想法与一种哲学原理不谋而合:即 Occam's Razor. 我们常用的正则函数有 L_2 Regularization: $R(W) = \sum_{k,l} W_{k,l}^2$, 以及 L_1 Regularization: $R(W) = \sum_{k,l} |W_{k,l}|$, 以及其混合等. 使用时需要注意: λ 需要控制正则项的量级,使其与损失函数相当,否则如果正则项过大,会使得网络全力向简单发展;反之则没有效果.

Dropout 简单地说,Dropout 就是以某个概率随机使神经元失效.

通常,Dropout 的概率设置在 0.2 到 0.5 之间,表示有 20% 到 50% 的概率丢弃一个神经元.需要注意的是,在训练完成后并应用到推理/测试阶段时,Dropout 是关闭的,此时所有神经元都会被使用.此外,为了保证神经网络的输出在训练和测试阶段的一致性,在测试阶段需要将所有神经元的输出乘以一个比例(这个比例等于 Dropout 的保留概率),以补偿训练阶段丢弃的部分(大小).

在实际使用 Dropout 时,通常是在全连接层和 RNN 层中使用,尽量避免在卷积层中使用,因为卷积层具有局部特征的结构,随机丢弃会破坏这种局部性.

点,加入图片中可以有效地扭曲高频特征.这也意味着较低频率的组件(通常是预期数据)也会失真,但这种困难可以被克服.

BatchNorm 作为正则化

BatchNorm 使得进入激活函数的输入必须服从特定的高斯分布, 这限制了模型的表达能力, 因此也是一种正则化的手段, 有助于减少过拟合. 使用 BatchNorm 后, 就不需要再添加 Dropout 了.

总之, 纵观各种正则化的方法, 处处透露出 Less is More 的思想, 即越简单的表达方式越好.⁴

7.4 Summary of Mitigating Overfitting

原则: 平衡数据的多样性和模型的容量.

方法:

1. Data Augmentation. (从数据的角度)
2. BatchNorm. (模型角度)
3. Regularization. (模型角度)
4. Dropout. (模型角度)

其中 DA 和 BN 用了总是比不用好. 后两者视情况而定.Dropout 一般只用于较大的 FC 层.

⁴Less is More, but More is different. (笑)

第 8 章 Classification

图片分类是 CV 领域的核心问题. 简单来说, 就是给定一张图片, 判断其属于何种分类, 比如是不是猫或狗等等, 这对图片的语义理解非常重要.

但是传统的方法对此类问题难以下手, 因为图片通常是由数字的矩阵来描述, 而从数字到语义有很大的鸿沟, 很难设计某个规则来判定是否属于某类. 比如: 对象不同的姿势, 不同的摄像机视角, 不同的背景信息, 不同的光照条件, 以及对象被隐藏和类内差异等问题.

对于一个好的图片分类器, 应该对上述无关因素不敏感, 而这也是 data augmentation 的意义. 比如 rotation 代表姿势和视角的改变, 颜色改变代表光照的变化等.

对于图片分类, 我们有下列方法: 无参方法有最近邻法, 参数方法则可以采用 CNN.

8.1 Nearest Neighbour Classifier

所谓最近邻, 就是将图片视为高维空间的点, 将每个训练数据作为已知点, 定义一种图片间距离的度量, 选取最近的一个(或几个)训练数据的类别作为待判断图片的类别. 这是一种非常低效的方法, 其完美避开了我们上面说到的应具有的标准, 对光照/背景/视角/姿势极为敏感, 正确率极低, 而且需要存储所有训练集. 因此, 实际中从不使用此种方法. 但是最近邻方法在度量学习当中仍有广泛应用.

8.2 Using CNN for image Classification

选用 CNN 之后, 我们需要面对的问题有两个: 选取何种网络结构, 以及如何设计损失函数. 如今分类问题的网络范式是 Softmax classifier + cross-entropy loss.¹

SoftMax

SoftMax 就是一个 $\mathbb{R}^k \rightarrow (0, 1)^k$ 的映射.

$$\sigma(z)_i = \frac{\exp \beta z_i}{\sum \exp(\beta z_j)} \quad (8.1)$$

一般取 $\beta = 1$. 当 $\beta \rightarrow \infty$ 时, SoftMax 变成 Argmax.

所以 SoftMax 是 Soft 的 Argmax.

关于 loss 的设计, 如果正确标签是 one-hot 的, 那么我们可以使用负对数概率 (NLL) 作为损失函数. 但是如果 ground truth 也是一个概率分布(有时这是人为的), 那么我们就需要对两个概率分布的距离度量给出定义. 在信息论领域常用的度量是 KL divergence $D(P \parallel Q)$, 其定义如下:

$$D(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)}. \quad (8.2)$$

这个度量并不满足距离的定义, 因为其满足正定性, 而不满足对称性和三角不等式.

我们不难看出

$$D(P \parallel Q) = \underbrace{- \sum_{x \in \mathcal{X}} P(x) \log Q(x)}_{H(P, Q)} - \underbrace{\left(- \sum_{x \in \mathcal{X}} P(x) \log P(x) \right)}_{H(P)}. \quad (8.3)$$

¹对二分类问题, 也可采用 SVM loss. 但是扩展的多分类 SVM loss 在如今已经极少使用了.

即 KL divergence 是相对熵和分布 P 的熵之差. 如果 P 是 groud truth 的分布, 那么第二项成为常数, 就得到了我们的交叉熵损失函数:

$$\mathcal{L}_{CE} = H(P, Q) = - \sum_{x \in \mathcal{X}} P(x) \log Q(x). \quad (8.4)$$

交叉熵函数在随机初始化时, 取值约为 $\log(\text{sum of classes})$. 它没有上界, 有下界 0.

所以 CrossEntropyLoss 应该在 \log 类别数开始下降

8.3 Cross Entropy Loss V.S. Accuracy

1.CEL 有可能已经降到了 $\log 2$, acc 仍是 0. 例子: $\text{Pr} = [0.499, 0.501]$, 仍然输出错误答案, 但是 $loss = \log 2$ 很小

2. $acc = 100\%$ 的时候, CEL 仍然可能是初始化的 $\log(N)$, 同理举一个例子: $\text{Pr} = [0.498, 0.001, 0.001]$

综上所述, 两者没有确定关系, 训练一定要同时画两个曲线

第 9 章 CNNs for Image Classification

当我们分析一个 CNN 的结构时, 需要考虑以下的方面:

1. 表示能力
2. 是否适合任务
3. 是否容易优化
4. 代价

9.1 Reception Field

如图, 使用三层 3×3 卷积层, 感受野与一层 7×7 卷积层相同.

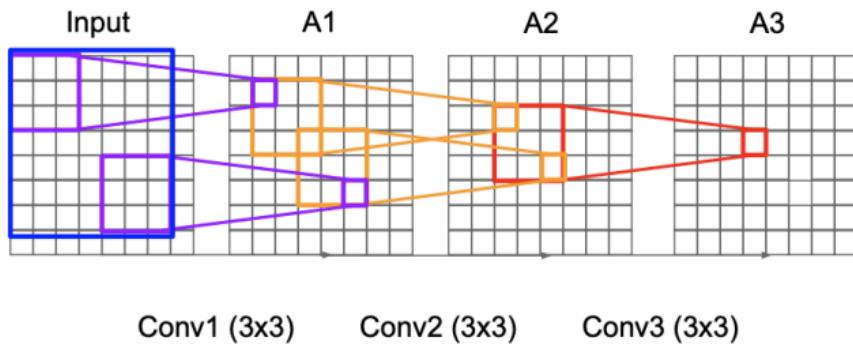


图 9.1: 三层 3×3 卷积核的感受野

感受野是一个很重要的概念, 它表征一个数据可以接收到原图多大范围的信息. 我们这里姑且认为感受野相同则表达能力相同. 我们希望可以在神经网络的中部将整个图片纳入感受野, 这样在后面可以进行全图的 pixel 信息的交流, 有利于结合多个特征. 例如: 结合狗的耳朵和毛色进行判断.

既然三层 3×3 卷积层, 感受野与一层 7×7 卷积层相同, 那么为什么要选用小而深的网络呢? 其一, 层数增加, 网络的非线性性增加, 分割能力更强. 此外, 参数量也更小 ($3 \times 3^2 C^2 < 7^2 C^2$).

第 10 章 Segmentation

当图片出现不同的对象 (例如同时出现猫狗), 那么能否将属于同类的 pixel 分到一起? Not a global label.

Image segmentation: 将图片分割为连贯的各个对象, 不关心其语义. 经典方法: grouping and clustering. 后者就是指将相似的数据点分为一组, 然后给予它们一个标识符.

clustering-based segmentation: 对于实际图片, 颜色当然不可能如此理想. 因此我们要选取三个“中心”来代表, 然后将每个 pixel 以离其最近的中心作为标记.

目标函数:

$$c^*, \delta^* = \arg \min_{c, \delta} \frac{1}{N} \sum_j^N \sum_i^k \delta_{ij} (c_i - x_j)^2. \quad (10.1)$$

很难求解.

很多时候, 这个问题是一个“先有鸡还是先有蛋”的问题. 即何者优先?

破解之法: 如果我们提前知道有 k 个 center, 则可以调用 kmeans. 实际就是 random initialization.

coordinate decent 是一个启发式策略 (先找一个方向的最小值, 然后下降). 但无论如何, kmeans 对于初始化非常敏感. 它必须提前知道几个 group, 并且是无监督的. kmeans 实际上进行了对 intensity 进行了 quantum. 除此之外还有 mean shift.¹

总结: grouping-based: 只关心局部, 不关心语义, 不可能达到 high-level. 是 bottom-up 的方法. 而神经网络则是 top-down 的.

由于纯粹的 Instance segmentation 已经极为少见, 因此今天提及这个词, 多是带语义的.

我们今天的问题: Semantic segmentation. 实际上就是对每个 pixel 进行 classification.

segmentation 时候的一大问题: 对每个 patch, 分类极度依赖上下文.(纯黑的牛) 因此, 将每个 pixel 割裂开是不可能实现分类的.

原先的神经网络: 图像->padding/strided cnn-> 拉成 vector->MLP->softmax-> 类别信息. 这是一个不断提取特征的过程. 但在 segmentation 当中, 我们的输出应该与输入一样大. 怎么把 feature map 先变小再变大? 我们的想法是: 先缩小提取特征 (两头牛), 然后回到原图上以此为原则, 寻找.

NN 核心概念: Auto-Encoder. 自监督. 定义 reconstruction loss

$$\|\mathbf{x} - \hat{\mathbf{x}}\|_2 \quad (10.2)$$

Encoder MLP $f : \mathbb{R}^m \rightarrow \mathbb{R}^h, h \ll m$. decoder: $g : \mathbb{R}^h \rightarrow \mathbb{R}^m$. 对于 $H \times W \times 3$ 的图像, 实际维度到不了这么多. 因此可以进行 encode.

考虑 classification: $\mathbb{R}^{H \times W \times 3} \rightarrow \{0, 1\}^k$. 这样我们就用一个词表示了原图. 同样, 对于 Auto-Encoder, 也是如此, 即我不相信这么多像素都是独立分布, 应该是 $H \times W \times 3$ 空间的低维曲面. 同样对于 segmentation, 我们希望从原图到特征 map, 然后回到原图, 获得分类, 而不需要原图的 RGB 信息.

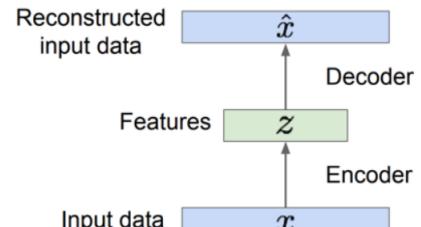


图 10.1: name of the figure

In-Network Unpadding: Unpooling.

第一种是 copy-paste, 需要用 conv 对 raw unpadding image 进行处理.

In-Network Unpadding: Max Unpooling: 记住哪个位置是池化中最大的. 也就是说, 并不是所有信息都在 bottleneck(中间小的部分)处. 这也需要进行大量 conv 加工.

能不能在一步之内完成? Learnable unpadding: Transpose convolution. 学习出来一个 conv, 从 $2 \times 2 \rightarrow 4 \times 4$. Fully Convolutional Network. 全都是 Conv 进行降采样和升采样.

每一次把 feature map 变小, 后面层的 conv 的 reception field 都会变大, 比不做 dimentional reduce 的图片感受野更大. 有助于提取 global context. bottleneck 的双重意义: 增大感受野, 减小 size.

关键: 增大感受野, 收缩 image 从而集中特征, 去除无用.

¹ 其实没人会拿 intensity 做 kmeans. 因为 intensity 显然不能体现 pixel 的相关性.

那么 FCN 有什么问题呢? 我们先来看看它的 bottle neck 存储了什么? 降低 resolution, 多个 channel. 现在在 neck 里面的内容本来就不多, 你还要要求通过相同的升采样之后, 还能还原出 boundary, 这件事与 classification 相比, 不是一句话能说完的, neck 里恐怕很难同时存储下特征和原图的 boundary 信息.

那么我们让各个层次的 resolution 经过几层 conv² 直接连接到对称的升维操作, 照着把 bound 画出来就可以, 中间 neck 就只需要记 context 了. 这是一个非常经典的结构, 被称为 UNet, 也是如今 Segmentation 乃至 depth detection, dense prediction 都需要参考的结构, 甚至也是 Diffusion model³ 的一个重要的 Backbone.

Evaluation Metrics: Pixel Accuracy.

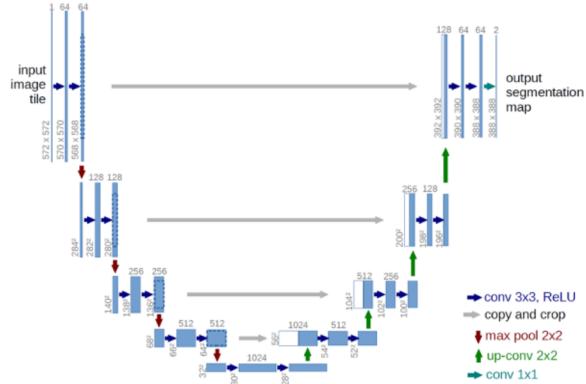


图 10.2: UNet 的结构示意图

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (10.3)$$

当然这很有问题: 比如一个人头只占据整个图片, 那么将整个图片预测为背景, 那么错误率也很低, 但是这很显然不对.

另一个 loss: IoU->mean of every class IoU -> soft IoU.

²前几层可以理解为在寻找 corner 和 edge.

³但是非常有趣的是 U-net 最初是在生物学语义分割中提出的. 2024 笔者注

第 11 章 3D Vision

Computer vision deals with: acquiring, processing, analyzing, and understanding, (might also include) generating or imagining visual data.

联系深度图, RGB 图和点云的概念—— camera. RGB 可以视作光打到深度图上, 对 RGB 进行采样得到.

如何设计 camera? 多个光源照在一点, 会产生 blurring. 因此产生了 Pinhole camera. 我们可以简单地描述其几何关系:

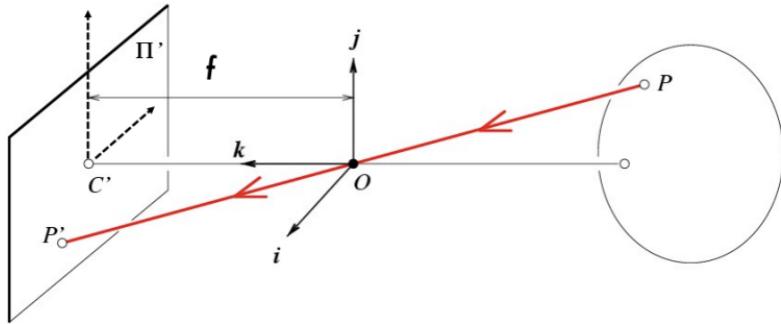


图 11.1: pinhole camera 的几何关系

满足如下变换

$$\begin{cases} x' = f \frac{x}{z} \\ y' = f \frac{y}{z} \end{cases} \quad (11.1)$$

aperture size 实际上控制模糊程度和亮度. 所以我们可以添加透镜 (lens), 透镜可能产生畸变, 因此真实相机往往复杂得多.

11.1 Intrinsics

上述变换的本质是一个映射 $E : \mathbb{R}^3 \rightarrow \mathbb{R}^2$. 投影操作, projection mapping. 我们还需要完成两件事: 从实际的度量(米)转换到 pixel 为单位, 以及在图片中, 左下角为零点, 而非中心, 因此需要添加 offset.

$$P = (x, y, z) \rightarrow P' = \left(\alpha \frac{x}{z} + c_x, \beta \frac{y}{z} + c_y \right) \quad (11.2)$$

能否用矩阵表示? 很遗憾, 这不是线性变换. 因此引入 homogeneous coordinate system, 即增加一个维度作为除法.

$$P'_h = \begin{bmatrix} \alpha x + c_x z \\ \beta y + c_y z \\ z \end{bmatrix} = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \beta & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (11.3)$$

如果考虑相机坐标系的 screw, 如下图, 则需要进行简单的变换:

$$\begin{cases} x = \alpha(\hat{x} - \cot \theta \hat{y}) + c_x \\ y = \beta \frac{\hat{y}}{\sin \theta} + c_y \end{cases} \quad (11.4)$$

最后得到

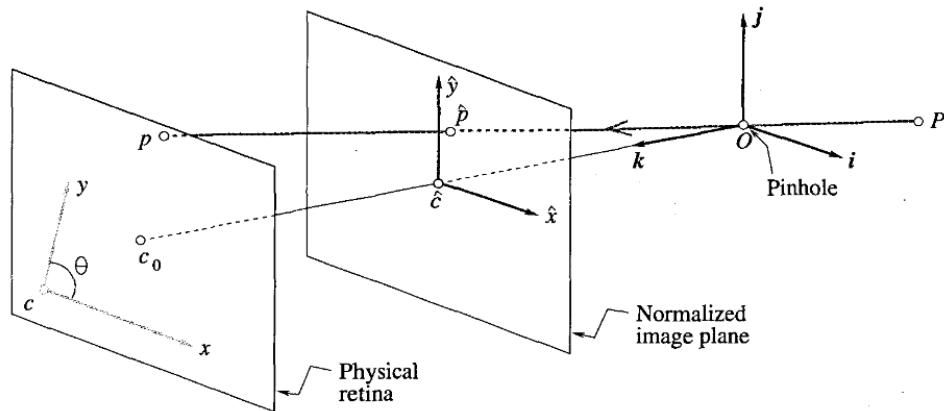


图 11.2: 成像过程坐标示意图

$$P' = \begin{bmatrix} \alpha & -\alpha \cot \theta & c_x & 0 \\ 0 & \frac{\beta}{\sin \theta} & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (11.5)$$

进行分解后表示为

$$P' = \mathbf{M}P = \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} P \quad (11.6)$$

其中, 矩阵 \mathbf{K} 定义为

$$\mathbf{K} = \begin{bmatrix} \alpha & -\alpha \cot \theta & c_x \\ 0 & \frac{\beta}{\sin \theta} & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (11.7)$$

它是相机的内部参数, 拥有 $\alpha, \beta, \theta, c_x, c_y$ 五个自由度.

11.2 Extinsics

上一节当中, 我们从 camera coordinate system->retina plane metric->image coordinate system. 但是我们希望完成从 world coordinate 到 image coordinate system 的转变. 不难看出, 这是两个空间直角坐标系的平移和旋转变换. 首先我们来看平移:

对于点 P , 如果要将其平移向量 \mathbf{T} , 则可以用如下的矩阵乘法表示:

$$P' \rightarrow \begin{bmatrix} \mathbf{I} & \mathbf{T} \\ \mathbf{0}^\top & 1 \end{bmatrix}_{4 \times 4} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (11.8)$$

对于旋转, 我们先考虑平面直角坐标系的情形. 假如最开始的坐标系为 S , 而 S' 是将 S 逆时针旋转 θ , 那么可以得出如果要将 S 中的向量逆时针旋转 θ 就是将坐标乘以 R_θ , 其中

$$R_\theta = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad (11.9)$$

原理即是将所有基底旋转 θ , 坐标表示不变, 则自然就随基底旋转. 同样对于两个坐标系, $Oijk$ 和 $Oi'j'k'$, 将前者坐标下的点 (x, y, z) 的基底向后者旋转, 则需要左乘

$$\mathbf{R} = \begin{bmatrix} i' \cdot i & j' \cdot i & k' \cdot i \\ i' \cdot j & j' \cdot j & k' \cdot j \\ i' \cdot k & j' \cdot k & k' \cdot k \end{bmatrix} \quad (11.10)$$

但是,如果我们想在后者的坐标系中表示同一个向量(注意这两者的区别),那就需要左乘此矩阵的逆.由于此矩阵正交,所以也就是左乘其转置.

可以结合二维情形验证上述表达式.除此之外,我们还可以将旋转分解为三个方向的旋转:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}, R_y(\beta) = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix}, R_z(\gamma) = \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (11.11)$$

$$P' \rightarrow \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix}_{4 \times 4} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (11.12)$$

将平移和旋转结合,我们就有了统一的形式:

$$P' = K \begin{bmatrix} I & 0 \end{bmatrix} P = K \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}_{4 \times 4} P_w = K \begin{bmatrix} R & T \end{bmatrix} P_w \quad (11.13)$$

其过程示意图如下:

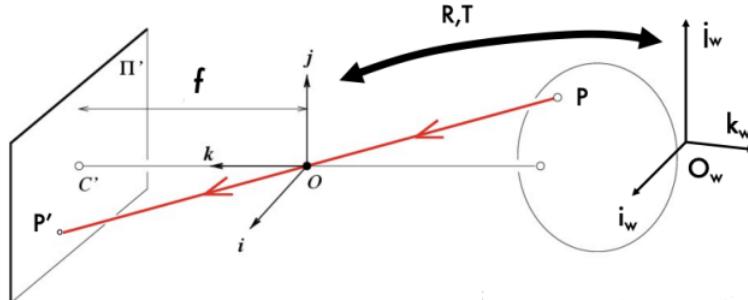


图 11.3: 坐标变换的全过程

若令 $M = \mathbf{K}[\mathbf{R} \ \mathbf{T}]$, 设其三个行向量为 $\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3$, 则有

$$P' = \left(\frac{\mathbf{m}_1 P_w}{\mathbf{m}_3 P_w}, \frac{\mathbf{m}_2 P_w}{\mathbf{m}_3 P_w} \right) \quad (11.14)$$

投影变换的性质: 点映射成点, 线映射成线, 近大远小.

11.3 weak perspective

在弱透视模型中, 点首先用正交投影投影到参考平面, 然后用射影变换投影到图像平面. 如下图: 这里的正交投影可以表示为

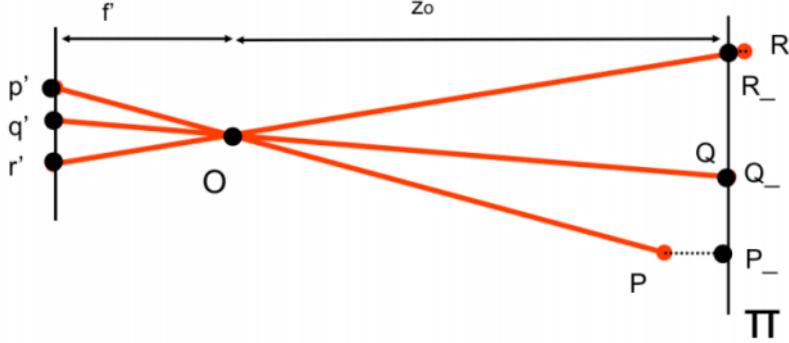


图 11.4: weak perspective

$$\mathbf{O} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \stackrel{\text{def}}{=} \begin{bmatrix} \mathbf{O}' & z_0 \\ \mathbf{0}^\top & 1 \end{bmatrix} \quad (11.15)$$

即将所有 z 分量变为 z_0 . 随后根据我们的变换公式

$$\mathbf{P}' = \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \mathbf{O} \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^\top & 1 \end{bmatrix} \mathbf{P} = \mathbf{K}_{3 \times 3} \begin{bmatrix} \mathbf{O}' \mathbf{R} & \mathbf{O}' \mathbf{T} + z_0 \end{bmatrix}_{3 \times 4} \mathbf{P}_{4 \times 1} \quad (11.16)$$

进一步, 由于 \mathbf{O}' 的第三行全零, 因此中间矩阵的第三行是 $[0, 0, 0, z_0]$. 记 $\mathbf{R}_2, \mathbf{t}_2$ 分别是 \mathbf{R}, \mathbf{T} 的前两行, 上式可以改写为

$$\mathbf{P}' = \mathbf{K} \begin{bmatrix} \mathbf{R}_2 & \mathbf{t}_2 \\ \mathbf{0}^\top & z_0 \end{bmatrix} \mathbf{P} \quad (11.17)$$

我们记

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_2 & \mathbf{p}_0 \\ \mathbf{0}^\top & 1 \end{bmatrix}, \quad \mathbf{K}_2 = \begin{bmatrix} \alpha & -\alpha \cot \theta \\ 0 & \frac{\beta}{\sin \theta} \end{bmatrix}, \quad \mathbf{p}_0 = \begin{bmatrix} c_x \\ c_y \end{bmatrix} \quad (11.18)$$

随后得到

$$\mathbf{P}' = \begin{bmatrix} \mathbf{K}_2 \mathbf{R}_2 & \mathbf{K}_2 \mathbf{t}_2 + z_0 \mathbf{p}_0 \\ \mathbf{0}^\top & z_0 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \quad (11.19)$$

不难看出, \mathbf{P}' 的第三个坐标 (即齐次项) 为常数 z_0 , 可以直接写成更简单的二维坐标形式:

$$\mathbf{p} = \mathbf{M} \mathbf{P} \begin{bmatrix} \mathbf{A} & \mathbf{b} \end{bmatrix} \mathbf{P} \quad (11.20)$$

其中

$$\mathbf{A} = \frac{1}{z_0} \mathbf{K}_2 \mathbf{R}_2, \quad \mathbf{b} = \frac{1}{z_0} \mathbf{K}_2 \mathbf{t}_2 + \mathbf{p}_0 \quad (11.21)$$

更简单:orthographic (Affine) projection. 正交投影. 没有近大远小. 这种投影在你不希望有近大远小的时候可以用到.

第 12 章 Camera Calibration

我们前面已经知道, 从 $P_w \rightarrow P'$ 的坐标变换是

$$P' = \mathbf{M}P_w = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix} P_w \quad (12.1)$$

其中

$$\mathbf{M} = \begin{pmatrix} \alpha \mathbf{r}_1^T - \alpha \cot \theta \mathbf{r}_2^T + u_0 \mathbf{r}_3^T & \alpha t_x - \alpha \cot \theta t_y + u_0 t_z \\ \frac{\beta}{\sin \theta} \mathbf{r}_2^T + v_0 \mathbf{r}_3^T & \frac{\beta}{\sin \theta} t_y + v_0 t_z \\ \mathbf{r}_3^T & t_z \end{pmatrix} \quad (12.2)$$

那么什么是 calibration problem 呢? 如果我们已知世界坐标 P_1, \dots, P_n (形式为 $[O_w, i_w, j_w, k_w]$) 和对应的图像坐标 p_1, \dots, p_n . 我们的目标是通过这些已知数据, 获得 intrinsic and extrinsic parameters. 这个问题的意义, 比如我们希望从图像运动获取实际运动.

问题有 11 个自由度: $5(\text{in})+3(\text{ex-r})+3(\text{ex-t}) = 11$. 需要 11 个方程, 6 个点.

对每个 $p_i(u_i, v_i)$, 我们有

$$\begin{aligned} u_i &= \frac{\mathbf{m}_1 P_i}{\mathbf{m}_3 P_i} \rightarrow u_i (\mathbf{m}_3 P_i) = \mathbf{m}_1 P_i \rightarrow u_i (\mathbf{m}_3 P_i) - \mathbf{m}_1 P_i = 0 \\ v_i &= \frac{\mathbf{m}_2 P_i}{\mathbf{m}_3 P_i} \rightarrow v_i (\mathbf{m}_3 P_i) = \mathbf{m}_2 P_i \rightarrow v_i (\mathbf{m}_3 P_i) - \mathbf{m}_2 P_i = 0 \end{aligned} \quad (12.3)$$

这样我们可以列出方程组:

$$\left\{ \begin{array}{l} u_1 (\mathbf{m}_3 P_1) - \mathbf{m}_1 P_1 = 0 \\ v_1 (\mathbf{m}_3 P_1) - \mathbf{m}_2 P_1 = 0 \\ \vdots \\ u_i (\mathbf{m}_3 P_i) - \mathbf{m}_1 P_i = 0 \\ v_i (\mathbf{m}_3 P_i) - \mathbf{m}_2 P_i = 0 \\ \vdots \\ u_n (\mathbf{m}_3 P_n) - \mathbf{m}_1 P_n = 0 \\ v_n (\mathbf{m}_3 P_n) - \mathbf{m}_2 P_n = 0 \end{array} \right. \quad (12.4)$$

将 m 展开, 获得方程组:

$$\mathbf{P}m = \mathbf{0} \quad (12.5)$$

其中

$$\mathbf{P} \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{P}_1^T & \mathbf{0}^T & -u_1 \mathbf{P}_1^T \\ \mathbf{0}^T & \mathbf{P}_1^T & -v_1 \mathbf{P}_1^T \\ \vdots & \vdots & \vdots \\ \mathbf{P}_n^T & \mathbf{0}^T & -u_n \mathbf{P}_n^T \\ \mathbf{0}^T & \mathbf{P}_n^T & -v_n \mathbf{P}_n^T \end{pmatrix} \quad (12.6)$$

以及

$$m = \begin{pmatrix} \mathbf{m}_1^T \\ \mathbf{m}_2^T \\ \mathbf{m}_3^T \end{pmatrix} \quad (12.7)$$

但是这个问题过定, 且有平凡解. 我们先对 m 添加一个 constrain: $\|m\| = 1$. 随后用 SVD 求解如下优化问题:

$$\begin{aligned} &\text{minimize } \|\mathbf{P}m\|^2 \\ &\text{s.t. } \|m\| = 1 \end{aligned} \quad (12.8)$$

进行 SVD 获得最小的特征向量, 即为解.

此时还没有结束, 我们需要定出收缩因子. 假定

$$\mathcal{M} = \begin{pmatrix} \alpha \mathbf{r}_1^T - \alpha \cot \theta \mathbf{r}_2^T + u_0 \mathbf{r}_3^T & \alpha t_x - \alpha \cot \theta t_y + u_0 t_z \\ \frac{\beta}{\sin \theta} \mathbf{r}_2^T + v_0 \mathbf{r}_3^T & \frac{\beta}{\sin \theta} t_y + v_0 t_z \\ \mathbf{r}_3^T & t_z \end{pmatrix} \rho \quad (12.9)$$

此时我们发现 \mathcal{M} 第三行可以直接定出 ρ . 于是这样我们就可以获得相应参数.

实际上, 对这个问题的求解就是进行 RQ 分解¹. 我们知道 RQ 分解就是将一个矩阵 \mathbf{A} 分解为:

$$\mathbf{A} = \mathbf{R}\mathbf{Q} \quad (12.10)$$

其中 \mathbf{Q}, \mathbf{R} 分别是正交矩阵和上三角矩阵. 当然, 得到的正交矩阵可能是瑕旋转矩阵, 即行列式为 -1 的正交矩阵. 这种情况我们要检查分解得到的上三角矩阵是否满足相机内参的要求, 比如 $\sin \theta > 0$, 此时将此列和正交矩阵对应行乘以 -1 即可.

当然并不是所有的 6 个点都可以. 不能在同一平面 (15:50). 对于一般的有畸变的相机, 有更复杂的非线性处理方式, 可能没有解析解.

12.1 Some Problems with Camera

问题: 只知道内参或深度信息可以唯一确定一个物体吗?

只有内参或者深度信息, 都不能确定一个物体, 必须两个都知道才能得到物体真实数据

只有内参: 外参和物体大小同时变化

只有深度信息: 物体在平行于相机的那个平面上的大小可以变化, 因为这个平面上的物体是同一个深度

问题: 假设我们在知道一个照片以外, 还知道每个像素的深度, 那么可以找出真实世界中两点的距离吗?

不可以. 因为不知道相机内参, u 和 v 代表着像素差, 无法确定相机参考系下的 $\Delta x, \Delta y$ 也就是真实距离差.

相机相关计算

1. Depth back projection:

$(K, u, v, z) \rightarrow (x, y)$

使用 K 的定义

2. Camera calibration:

$(x, y, z, u, v) \rightarrow K$

如果不知道 K . 那么就是相机标定

问题: 为什么相机标定的时候所有参考点不能在同一个平面上?

如果所有参考点都在同一个平面上, 那么相机的观测将缺乏深度信息, 因为所有的标定点都位于一个二维平面内. 这会导致所谓的“退化配置”(degenerate configuration), 在这种配置下, 我们不能唯一地确定相机的内外参数, 尤其是关于深度和空间位置的信息.

例如: 我们无法区分相机距离标定平面远但焦距短, 与相机距离标定平面近但焦距长的情况. 这两种情况在所有参考点都在同一平面上时可能会产生相似的投影图像.

问题: 根据 depth back projection 计算出来的相机坐标系下的 Δx 和 Δy 是不是 world coordinate 下的距离?

是的.

因为旋转和平移是保角保距离变换.

¹关于 QR 分解, 请参见附录 D

第 13 章 Single View Geometry

这一章节在 2024 年教学中已经被删去, 为了让有兴趣的读者了解, 故保留

上面我们解释了从 3D 转换到 2D 的过程, 那么反过来, 能否从单视角还原成 3D 呢? 很显然, 这并不是一个有唯一解的问题. 无论是从维度的角度, 还是实际操作角度都是如此. 成像的时候, 一个 pixel 代表着一条射线, 而其深度无法确定.

尽管如此, 我们还是能得到一些有意思的结论. 比如我们知道点会映射成点, 而线映射成线. 以及右图中, 似乎铁轨和草皮交于图像上方的一点.



图 13.1: 一张风景图

13.1 Transformation in \mathbb{R}^2

我们先来看看二维平面中的线. 对于 $ax + by + c = 0$, 用齐次坐标表达为 $l = [a, b, c]^\top$, 则点在线上等价于 $x^\top l = 0$. 这里 $x = [x, y, 1]^\top$ 为齐次坐标. 两条线 l, l' 的交点坐标是 $x = l \times l'$.

我们还能发现, 这样的表示还能适用于平行线的交点. 不难看出两条平行线的叉乘

$$\begin{vmatrix} i & j & k \\ a & b & c \\ a' & b' & c' \end{vmatrix} \quad (13.1)$$

的齐次坐标 $ab' - a'b = 0 \cdot x_\infty = [b, -a, 0]^\top$. 不难验算得到所有与这条直线平行的点都交于 x_∞ .

我们还可以定义 $1_\infty = [0, 0, 1]^\top$. 这条线是所有无穷点的共线, 即无穷远处的直线. 这是在欧式空间无法表达的一条直线. 其齐次坐标当然也可以是任何常数.

我们尝试将射影变换¹作用于某个无穷远点, 得到如下结果:

$$\mathbf{p}' = \mathbf{H}\mathbf{p}_\infty = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{v} & b \end{bmatrix} \begin{bmatrix} a \\ b \\ 0 \end{bmatrix} = \begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix} \quad (13.2)$$

也就是说, 无穷远点可能会被映射到某个有限远的点. 但是对于仿射变换, 就不存在这种问题了:

$$\mathbf{p}' = \mathbf{H}\mathbf{p}_\infty = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ 0 \end{bmatrix} = \begin{bmatrix} p'_x \\ p'_y \\ 0 \end{bmatrix} \quad (13.3)$$

对于变换后的直线, 根据射影变换的性质, 如果将 ℓ 变换为 ℓ' , 那么如果 $\mathbf{x} \in \ell$, 则 $\mathbf{x}' = \mathbf{H}\mathbf{x} \in \ell'$. 设对直线的变换是 \mathbf{T} , 那么有

$$\mathbf{x}'^\top \mathbf{T} \ell = \mathbf{x}^\top \mathbf{H}^\top \mathbf{H}^{-\top} \ell = 0 \quad (13.4)$$

得出 $\mathbf{T} = \mathbf{H}^{-\top}$. 同样我们可以发现射影变换可能会将无穷远处的直线映射到有限远, 而仿射变换不会.

13.2 Vanishing Points

有了上面的铺垫, 我们来看看 vanishing point 的概念. 我们可自然地将上述概念推广到三维的情形, 那么对三维空间中一系列平行线的无穷远点 \mathbf{x}_∞ , 经过相机变换 \mathbf{M} , 就会成为二维中的有限远的一个点 \mathbf{p}_∞ , 也就是 vanishing point, 如下图:

¹附录B当中简单介绍了几种变换的形式和性质.

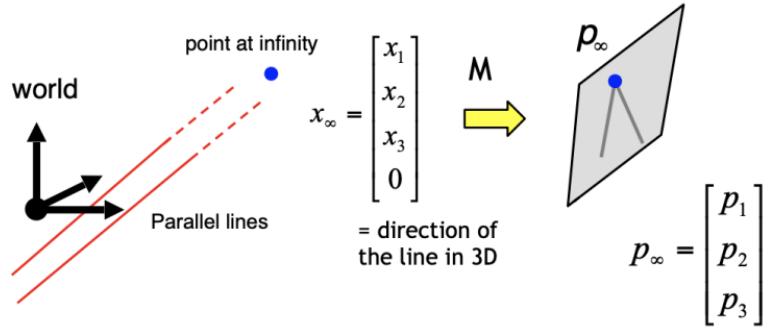


图 13.2: Vanishing points 在 world coordinate 中的示意图

上图的坐标是 world coordinate 当中. 如果是在 camera coordinate 当中, 设 $\mathbf{d} = (a, b, c)$ 是直线的方向向量, \mathbf{v} 代表消失点, 我们还可以导出如下关系²:

$$\mathbf{v} = \mathbf{K}\mathbf{d} \quad (13.5)$$

这一点不难证明: 由于是在相机坐标之下, 无需考虑 extrinsic 参数, 直接有

$$\mathbf{v} = \mathbf{M}\mathbf{x}_\infty = \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ 0 \end{bmatrix} = \mathbf{K}\mathbf{d} \quad (13.6)$$

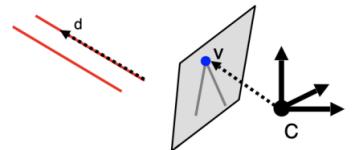


图 13.3: 相机坐标下的方向

这样, 如果已知 vanishing point 的坐标和相机参数, 我们可以知道相机坐标下的这一组平行线的方向:

$$\mathbf{d} = \frac{\mathbf{K}^{-1}\mathbf{v}}{\|\mathbf{K}^{-1}\mathbf{v}\|} \quad (13.7)$$

如果我们继续推广在二维空间中的结论, 那么一系列平行平面将会相交于无穷远处的一条线 l_∞ . 将这条线也作同样的变换, 就可以得到 Vanishing Line, 即 Horizon, 译为天际线, 也就是 3 维空间 l_∞ 在二维的投影:

$$l_{\text{hor}} = \mathbf{H}^{-T} l_\infty \quad (13.8)$$

通过这种方式, 我们可以确定两条线是否平行. 首先找出天际线, 如果照片上的两条线交于天际线的同一点则平行, 反之不平行.

我们还可以通过两个 vanishing point 确定其对应的平行线之间的夹角.

如右图, 根据 $\mathbf{d} = \mathbf{K}^{-1}\mathbf{v}$, 我们可以求出其夹角:

$$\cos \theta = \frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{\|\mathbf{d}_1\| \|\mathbf{d}_2\|}$$

令 $\Omega = (KK^T)^{-1}$, 得到

$$\cos \theta = \frac{\mathbf{v}_1^\top \Omega \mathbf{v}_2}{\sqrt{\mathbf{v}_1^\top \Omega \mathbf{v}_1} \sqrt{\mathbf{v}_2^\top \Omega \mathbf{v}_2}} \quad (13.10)$$

当 $\theta = \frac{\pi}{2}$ 的时候有 $\mathbf{v}_1^\top \Omega \mathbf{v}_2 = 0$. Ω 有五个独立变量. 如果我们已知空间中的三组互相垂直的平行线和它们在图片上的 vanishing point, 则我们可以获得三个方程

$$\begin{cases} \mathbf{v}_1^\top \Omega \mathbf{v}_2 = 0 \\ \mathbf{v}_1^\top \Omega \mathbf{v}_3 = 0 \\ \mathbf{v}_2^\top \Omega \mathbf{v}_3 = 0 \end{cases} \quad (13.11)$$

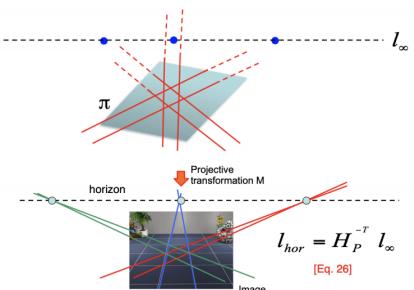
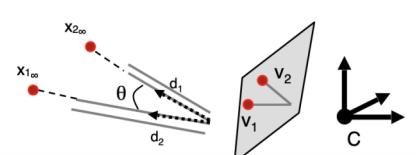
图 13.4: 天际线
(13.9)

图 13.5: Angle from two vanishing points.

²注意在右图里, \mathbf{v} 并不是从相机坐标中心出发的三维矢量, 而是图片坐标下的齐次坐标.

如果我们假定相机具有零偏置和方形像素, 那么不难得到

$$\mathbf{K} = \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \alpha & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{KK}^\top = \begin{bmatrix} \alpha^2 + c_x^2 & c_x c_y & c_x \\ c_x c_y & \alpha^2 + c_y^2 & c_y \\ c_x & c_y & 1 \end{bmatrix} \quad (13.12)$$

$$\boldsymbol{\Omega} = (\mathbf{KK}^\top)^{-1} = \frac{1}{\alpha^4} \begin{bmatrix} \alpha^2 & 0 & -\alpha^2 c_x \\ 0 & \alpha^2 & -\alpha^2 c_y \\ -\alpha^2 c_x & -\alpha^2 c_y & (\alpha^2 + c_x^2)(\alpha^2 + c_y^2) - c_x^2 c_y^2 \end{bmatrix} \quad (13.13)$$

$\boldsymbol{\Omega}$ 的形式变为:

$$\boldsymbol{\Omega} = \begin{bmatrix} \omega_1 & 0 & \omega_4 \\ 0 & \omega_1 & \omega_5 \\ \omega_4 & \omega_5 & \omega_6 \end{bmatrix} \quad (13.14)$$

四个变量, 三个线性方程, 在常数倍意义下我们可以得到 $\boldsymbol{\Omega}$, 随后运用 Cholesky 分解得到 \mathbf{K} . 之后我们就可以对场景进行三维重建, 例如计算图片中所有平面的方向, 因此一张图片实际上蕴含了周围场景的丰富信息.

在这一节的最后, 我们仍然要指出, 单纯从一张图片是不可能完整重建原 3D 场景的, 这是因为物体的深度和大小之间存在 ambiguity. 在下一节当中, 我们会试图解决这个问题.

第 14 章 Epipolar Geometry

这一章节在 2024 年教学中已经被删去, 为了让有兴趣的读者了解, 故保留

既然单个视角无法确定深度, 那么我们自然会想到如果有多台相机或许就可以确定距离, 如同人眼一样.

原则上, 两台相机就可以确定点的位置. 我们看右图: O_1, O_2 代表两台相机的镜心, 两个平行四边形代表各自的像平面, 如果一个点 P 在两个像平面的投影点 p, p' 已知, 我们就可以确定 P 点在镜心到对应点的连线交点的位置.

在前面的章节当中我们已经介绍了校准相机的方法, 而两台相机的位置参数 \mathbf{R}, \mathbf{t} 一般都是已知的, 此时的关键问题就是: 确定一个点在另一个相机图片的位置.

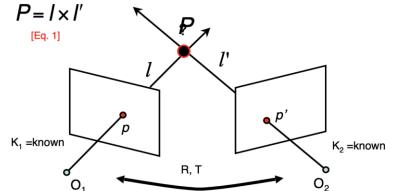
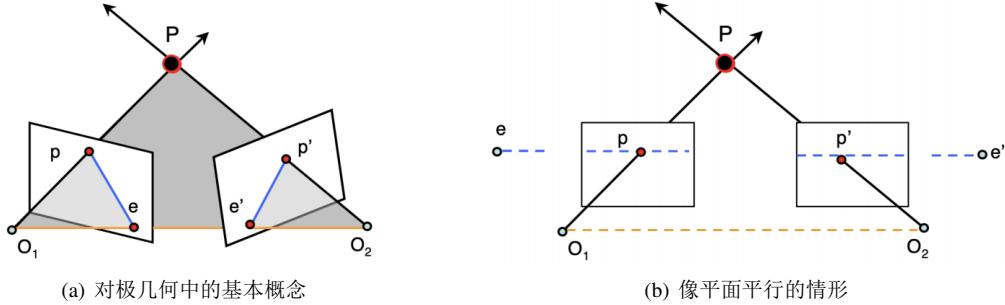


图 14.1: Triangulation



与这一任务相关的几何内容被称为对极几何 (epipolar geometry). 我们先给出对极几何当中的一些概念.

左图中由 O_1, O_2, P 三点构成的平面 (图中灰色填充) 被称为 epipolar plane. 连线 O_1O_2 被称为 baseline. baseline 与两个像平面的交点被称为 epipole. 各个像平面内,epipole 与 P 的投影点被称为 epipolar line.

当 P 点移动时, 相当于极平面上下翻动, 此时对应的极线也会变化, 但是所有的极线都通过极点.

一种特殊情况是两个像平面相互平行, 此时极点位于无穷远处, 图中所有极线都平行于基线. 那么, 给定图片中的一个点, 要如何去确定在另一张图片中这个点的对应呢? 我们看下图, 根据给定点和基线我们可以确定极平面, 极平面与右像平面的交点即为极线, 对应点 p' 必定在此极线上, 也就将对应点的搜索从 2 维降低到了 1 维.

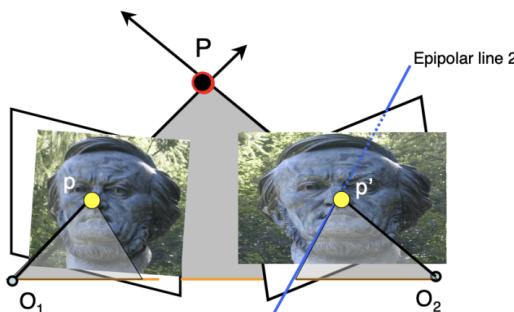


图 14.2: 两张图片的对应

14.1 Epipolar Constraint

首先我们先确定两台相机的位置. 如下图, 方便起见, 假设世界坐标与第一台相机的坐标重合, 第二台相机的坐标系由旋转 \mathbf{R} 和偏置 \mathbf{t} 确定. 对于一点 P , 如果它在两台相机的坐标分别是 $\mathbf{p}_E, \mathbf{q}_E$, 则有关系

$$\mathbf{q}_E = \mathbf{R}^\top (\mathbf{p}_E - \mathbf{t}) \quad (14.1)$$

因此, 两台相机的变换矩阵分别是

$$\mathbf{M} = \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix}, \quad \mathbf{M}' = \mathbf{K}' \begin{bmatrix} \mathbf{R}^\top & -\mathbf{R}^\top \mathbf{t} \end{bmatrix} \quad (14.2)$$

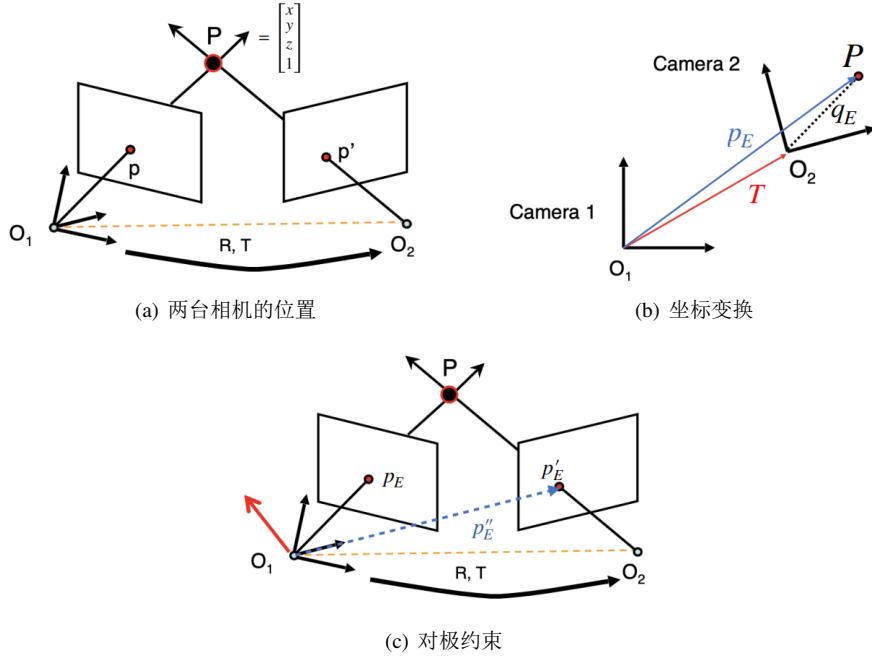


图 14.3

如图 14.3(c), 我们规定以下符号: $\mathbf{p}_E, \mathbf{p}'_E, \mathbf{p}''_E \in \mathbb{R}^3$ 分别代表左投影点在左相机坐标系 (即世界坐标系) 的坐标, 右投影点在右相机坐标系下的坐标, 右投影点在左相机坐标系下的坐标. 我们可以求出极平面的法向量 (图中红色箭头):

$$\mathbf{n} = \mathbf{t} \times \mathbf{p}''_E = \mathbf{t} \times (\mathbf{R}\mathbf{p}'_E + \mathbf{t}) = \mathbf{t} \times \mathbf{R}\mathbf{p}'_E \quad (14.3)$$

对于两个向量 $\mathbf{a} = [a_x, a_y, a_z]^\top, \mathbf{b} = [b_x, b_y, b_z]^\top$ 的叉乘, 可以表示成矩阵形式:

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}_\times] \mathbf{b} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \quad (14.4)$$

不难看出这是一个反对称矩阵, 秩为 2.

显然有 $\mathbf{p}_E \perp \mathbf{n}$, 写成矩阵形式就是

$$\mathbf{p}_E^\top [\mathbf{t}_\times] \mathbf{R} \mathbf{p}'_E = 0 \quad (14.5)$$

我们定义

$$\mathbf{E} = [\mathbf{t}_\times] \mathbf{R} \quad (14.6)$$

为本质矩阵 (essential matrix). 它是一个形状为 3×3 , 秩为 2 的矩阵, 有五个自由度.¹

代入式 14.5 得到:

$$\mathbf{p}_E^\top \mathbf{E} \mathbf{p}'_E = 0 \quad (14.7)$$

虽然有了本质矩阵是在两个相机坐标系下的三维坐标的变换, 我们自然希望能用二维的图片坐标进行直接对应. 考虑到

$$\mathbf{p}_E = \mathbf{K}^{-1} \mathbf{p}, \quad \mathbf{p}'_E = \mathbf{K}'^{-1} \mathbf{p}' \quad (14.8)$$

¹ 在下文的基本矩阵的讨论中, 我们也会涉及到自由度的问题, 这一度令笔者陷入困惑. 这些内容将在附录 C 当中作进一步探讨.

代入式 14.5 即得

$$\mathbf{p}^\top \mathbf{K}^{-\top} [\mathbf{t}_\times] \mathbf{R} \mathbf{K}'^{-1} \mathbf{p}' = 0 \quad (14.9)$$

我们定义

$$\mathbf{F} = \mathbf{K}^{-\top} \mathbf{E} \mathbf{K}'^{-1} = \mathbf{K}^{-\top} [\mathbf{t}_\times] \mathbf{R} \mathbf{K}'^{-1} \quad (14.10)$$

为基本矩阵 (fundamental matrix), 它是一个秩为 2, 自由度为 7 的 3×3 矩阵. 式 14.10 变为

$$\mathbf{p}^\top \mathbf{F} \mathbf{p}' = 0 \quad (14.11)$$

除此之外, 基本矩阵还有如下性质: 如图 14.4, 首先, $\mathbf{l} = \mathbf{F} \mathbf{p}'$ 和 $\mathbf{l}' = \mathbf{F}^\top \mathbf{p}$ 分别是 \mathbf{p}, \mathbf{p}' 对应的极线. 其次, $\mathbf{F} \mathbf{e}' = 0, \mathbf{F}^\top \mathbf{e} = 0$.

先证明前者. 式 14.11 说明 $\mathbf{p} \in \mathbf{l}$. 设 \mathbf{e} 对应的世界坐标为 \mathbf{e}_E , 则由于 $\mathbf{e}_E \perp \mathbf{n}$, 因此式 14.5 及以后的各式将 \mathbf{p}_E 替换成 \mathbf{e}_E 都成立, 那么同式 14.11 的形式, 我们亦可以得到

$$\mathbf{e}^\top \mathbf{F} \mathbf{p}' = 0. \quad (14.12)$$

由此得出 $\mathbf{e} \in \mathbf{l}$.²

对于后者, 我们知道 $\forall \mathbf{p}$, 都有式 14.12 成立, 因此只能有

$$\mathbf{F} \mathbf{e}' = 0, \quad \mathbf{F}^\top \mathbf{e} = 0. \quad (14.13)$$

我们对本质矩阵和基本矩阵做一个总结.

\mathbf{E} 是两个标准相机相对关系的内参, 换言之, 给定两个已标定相机的相对位置关系, \mathbf{E} 即被确定.

\mathbf{F} 是任意两个相机相对关系的内参, 换言之, 给定了两个相机的相对位置关系和相机内参, \mathbf{F} 即被确定.

如果我们能够求出 \mathbf{F} , 那么就可以在不进行三维重建的情况下, 确定两张照片的对应关系. 还记得我们在上节末尾和这节的开头提出的问题吗? 在上一节的末尾, 我们说通过两台相机即可确定深度信息, 在这一节的开头我们引入了对极几何的概念, 我们的目的是确定两张照片中的对应点, 再通过两台相机的方向即可求出点的深度, 而方向已经在上一节中解决. 现在我们离解决这个问题只剩最后一步, 即如何求出 \mathbf{F} ?

我们设 $\mathbf{F} = (F_{ij})$, 根据式 14.11, 如果有 8 对对应点³的坐标 $(u_i, v_i, 1)^\top \leftrightarrow (u'_i, v'_i, 1)^\top$, 那么我们可以得到方程组

$$\begin{pmatrix} u_1 u'_1 & u_1 v'_1 & u_1 & v_1 u'_1 & v_1 v'_1 & v_1 & u'_1 & v'_1 & 1 \\ u_2 u'_2 & u_2 v'_2 & u_2 & v_2 u'_2 & v_2 v'_2 & v_2 & u'_2 & v'_2 & 1 \\ u_3 u'_3 & u_3 v'_3 & u_3 & v_3 u'_3 & v_3 v'_3 & v_3 & u'_3 & v'_3 & 1 \\ u_4 u'_4 & u_4 v'_4 & u_4 & v_4 u'_4 & v_4 v'_4 & v_4 & u'_4 & v'_4 & 1 \\ u_5 u'_5 & u_5 v'_5 & u_5 & v_5 u'_5 & v_5 v'_5 & v_5 & u'_5 & v'_5 & 1 \\ u_6 u'_6 & u_6 v'_6 & u_6 & v_6 u'_6 & v_6 v'_6 & v_6 & u'_6 & v'_6 & 1 \\ u_7 u'_7 & u_7 v'_7 & u_7 & v_7 u'_7 & v_7 v'_7 & v_7 & u'_7 & v'_7 & 1 \\ u_8 u'_8 & u_8 v'_8 & u_8 & v_8 u'_8 & v_8 v'_8 & v_8 & u'_8 & v'_8 & 1 \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{pmatrix} = \mathbf{0} \quad (14.14)$$

我们记上式为

$$\mathbf{W} \mathbf{f} = \mathbf{0} \quad (14.15)$$

如果 $\text{rank}(\mathbf{W}) = 8$, 则 \mathbf{f} 有唯一非零解. 当对应点对数目多于 8 个, 则限制 $\|\mathbf{f}\| = 1$, 使用 SVD 求解. 但是 \mathbf{F}

²细心的读者不难发现, 极平面内所有的点都可以这样做, 而这个结论也是非常自然的, 因为极线正是极平面射影在左像平面的结果.

³这里读者可能会产生一个问题, 既然 \mathbf{F} 的自由度为 7, 为什么要 8 对对应点呢? 其中原因之一是, 相机的内部性质多为非线性, 所以使用最小点数求解可能会比较麻烦, 因此经常只考虑尺度的等价性, 忽略奇异性条件, 此时自由度为 8.

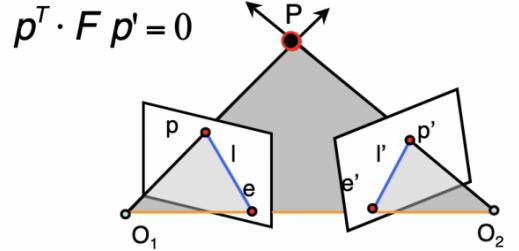


图 14.4: 基本矩阵的性质

的秩为 2, 那么问题转化为

$$\begin{aligned} & \text{minimize} \quad \|\mathbf{F} - \hat{\mathbf{F}}\| \\ & \text{s.t.} \quad \det \mathbf{F} = 0. \end{aligned} \tag{14.16}$$

而 SVD 告诉我们, 设 $\text{rank}(\mathbf{F}) = r$, 若将 \mathbf{F} 按奇异值降序分解为

$$\mathbf{F} = \sum_{i=1}^r \sigma_i \mathbf{u}_i \mathbf{v}_i^\top \tag{14.17}$$

则在所有秩不大于 k 的矩阵当中,

$$\mathbf{F} = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^\top \tag{14.18}$$

就是使得 $\|\mathbf{F} - \hat{\mathbf{F}}\|$ 最小的矩阵. 因此我们需要对式 14.14 求解出的 \mathbf{F} 再次进行 SVD, 并取前两个奇异值和左右奇异向量构成解 \mathbf{F} .

第 15 章 3D Data

15.1 Sensors

第一种 sensor: Depth Sensors

得到的是一个深度图, 但是深度图 RGBD 是 2.5D 的, 如果不知道相机内参, 不能完全表示 3D 的信息

第二种 sensor: Stereo Sensors

Pros:

1. Robust to the illumination of direct sunlight
2. Low implementation cost

Cons:

1. Finding correspondences along Image_L and Image_R is hard and erroneous

可以升级为 Active Stereo, 之前的双目相机是 passive, 现在看看打光的主动识别, 在两个相机之外加入一个红外线发射器

但是红外线主动 sensor 仅限于室内使用, 室外使用会受到阳光的干扰, 并且不能处理透明或者非漫反射的物体

第三种 sensor: time-of-light

即用光的传播时间计算距离, 具体有如下两种

dTOF(direct Time of Flight)

1. 发射脉冲波, 通过光的传播时间计算距离, 被认为是比 iTOF 更好的方法
2. 理论上精度高, 但是现在分辨率还很低
3. 对器件要求高, 需要使用 SPAD (单光子雪崩二极管), 无法做得很密集, 造价高

iTOF(indirect Time of Flight)

1. 发射正弦波, 通过光的相位差计算距离
2. 精度低 (误差与距离成正比), 但是分辨率高
3. 造价低

Summary of Different Depth Sensors

15.2 Multiple 3D representation

1. Multiple images. 从各个视角获得的多张图片. 它包含 3D 信息, indirect, not a true 3D representation.

CONSIDERATIONS	STEREO VISION	STRUCTURED-LIGHT	TIME-OF-FLIGHT (TOF)
Software Complexity	High	Medium	Low
Material Cost	Low	High	Medium
Compactness	Low	High	Low
Response Time	Medium	Slow	Fast
Depth Accuracy	Low	High	Medium <i>Quickly improving!</i>
Low-Light Performance	Weak	Good	Good
Bright-Light Performance	Good	Weak	Good
Power Consumption	Low	Medium	Scalable
Range	Limited	Scalable	Scalable

图 15.1: Summary of Different Depth Sensors

2. Depth image. 只知道深度, 不知道相机内参, 无法确定两个点的距离. 因此被称为 2.5D.
3. Voxels. 物体占据了位置则设为 1. 能被 index. 但是非常昂贵. $O(n^3)$. 没有表面的表示. 一旦分辨率低, 则无法还原丢失的信息.
4. Irregular 3D representation. Mesh, point Cloud, Implicit representation.

15.3 Mesh

在表面取点, 用过三点的平面近似表面. 实际上也不限于三角形. 下面我们讨论 triangle mesh.

$$\begin{aligned} V &= \{v_1, \dots, v_n\} \in \mathbb{R}^3 \\ E &= \{e_1, \dots, e_n\} \in V \times V \\ F &= \{f_1, \dots, f_n\} \in V \times V \times V \end{aligned} \tag{15.1}$$

好的 mesh: watertight(不漏), manifold(外法向量连续).

15.4 Point Cloud

Point cloud 是一个 $3n$ 级别的存储方式, 当然也可以添加其他分量. 它是不规则 (irregular) 且无序的数据. 换言之, 其数据的序并不是必要的, 不能提供额外信息, 如同 set 一样. 好的算法应该尽可能不去运用序. 它是非常轻量级, 紧致的, 线性级别的存储空间. 容易存储, 容易理解和生成, 容易在其上构建算法.

当然它也不是完美的, 比如并不容易通过它判断表面的位置. 点云实际上是在表面上采样, 那么怎样从一个 mesh surface 上取样呢?

Sampling Strategy: Uniform Sampling. 计算每个表面的面积, 以面积为权独立同分布地选取三角形. 在三角形里如何均匀选取呢? 仿射变换形成直角三角形, 再拼接成矩形.¹

但是这样取样之后, 在取样量不算非常大的时候, 经常会出现不太均匀的情形. 此时我们可以采用 Farthest Point Sampling(FPS). 目标是选取 N 个点使其两两距离求和最大. 但这是一个 \mathcal{NP} -hard 的问题. 我们采取一个近似的贪心算法. 我们先均匀选取 10000 个点. 在这 10000 个点最远的 1024 个点成为组合优化问题. 但仍然比较困难. 我们先确定一个点, 最后依次选取最远的点. 我们也只是希望点云看起来比较均匀, 因此没必要一定取得数学上的最优解.

除此之外, 点云的距离度量也成为一个问题, 这也是无序带来的问题之一. 我们希望找到一个 permutation invariant 的度量: Chamfer distance.²

$$d_{CD} = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2 \tag{15.2}$$

对于每个单项, 称为 uni chamfer distance. 在一个点云是另一个子集的时候有用. 并且这里似乎应该除以二才能和 EMD 进行比较.

另一个度量是 Earth Mover's distance³. 与 CD 不同的是, 它要求两个点云数量相同, 且每个点必须找到互不重複的对应.⁴

$$d_{EMD}(S_1, S_2) = \min_{\phi: S_1 \rightarrow S_2} \sum_{x \in S_1} \|x - \phi(x)\|_2 \tag{15.3}$$

¹直接拼接成平行四边形如何? 对三角形均匀采样可以在正方形(平行四边形)中采样然后对角线截半.

²在某些文献当中, 式 15.2 有时也会带平方. 但是如果带平方, 则三角不等式不成立.

³在 WGAN 中使用.

⁴Earthmover, 中文直译为推土机. EMD 距离用于衡量(在某一特征空间下)两个多维分布之间的 dissimilarity, 它的计算基于著名的运输问题. 但是精确求解 EMD 亦非易事, 调用的 package 也多为近似算法.

CD 对于取样情况不太敏感, 而 EMD 则比较敏感. 比如同样对于 Stanford bunny, 如果一个点云多集中在头部, 另一个比较均匀, 则 CD 变化不大而 EMD 变换显著. 由于点云是 surface+sampling, 因此如果对于取样有要求, 应该使用 EMD.

15.5 CD vs EMD

区别在于 CD 对于采样不敏感, 而 EMD 对于采样敏感.

同时 EMD 需要两个点云点的数量相同.

也就是如果 mesh 一样但是 sample 不一样, 那么 EMD 会很大

15.6 Implicit Representation

对于一个点, 使用一个关于这个点的函数的值来表示这个点是否在物体表面

具体的表示方法是隐式表达, 即 SDF(signed distance function)

简单地说, 这个函数表示空间中点到物体表面的距离, 内部为负, 外部为正.

其数学定义为: 设 Ω 为度量空间 X 的子集, d 为 X 的度量, 则 SDF 定义为

$$f(x) = \begin{cases} -d(x, \partial\Omega) & \text{if } x \in \Omega \\ d(x, \partial\Omega) & \text{if } x \in \Omega^c \end{cases} \quad (15.4)$$

其中 $d(x, \partial\Omega) \stackrel{\text{def}}{=} \inf_{y \in \partial\Omega} d(x, y)$.

SDF 的用处:

1. **collision check:** 如果一个东西跟另外一个东西的碰撞. 那么其中物体 A 它上面的所有点到底有没有碰到物体 B 呢? 我们就可以通过 query 所有点在 B 的物体的 SDF 里头它的值是多少. 如果都是大于 0, 说明 A 和 B 没有碰撞
2. 使用神经网络把物体的几何通过 SDF 的方式存储下来

Mesh to SDF

在处理 Mesh 到 Signed Distance Field (SDF) 的转换时, 首先需要一些坐标点, 基于这些点的距离值, 可以尝试开发算法来计算对应的 SDF 值

15.6.0.0.1 Calculate Unsigned Distance 在几何处理中, 一种基础的方法是计算点到表面的距离 (Unsigned Distance), 不区分点是在物体内部还是外部. 这可以通过将所有三角面视为平面, 对于每个点, 计算其到每个三角面的垂线距离. 如果垂足不在三角面上, 则该距离不是最短距离; 如果垂足位于三角面上, 则从这些距离中找出最短的, 即为该点到表面的最短距离.

15.6.0.0.2 Unsigned Distance to Signed Distance 要将 Unsigned Distance 转换为 Signed Distance, 使用等高线将空间中的点进行分层表示. 这些等高线描绘了在某些区域距离先减小然后增加, 其中距离最近的点附近的区域距离为零, 利用距离为零的区域判断点是在物体的内部还是外部.

SDF to Mesh

使用 marching cube

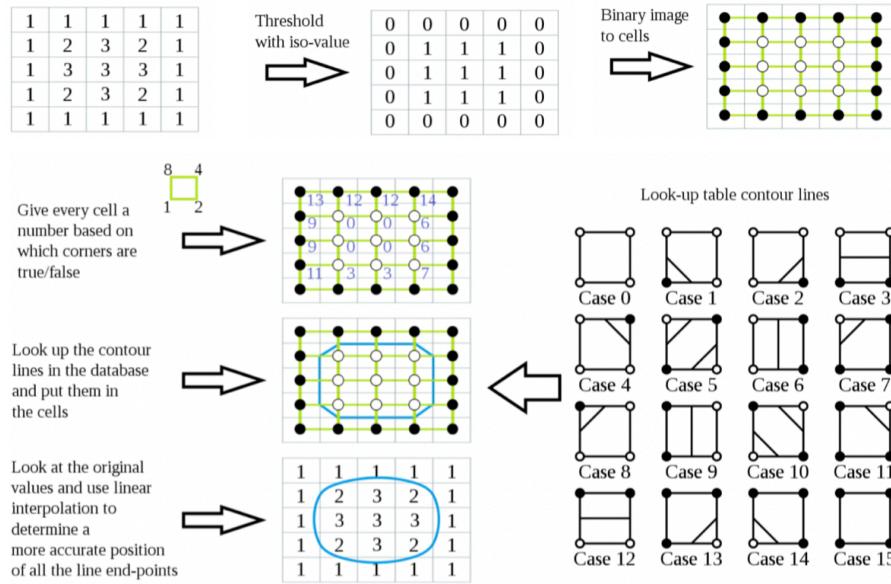


图 15.2: Marching cube pipelines

1. 离散化
2. 二值化
3. 使用 look-up table 找轮廓
4. 使用线性插值优化轮廓

在第三步, 很多 case 都会有这种 ambiguity, 如图 15.3, 所以还需要根据周围的 mesh 二次优化, 决定 ambiguity 的位置, 对于三维 mesh 重建, 这个问题更加严重.

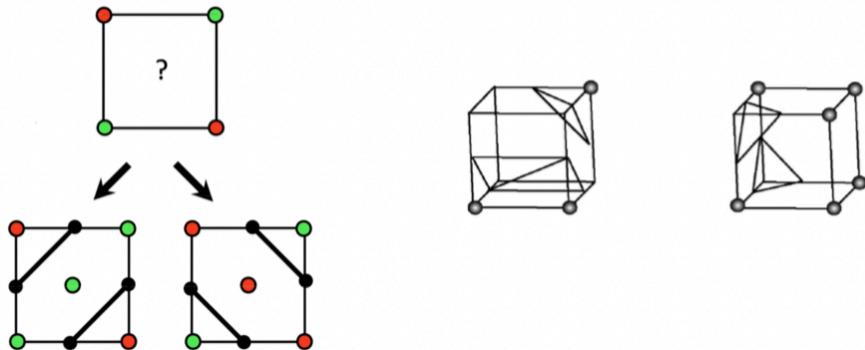


图 15.3: Problem in mesh reconstruction

15.7 Deep SDF

两个 trick:

1. 如何分布训练数据点: 如果这个神经网络 SDF 最终需要使用 marching cube, 那么应该在 SDF 为 0 附近多多采样
2. 训练 loss 的设置, 可以加入一个约束优化: 让 SDF 的梯度为 1. 这样可以让 SDF 更加 smooth
SDF 还满足 Eikonal equation, 即

$$\|\nabla F\| = 1. \quad (15.5)$$

这点不难理解, 因为沿与距离最短的点的连线方向的方向导数之模为 1, 而函数本身即表达最短距离, 也就是场变化最快的方向, 不可能沿着梯度前进一个单位的长度, 距离变化却超过一单位, 因此这也是最大值.

将不同的 SDF 关联起来, 并且保证连续性 [Park2019CVPR]

1. use the network to overfit a single shape
2. use a latent code to represent a shape, so that the network can be used for multiple shapes

15.8 NeRF, 3DGS, Convolution on Mesh/Graph

了解即可

第 16 章 3D Deep Learning

对于点云来说, 直接拉成 vector 显然是不行的. 因为点云本身无序. 有一些直接的处理方法: 转换成 voxel grid(从点云到表面, 再重建 voxel), 或者 2D 投影 (这属于是越活越回去了). 我们想要的 network 需要对 $N!$ 种不同的点云顺序具有不变性.

一种想法: 排序赋予序关系如何?

我们用 Permutation Invariant 的函数:Symmetric Function. 也就是

$$f(x_1, x_2, \dots, x_n) \equiv f(x_{\pi_1}, x_{\pi_2}, \dots, x_{\pi_n}), \forall x \in \mathbb{R}^N \quad (16.1)$$

实际上, 这样的函数随处可见. 比如均值, 极值等. 文献 [PointNet] 、 PointNet.¹

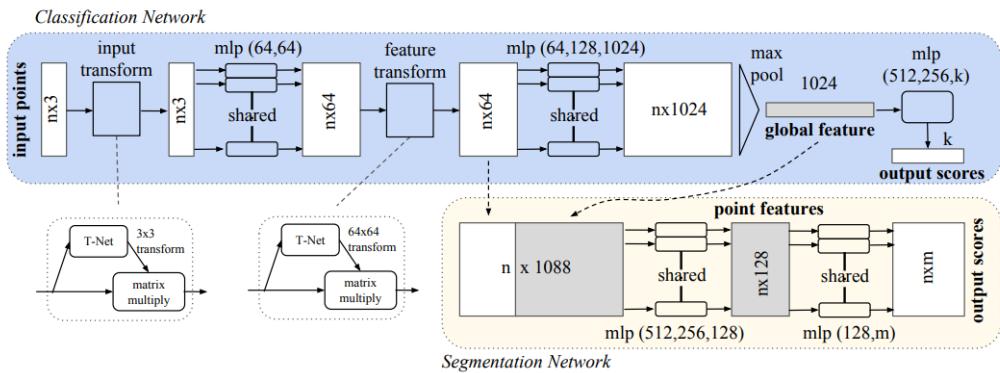


Figure 2. **PointNet Architecture.** The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. “mlp” stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net.

图 16.1: PointNet 的结构图

做 segmentation: 将每个点自身的特征 (local) 和 global feature 结合, 过 MLP 就能进行分类.

诸多挑战: Resolution,Occlusion,Noise,Registration.

PointNet 对于 Resolution 和 Noise 比较 robust. 在 ModelNet40 上, 去掉 50% 的数据,Furthest 的准确率只下降了 2%. 能够如此稳定的关键在于 max 函数, 在连续取样时, 一些点的丢失可以用周围的点来弥补. 而且如果 pointnet 只有 1024 个 feature, 那么它最多也只会选取 1024 个点, 可能去掉的点根本就没有用到.

PointNet 的不足

一步直接 maxpooling, 没有 local context. 但是 PointNet 仍是目前不追求高性能时对 point cloud 处理的首选方法.

因为这个不足, 有了 PointNet++. 即设定一定的范围, 将某个点及其周围的点过 MLP, 再 maxpooling. 等同于在 image 里进行 conv. 当然, 求 neighbour 可能比较花时间. 另外, 所有的 neighbour 都通过了同样的 MLP, 毕竟在点云里也无法和图像一样, 使用卷积核直接提取局部特征.

PointNet++ 怎么进行下采样? 2d 中是 pooling, 而 pointnet 则进行 grouping

PointNet++ 怎么进行上采样? 使用 Skip-Link, 将下采样产生的点输入回来.

上采样时候添加的点没有 feature, 所以进行 interpolate, 新的点选择与它最近的三个点, 按照距离反比为权插值获得特征, 这里 skip-link 是必要的, 否则无法获得点的具体位置.

¹王老师：“虽然我不在这个组里, 但 PointNet 这篇论文的诞生我也是亲眼目睹过的, ICCV 的投稿截止在 11 月, 9 月底大家还不知道投什么, 于是有人说要不试试 PointNet 吧. 写到最后只有两个周调参了, 还是打不过 voxel, 于是加了 T-transform. 我们这里不讲, 因为现在几乎没有用这个的了”.(笑)

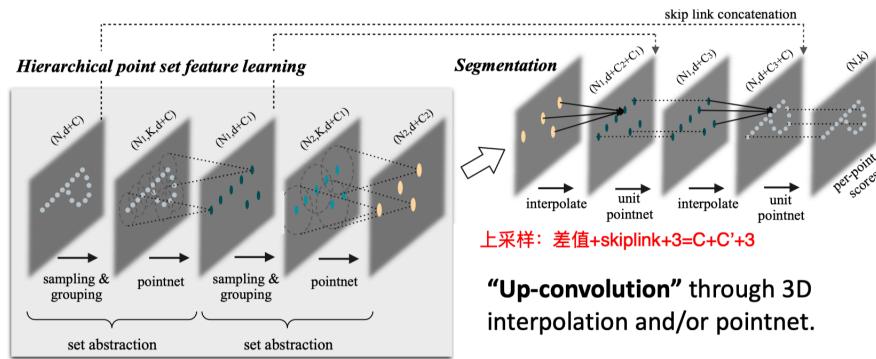


图 16.2: PointNet++ Pipeline

16.1 Sparse Conv

3D:voxel Net → sparse conv, 屏蔽没有值的 voxel, Hash 有值的坐标, 从 NVIDIA,cuda 的层次写起.

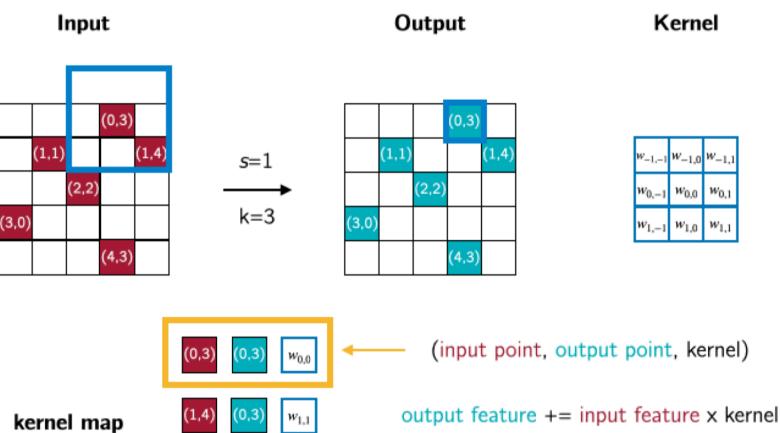


图 16.3: Sparse Conv

结合图示16.3来理解稀疏卷积的过程:

1. 输入特征图: 图中的红色方块表示非零值的位置, 比如在输入特征图中, 位置 (0,3)、(1,1) 等具有非零值.
2. 卷积核: 一个 3×3 的卷积核, 包含权重参数 $w_{-1,-1}$ 等.
3. 输出特征图: 稀疏卷积的输出特征图. 只有当卷积核的中心对准输入特征图的非零值时, 才会计算输出值.
4. 稀疏卷积过程: 当卷积核的中心对准输入特征图的 (0,3) 位置时, 计算对应的输出值. 例如, 卷积核中心对准 (0,3) 时, 只有卷积核的中心权重 ($w_{0,0}$) 参与计算, 输出值为:

$$\text{output}(0,3) = \text{input}(0,3) \times w_{0,0}$$

当卷积核中心对准 (1,4) 位置时, 计算对应的输出值, 输出值为:

$$\text{output}(1,4) = \text{input}(1,4) \times w_{1,1}$$

5. 通过这种方式, 稀疏卷积仅计算非零值对应的输出, 从而减少了大量的计算和内存开销.

Sparse conv:kernel 是空间各向异性 (spatial anisotropic) 的. 因此它的 acc 能够达到惊人的 70% 以上, 而 point cloud 无论如何添加特性, 最多也在 64% 左右. 而且因为可被标签, 使用更加方便. 但是它的分辨率受限.

Sparse Conv Pros and Cons?

Sparse Conv 优点:

1. 比 dense conv 更高效, 稀疏卷积只对非零值进行计算, 跳过零值, 从而显著减少了计算量和内存使用, 提高了

计算效率.

2. 支持索引的网格, 稀疏卷积仍然在规则的网格上操作, 这意味着可以利用现有的索引机制来高效地访问数据.
3. 与 2D Conv 具有类似的表达能力
4. 与 2D Conv 具有类似的平移等变性

Sparse Conv 缺点:

1. 离散化误差, 由于稀疏卷积只对非零值进行计算, 可能会导致离散化误差. 这种误差源于对数据的稀疏表示, 可能会影响卷积操作的精确性.

Sparse Conv vs. Point Cloud Networks?

Sparse Conv:

1. 优点: 核是空间各向异性的
2. 优点: 更高效的索引和近邻查询
3. 优点: 适用于大规模场景
4. 缺点: 分辨率有限

Point Cloud Nets:

1. 优点: 分辨率高
2. 优点: 鲁棒性强
3. 缺点: 表现稍差一些
4. 缺点: 执行 Farthest Point Sampling 和 Ball Query 等操作较慢

16.2 PointNet++ V.S. Conv

PointNet++ 相当于卷积网络的先做 1×1 conv 然后 $n \times n$ MaxPooling, 都是变少了点的数目, 但是每一个点的特征变多了

区别在于: PointNet++ 对于一个局部部分, 不同的点是各向同性的, 但是卷积网络对于一个局部是不同的, 因为一个卷积核不同位置的权不一样.

具体表现在: 如果把一个邻域内的点更换位置, 那么 PointNet++ 的结果不变, 但是卷积网络的结果会变.

这是由于两个原因:

1. 卷积网络的卷积核是 3×3 的, 而 PointNet++ 的 MLP 是全局的, 相当于 1×1 的 CONV, 而 1×1 的 CONV 很容易想到是各向同性的, 如果把一个邻域内的点更换位置, 那么每个点的 MLP 结果不变.
2. 卷积网络的 MaxPooling 不是全局的, 而 PointNet++ 的 MaxPooling 是全局的, 因此如果把一个邻域内的点更换位置, 那么卷积网络的结果会变, 而 PointNet++ 的结果不变, 因为怎么变换位置都是取最大值.

PointNet: $C \times C'$ params

ConvNet: $3 \times 3 \times C \times C'$ params

16.3 设计一个二维 PointNet

如何设计一个二维的 pointnet 呢?

我们可以将图片看作二维的点云, 对于每一个像素 x_i , 将其颜色或者透明度等信息看作是 x_i 的三维坐标, 然后使用 pointnet 的方法进行处理

具体来讲, 图片维度为 $H \times W \times C$, 那么我们可以将其看作 $H \times W$ 个点, 每个点的特征是 C 维的, 每一个像素通过相同的 MLP 映射到 C' 维, 然后进行 maxpooling, 最终得到 C' 维的全局特征.

第 17 章 Sequential Modeling

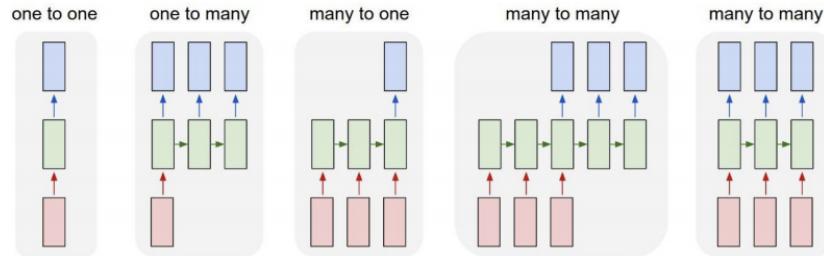


图 17.1: 各种输入输出方式

我们之前的网络, 都是 one-to-one, 一个输入, 一个输出.

one-to-many: 例如看图说话, Image Captioning, image → sequence of words

many-to-one: 例如动作预测.

many-to-many: 例如 video captioning, 或者 Video classification on frame level 可以看作是沿着时间维度的 action segmentation.

此外还有 offline 和 online 的处理. 前者全部看完, 后者边看边输出.

17.1 Recurrent Neural Network (RNN)

还有 recursive NN, 但是不常用

Key point: RNNs 有一个在处理序列时会更新的 hidden state, 就是 h

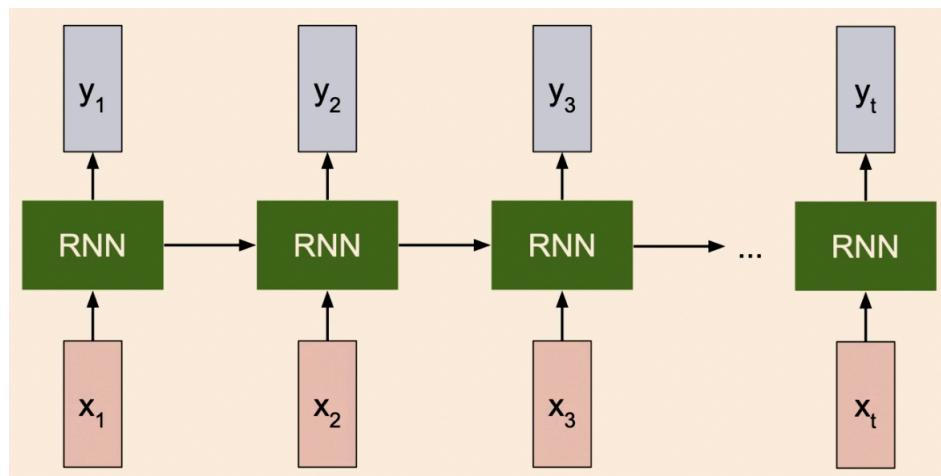


图 17.2: RNN: 典型的 many-to-many

值得注意的是, RNN 的架构很灵活, 可以容易改变输出输出从而变成 many-to-one, one-to-many 等等

$$\begin{cases} h_t = f_W(h_{t-1}, x_t) \\ y_t = f_{W_{hy}}(h_t) \end{cases} \quad (17.1)$$

输出的 y_t 作为一个 decoder, 在需要输出结果的时候输出, 否则不输出

最开始的 h_0 一般是全零的, 也可以是随机的, 只要一样就行

17.2 Vanilla RNN

最原始的 RNN

P.S. 叫做 vanilla 是因为它是最原始的

$$\begin{cases} h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t = W_{hy}^*h_t \end{cases} \quad (17.2)$$

计算图, 各个方向流向 W:

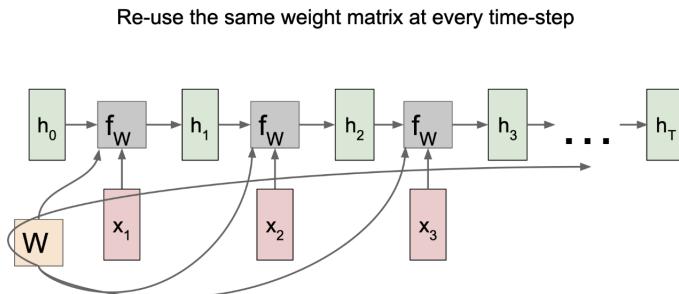


图 17.3: Vanilla RNN Computational Graph

Characer-Level Language Model

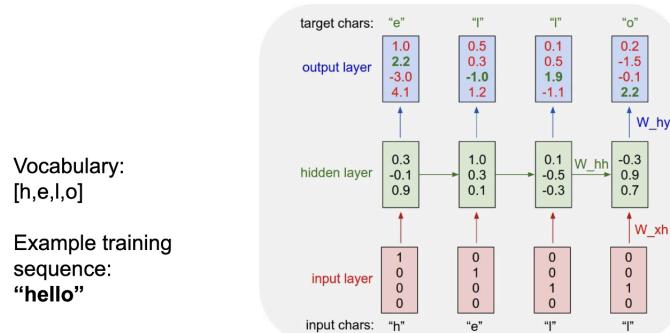


图 17.4: Characer-Level Language Model

先考虑简单的模型, character-level language model, 有限集合的字符, 比如英文 26 个字母, 逗号, 句号等等 BP 时, 不同位置的 W 被 loss 调用了不同次. 这个操作开销极大.

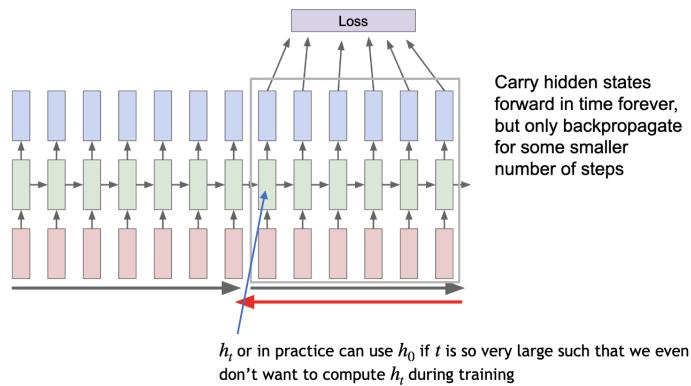


图 17.5: Truncated BP

解决方法:truncated BP 截断回传. 比如 FP 时是从 0 到 t, BP 只算从 t 到 t-6.

这个 window length 是 sequence length

Why share weight?

就像 CNN 一样, 我们希望词特征是等变的. 同一句话, 在文章中的不同位置, 应该具有相同的特征.

Different Sampling Strategies

- Greedy sampling: 总是取最高概率的 token
问题: 确定性, 只能在给定初始标记和隐藏状态的情况下生成一个序列
- Weighted sampling: 根据预测的概率分布采样下一个标记
优点: 可以生成多样的序列
问题: 可能会意外获得错误的标记并造成生成过程的错误
- Exhaustive Search

$$\begin{aligned} P(y|x) &= P(y_1|x)P(y_2|y_1, x)P(y_3|y_1, y_2, x) \dots, P(y_T|y_1, \dots, y_{T-1}, x) \\ &= \prod_{t=1}^T P(y_t|y_1, \dots, y_{t-1}, x) \end{aligned}$$

- Beam Search: 最有可能的前 K 个, 下次从 KV 结果中选择 K 个最有可能的

实际上将 h 和 x 结合的时候, 常常先将 x 进行 embedding, $W(h, x) \rightarrow W(h, g(x))$. 比如, hidden layer 可能之后 512 维, 但是输入如果是词向量, 那么可能有几万维, 非常不均衡. 因此将 embedding 到合适的维度. 此外, word embedding 已经有现成的处理.

RNN 在长程记忆里会出现问题, ΔT 一般被称为 sequence length, 如果过短则相关性不强, 过长则 cost too much.

17.3 RNN tradeoff

- RNN 优点:
 - 可以处理任何长度的输入
 - 步骤 t 的计算 (理论上) 可以使用许多步之前的信息
 - 模型大小不会因输入更长而增加
 - 在每个时间步上应用相同的权重, 因此输入处理方式是等变的
- RNN 缺点:
 - 循环计算很慢
 - 在实践中, 很难从许多步骤前获取信息

17.4 Multilayer RNNs

增强 RNN 非线性的能力

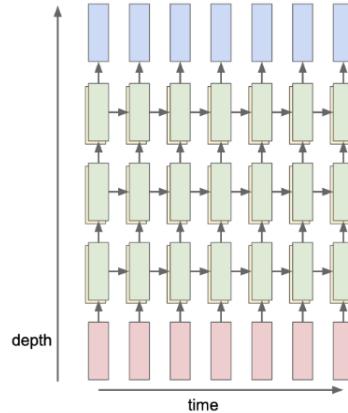


图 17.6: Multilayer RNN

17.5 Vanilla RNN Gradient Flow

Gradients over multiple time steps:

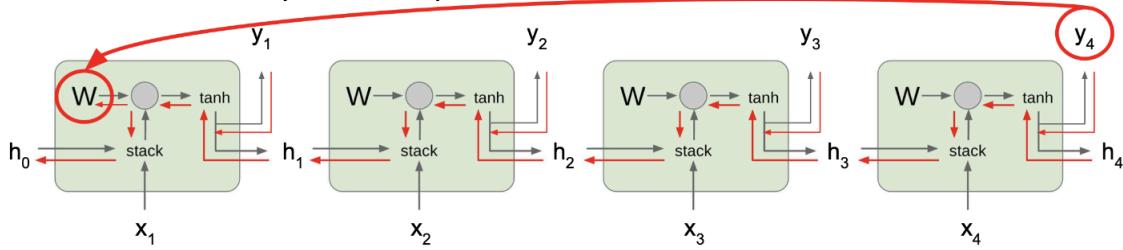


图 17.7: Vanilla RNN Gradient Flow

$$\begin{aligned}
 h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
 &= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
 &= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
 \end{aligned}$$

\tanh 的梯度恒小于 1, 梯度消失:

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^T \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}$$

远处的梯度信号丢失了, 因为它比近处的梯度信号小得多, 模型权重仅针对近期进行更新, 而非长期

gradient clipping 对于 exploding grad 有一定作用, 但是解决不了 vanishing gradient 的问题. 必须 Change RNN architecture

17.6 LSTM

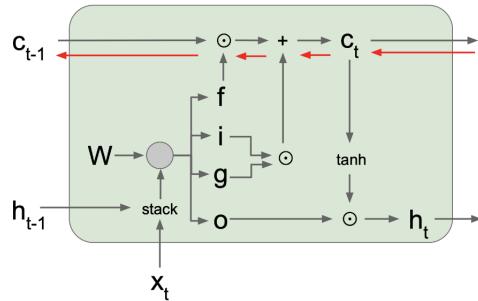


图 17.8: LSTM Gradient Flow

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

- i: Input gate, Whether to write to cell
- f: Forget gate. Whether to erase cell
- o: Output gate, How much to reveal cell
- g: Gate gate (?), How much to write to cell

cell state 是 long-term memory. 我们知道对于普通的 \tanh , 往 01 之间映射, 那么 h_t 和很早之前的 h_i 之间的联系就比较微弱了. 而 LSTM 中的 c 则一直把信息保留 (f 也是经过 sigmoid 激活的, 在 01 之间, 可以选择记住或者遗忘), 最后 h 将 c 进行处理.

核心结构: old info 和 new info 的加和. 若 $f=1$, 则就是 skip link. 换言之 c 之间有梯度的旁路.

因为 c 是旁路, 梯度可以直接流过来

Do LSTMs Solve the Vanishing Gradient Problem?

不能保证. 但是这种 skip link 的结构可以缓解这个问题.

Long term dependencies 的问题得到了很好的解决

17.7 Gated Recurrent Unit (GRU)

$$\begin{aligned} r_t &= \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \\ z_t &= \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \\ \tilde{h}_t &= \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \\ h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \end{aligned}$$

性能不输 LSTM, 有的时候还更好

了解即可

17.8 Summary

LSTM 是一种 RNN, 所有 Sequencial Model 都是 RNN, 只是不是 vanilla RNN

第 18 章 Video analysis

video=2D Dims+time Dim

18.1 Problems: Videos are Big

数据量太过庞大

SD (640 x 480): 1.5 GB per minute HD (1920 x 1080): 10 GB per minute

Solution: Train on short clips, low fps and low spatial resolution e.g. T = 16, H=W=112 (3.2 seconds at 5 fps, 588 KB)

18.2 Video Classification: Single-Frame CNN

Simple idea: 训练 2D CNN 分类视频每一帧, 然后取平均.

这在 video classification 的类别差异比较大的时候, 是一个很强的 baseline

18.3 Video Classification: Late Fusion with FC

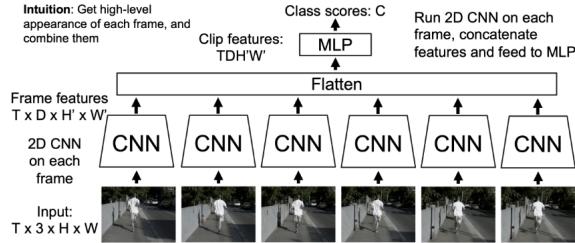


图 18.1: Late Fusion with FC

看得出来正着走和倒着走的区别吗? 可以看出来. Concatenate 是有顺序的, 但是没有使用时间 equalvariance 的信息.

18.4 Video Classification: Late Fusion with Pooling

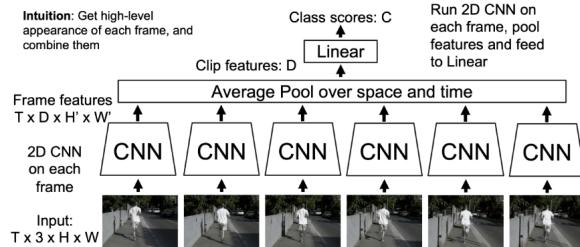


图 18.2: Late Fusion with Pooling

看得出来正着走和倒着走的区别吗? 看不出来

看得出来跑和走的区别吗? 看得出来, 分辨重心 feature.

更好得区分跑和走? 使用 MaxPool, 在 fusion 的时候不要模糊所有帧的特征

18.5 Video Classification: Early Fusion

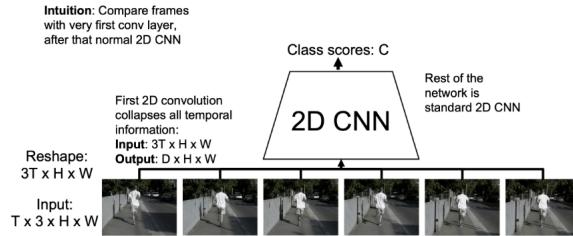


图 18.3: Early Fusion

early fusion: 所有时间信息在第一步提取, 难度较高
所以一层处理时间信息可能不够

18.6 Video Classification: 3D CNN

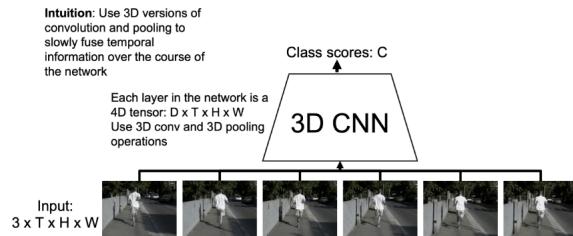


图 18.4: 3D CNN

3D CNN 将时间视作第三个空间维度. 但问题在于: 这对感受野的要求比较大.

在视频当中, 一件事情的效应可能在相当长的时间之后才能体现, 而 3D CNN 在各个维度感受野的增长又比较均匀, 3×3 的时间维度卷积可能远远不如 3×3 的图片卷积对于最终的分类结果用处大, 从这里可以看出, 在视频处理当中, 空间范围和时间范围地位并不对等. 但 CNN 也不可能 cover 任意长的时间序列, 这就是 3DCNN 的局限性.

到目前为止, 我们的所有时 3DCNN 仅在大约 2-5 秒的视频片段中建模帧之间的局部运动, 无法提取长期结构特征.

18.7 Comparison

3D CNN 与 2D CNN 相比, 把卷积核扩展到时间维度, 感受野在时间维度上也是逐渐增大的.

更重要的是 Late Fusion 或者 Early Fusion 是没有办法处理时间局部信息的, 3D CNN 是可以处理时间局部信息的, 因为它有时间维度上的卷积操作.

- Late Fusion: Build slowly in space, All-at-once in time at end
- Early Fusion: Build slowly in space, All-at-once in time at start
- 3D CNN: Build slowly in space, Build slowly in time "Slow Fusion"

18.8 C3D: The VGG of 3D CNNs

C3D: The VGG of 3D CNNs. 第一次 pool 没有在时间维上处理, 不希望提取得太早.

	Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Late Fusion	Input	3 x 20 x 64 x 64	
	Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3
	Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6
	Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14
	GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64
Early Fusion	Input	3 x 20 x 64 x 64	
	Conv2D(3x3, 3*20->12)	12 x 64 x 64	20 x 3 x 3
	Pool2D(4x4)	12 x 16 x 16	20 x 6 x 6
	Conv2D(3x3, 12->24)	24 x 16 x 16	20 x 14 x 14
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64
3D CNN	Input	3 x 20 x 64 x 64	
	Conv3D(3x3x3, 3->12)	12 x 20 x 64 x 64	3 x 3 x 3
	Pool3D(4x4x4)	12 x 5 x 16 x 16	6 x 6 x 6
	Conv3D(3x3x3, 12->24)	24 x 5 x 16 x 16	14 x 14 x 14
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64

图 18.5: Video analysis method comparison

3d 的 kernel size 比较敏感, $5^3 > 11^2$. 移动多了一个维度, 计算代价更大. 得益于其网络的精良设计, C3D 在 sports-1m 数据集上的 top5 acc 还是达到了 84%.

Problem: 3x3x3 conv is very expensive!

人类识别运动的关键: Motion, 不是 pixel.

Use Both Motion and appearance: two-stream fusion network. 它使用经典算法获得 flow 输入神经网络. 神经网络分为两支, 一支做空间, 一支做时间. 时间 (flow) 这一支使用了 early fusion, 因为相对于 RGB, 光流的信息已经比较清晰.

18.9 Modeling Long-Term Temporal Structure

我们希望处理序列, RNN 如何? aggregation(聚合).

CNN 和 RNN 一起训练计算代价太大. 因此先 train CNN, 不向其传递梯度, 只训练 RNN. 但这样 CNN 与 RNN 可能优化目标不同.

如何 pre-train? 使用 Imagenet 不好, 因为 ImageNet 的分类和 Sports1M 的分类粒度不同.

end to end training: 端到端训练. 所有 optimization variable 同时被优化.

18.10 Recurrent convolutional Network

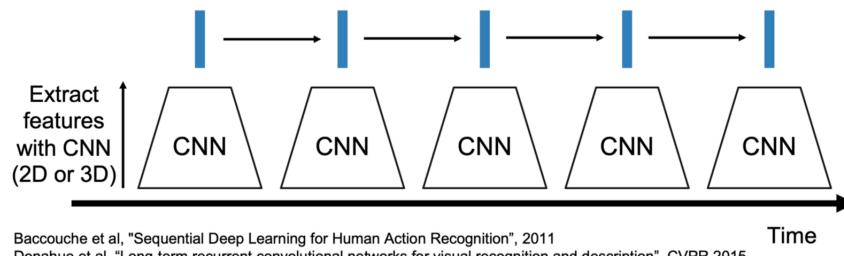


图 18.6: Recurrent Convolutional Network

- 在 CNN 中每个值是固定时间帧 (代表了局部时间结构) 的函数
 - 在 RNN 中每个向量是所有先前向量 (代表了全局时间结构) 的函数
- 把这两个网络合并起来?

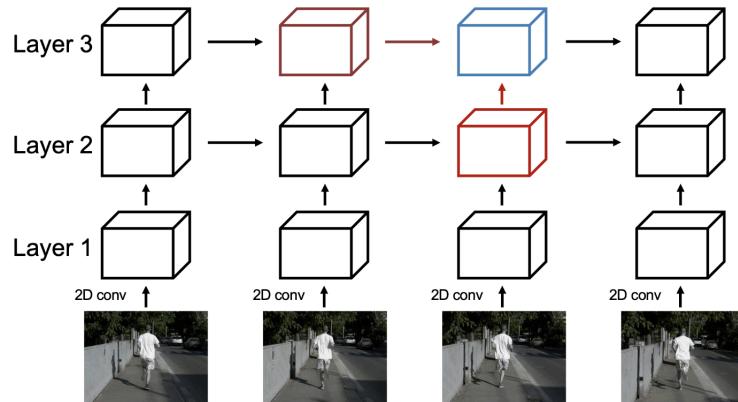


图 18.7: Recurrent convolutional Network

每层依赖于两个输入

- 同层, 前一个时间步
- 前一层, 同一个时间步

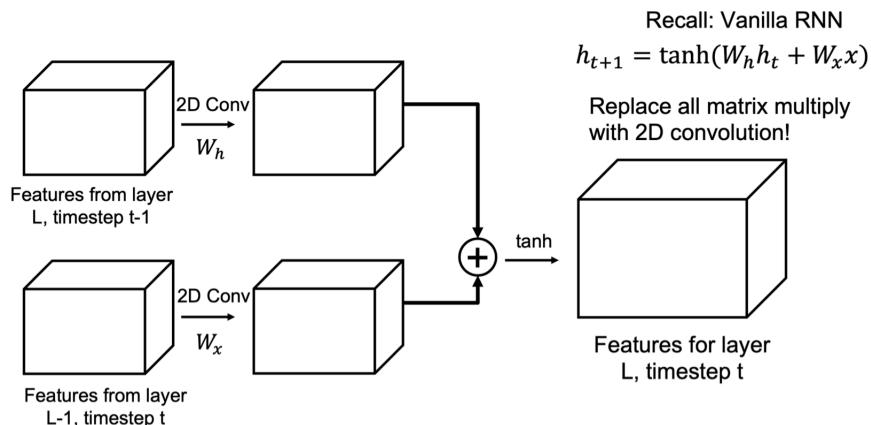


图 18.8: How to combine CNN and RNN

在每一层使用不同的权重, 在时间上共享权重

问题在于循环神经网络不能并行训练, 因此训练很慢.

跟 3D CNN 区别: 3D CNN 不是 recurrent 的, 是被 3D 卷积核决定的一个固定长度, 而 recurrent convolutional network 是一个 recurrent network, 可以读取到之前所有的信息.

第 19 章 Transformer

19.1 Attention

我想可能看代码更方便理解, 这是一个非常好的网站 [Multi-Headed Attention](#)

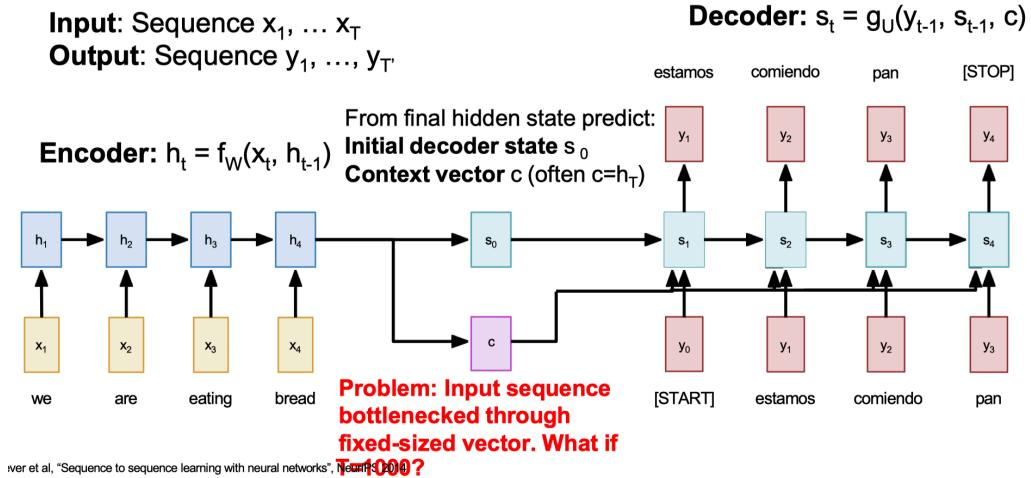


图 19.1: seq2seq

这样的 c : Context vector 太小了, 无法表达所有句子的信息, 所以考虑使用一个自适应的 Context vector 表达每一个 token 的信息, 这就是 attention 的思想.

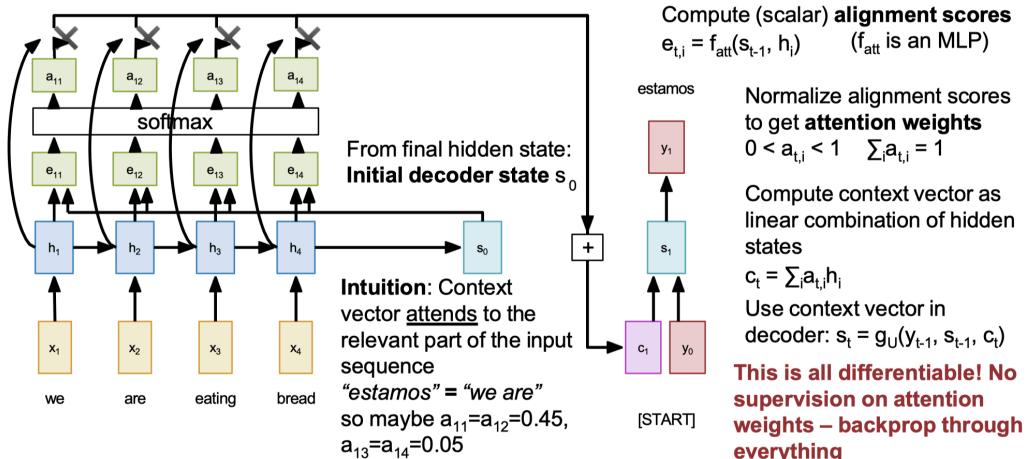


图 19.2: Attention

对应到 Transformer 中, Query 是 s , Cross attention 的输出就是 c .

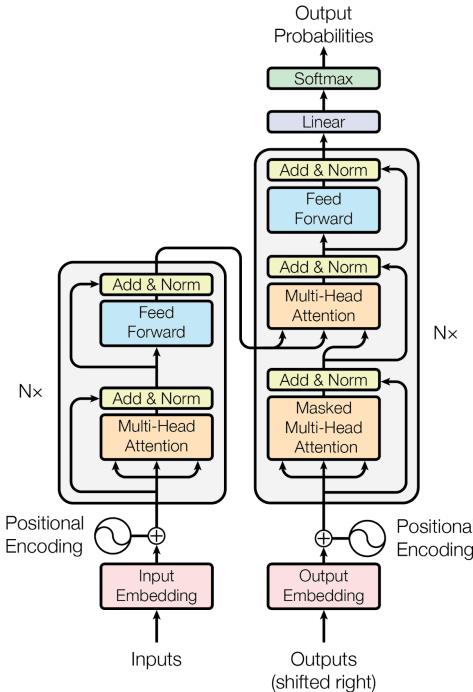


图 19.3: Transformer

如何把 attention 应用到图片中呢? 这和语言中的基本单元类似, 语言中的基本单元在 embedding 后已经不是一个单词了, 而是一个个 token. 考虑把卷积之后的每个像素的所有 feature 看作 token, 然后进行类似结构即可.

Problem: Input is "bottlenecked" through c

- Model needs to encode everything it wants to say within c

This is a problem if we want to generate really long descriptions? 100s of words long

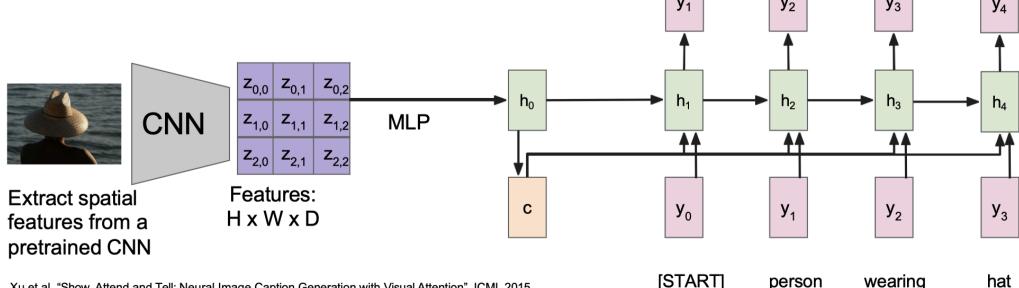


图 19.4: Image seq2seq

这里实际上没有使用 attention, 是经典的 seq2seq 模型, 但是可以使用 attention 来改进.

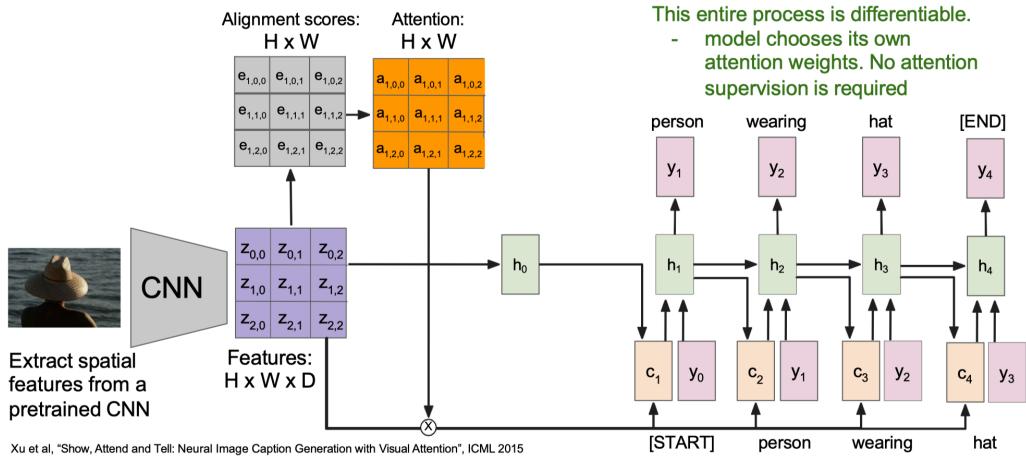


图 19.5: Image Attention Grad Flow

跟 NLP 中的 attention 很类似。

一个实现细节: 相似度使用内积还是余弦相似度? 取决于有没有对归一化

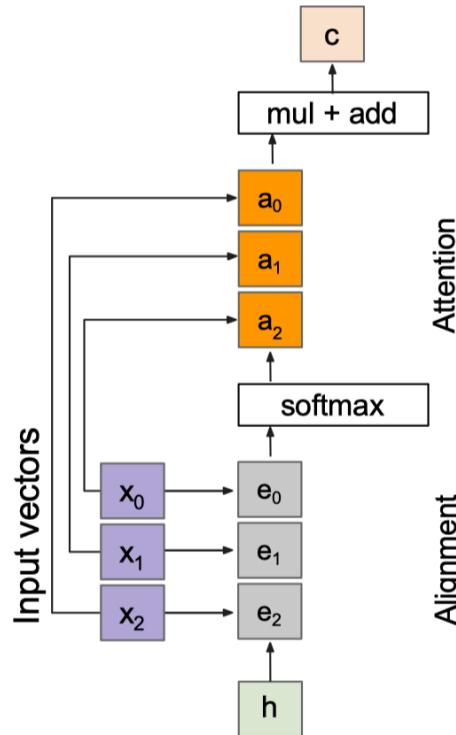


图 19.6: General Attention Layer

但是可以看到, h 依然是存在前后依赖的, 无法并行计算, 所以考虑把 h 也修改为与 c 类似的自适应的, 这就产生了 Self attention layer.

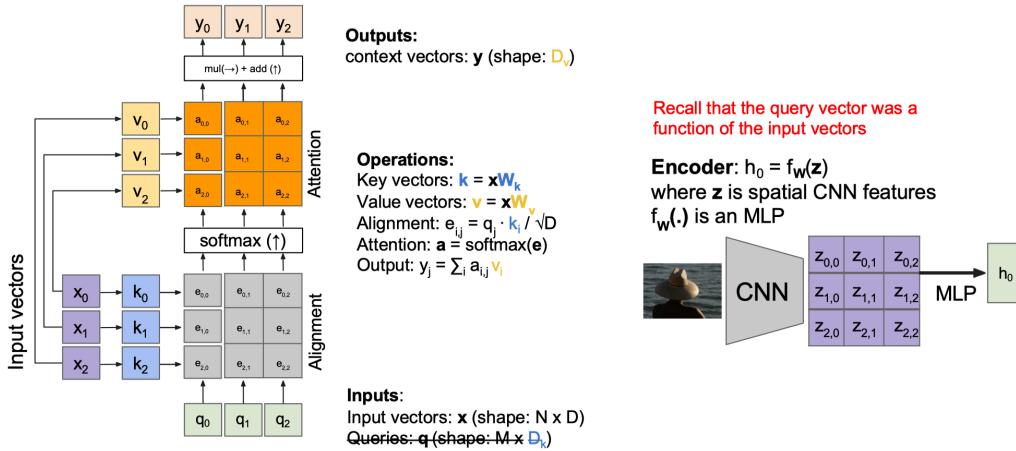


图 19.7: Self attention layer

根据相似度的计算方法, 可以发现 Self attention 是 Permutation equivariant 的, 就像 PointNet 一样.
但是这样就无法区别不同位置的 Token 了, 所以需要 Positional encoding

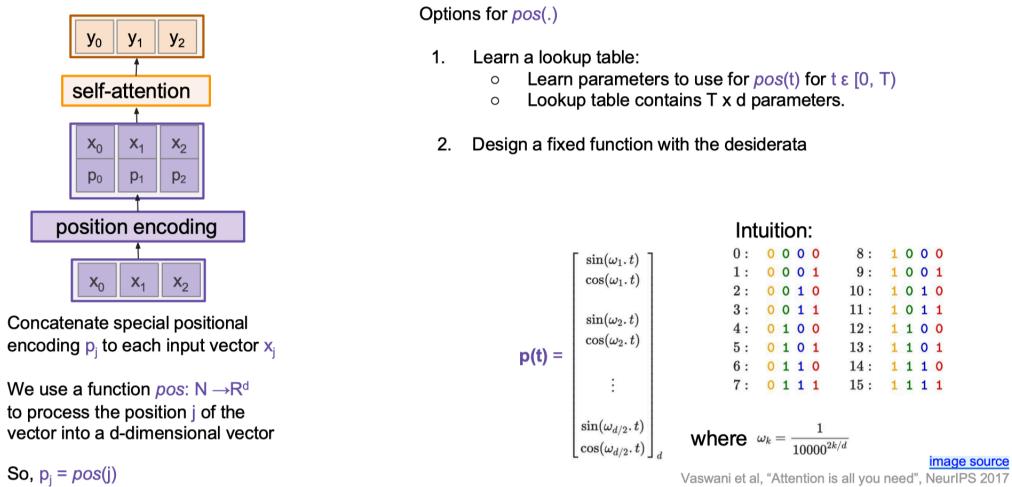


图 19.8: Positional encoding

不要小看 Positional encoding, 在处理大模型长序列的时候, Positional encoding 是核心技术点.

同时还不能让前面的 Query 对后面的 Key 进行查询, 所以需要 mask.

为了提高多语义的能力, 可以加入 multi-head attention, 可以理解为不同的 head 可以学到同一个 token 的不同意思, 在图像中可以理解为几千维的 feature 代表了一个像素区域, 现在需要把这些像素区域分开计算 attention, 会比一起计算有更好的表达能力.

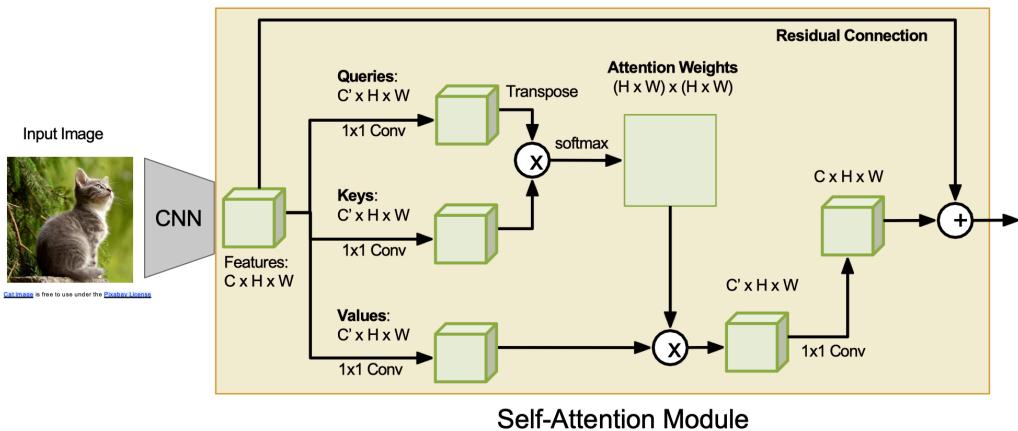


图 19.9: CNN with Self-Attention

一个例子: 图像上的自注意力机制

19.2 Transformer

编码器 (Encoder)

并行处理: 编码器在训练和推理时都是并行处理的. 编码器的每一层都可以同时处理输入序列中的所有位置, 因为它们不依赖于序列的时间步.

Mask 操作: 编码器中常用的 mask 是填充 mask(**Padding Mask**). 填充 mask 用于忽略输入序列中的填充标记(padding tokens), 以避免它们对结果的影响.

解码器 (Decoder)

并行处理 (训练时): 在训练时, 解码器也可以并行处理所有的时间步. 然而, 为了确保模型只看到当前时间步之前的输出, 需要使用前向遮掩 (**Look-Ahead Mask**).

串行处理 (推理时): 在推理时, 解码器需要逐步生成序列的每个时间步的输出, 因此是串行处理的. 这是因为在推理过程中, 解码器只能基于已经生成的部分序列来预测下一个标记.

Mask 操作

1. 填充 mask(Padding Mask): 与编码器相同, 用于忽略填充标记.
2. 前向遮掩 (Look-Ahead Mask): 确保解码器在预测当前时间步时, 不能看到未来的时间步.

19.3 Self-Supervised Learning (Pre-training)

SimCLR design choices: large batch size 原因: 不希望在一些例子上 over fit.

MAE, CLIP

Other examples: skip.

19.4 Large Multi-modal Models

BLIP

BLIP-2: Image Encoder 用 CLIP based 会好一些

InstructBLIP

Frozen: 发现 language model 不用 train, 只用 train vision Encoder

Flamingo

CLIP 的问题: 只能提取语义信息, 细节都丢失了.

交互是通往 AGI 的真正途径, 看了很多骑自行车的视频不代表我真的会了骑自行车, embodied AI, 具身多模态大模型

第 20 章 Object Detection and Instance Segmentation

20.1 任务简介

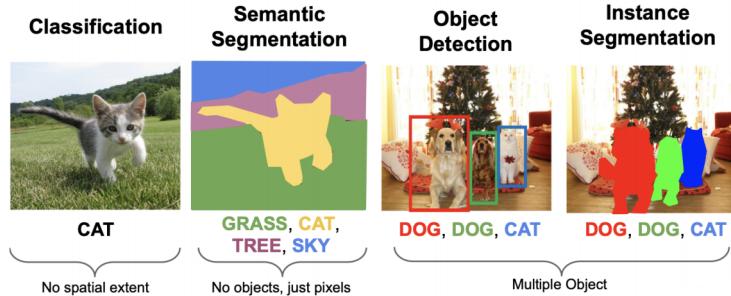


图 20.1: 分类任务

如上的任务当中, 最左侧的是分类, 即一张图片里只有一个待分类的事物.

随后是语义分割, 即将代表不同语义的像素区分开来, 每个像素都要有一个输出, 它不区分不同的个体. 随后是目标检测, 它区分不同个体, 但并不一定每个像素都被一个 bounding box 包围.

最后在此之上可以继续进行语义的分割.

20.2 Object Detection: Single Object

它的目标是定位和分类, 网络输出是 2D 的 bounding box, 且是 axis aligned 的.¹

确定 bounding box 需要四个参数 x, y, w, h , 即 bbox 左上角的坐标和大小.²

总的来说, 单目标检测可以被划分成两个任务:(对图片的) 分类和(对 bbox) 的回归. 如下图:

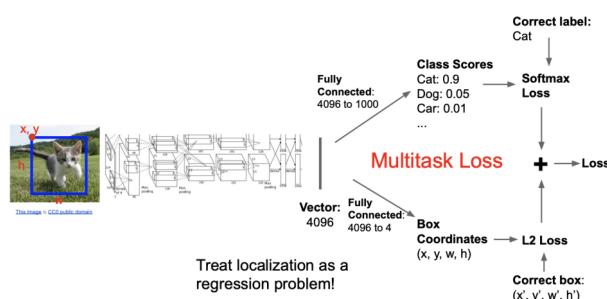


图 20.2: 单目标检测的网络概念图

假设我们得到了图片的特征向量, 那么可以通过 MLP 变成分类概率的向量, 然后使用交叉熵进行度量; 然后我们对图片 MLP 到四个数值, 然后用 L^2 loss 进行回归

注意 L^2 norm 和 L^2 loss 的区别:

L^2 norm 是模长, L^2 loss 是平方和. 当 L^2 norm 作为损失函数时, 被称为 rooted mean squared error(RMSE), 而另一个则是 mean squared error.

¹到了三维的时候, 由于阶数的提升, axis-aligned bounding box 中有非常多的部分并不属于这个物体, 因此可能需要旋转. 但是对 2D 来说, 这并不成为问题.

²如果我们用八个点的坐标作为表示, 那么显然有很多冗余. 尤其在高维的情形, 如何找到冗余尽可能少而能够便利地表达某一对象的方法是非常重要的. 在后面的旋转部分, 我们还会看到这一点.

RMSE 因为开根号的问题, 在接近 0 时可能出现问题, 很少用.

MSE 在接近收敛的时候, 梯度也小, 不容易越过, 同时 MSE 在较大的时候, 梯度很大, 容易导致神经网络训练效果不好, 而且还容易出现 NaN.

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

Smooth L1 Loss: 结合两者, 在较小的时候二次, 在较大的时候一次, 同时加入常系数以保证一阶和二阶连续. 这是一个 Multitask Loss, 即分类 loss 和 bounding box 的 loss 之和.

这两个 loss 应该以何种比例叠加? 比如分类的 loss 最大大概在 $\log N$ 的量级, 但是对 bounding box 来说, 如果采用 L^2 , 差 5 个 pixel 那就是 25. 而在这个损失函数中 $\text{Loss} = \alpha \cdot \text{Softmax Loss} + \beta \cdot \exp(\text{L2 Loss})$, e^{25} 是一个非常大的数字. 所以即使 loss 的量级差不多, gradient 的量级也不一定是相同的.

除此之外, 这个网络还有一个特殊性质: 对于不同图片, 可能需要不同数量的输出. 这在以前的 Neural Network 是无法做到的.

一个方法是用 sliding window 进行遍历检测, 但这也自然而然带来一个问题: 不知道 window 多大, 并且计算量极大

后来的传统方法采用了 Region Proposal 的方法, 即先提出一些可能是物体的 bbox, 在其中进行检测. 后来将这些 proposal 称为 Region of Interest (RoI).

20.3 Region-based CNN

第一个深度学习的相关工作 [RCNN]: R-CNN (Region-based CNN).

它大致流程是: 先提出一些 proposal, 统一变换到 224×224 的大小, 然后通过 SVM 进行分类, 以及对 bbox 的回归, 回归的输出是 (dx, dy, dh, dw) , 即 bbox 应该进行的调整.

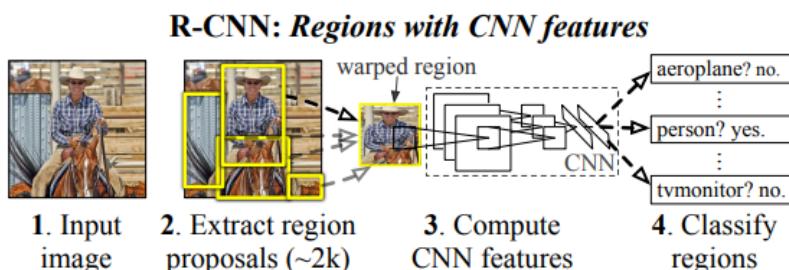


图 20.3: RCNN 分类部分

训练时我们会有很多的 RoI, 但 ground truth (后文简写成 gt) 的数量显然要少得多. 那么如何确定一个 RoI 被分成哪一类, 以及向哪个 gt 进行回归呢?

首先, 如果一个 bbox 与任何一个 gt 的交都小于某一阈值, 那么就将它单独分类成 background, 此时我们不再关心其 regression 的情况; 如果一个 bbox 同时包含了多个 gt, 那么可以计算它与这些 gt 的 IoU, 将 IoU 最大的那个 gt 作为其分类和回归的对象.

这里要注意的是, 我们不关注 background 的回归是必须采取的行为, 因为如果要监督一个 background 的回归, 那么它的 loss 是非常大的, 会直接破坏其他 RoI 的训练.

做 research 一定要着眼于已有工作的不足, 尤其是可以一眼看出来的问题.

这个工作的两个问题:

- 首先 proposal 可能太多了, 在测试时速度太慢.
- 其次若给出的 RoI 比物体小, 将其 Crop 之后的部分无法获知周围的信息, 几乎不可能准确对 bbox 进行回归.

20.4 Fast R-CNN

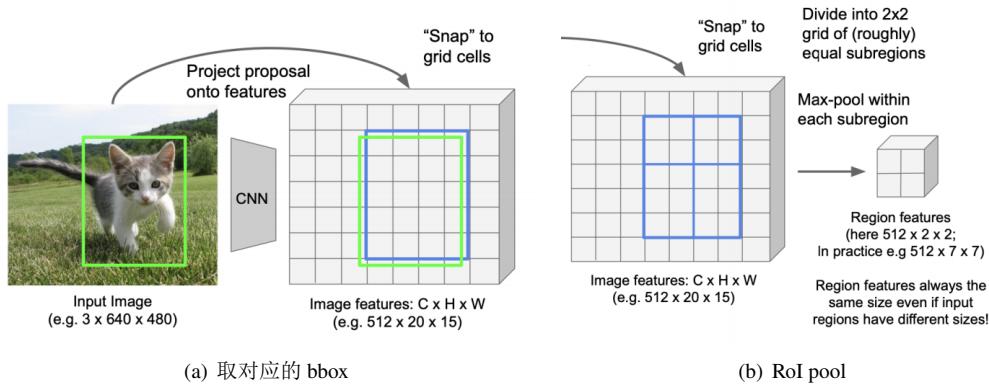


图 20.4: Fast RCNN

随后提出的 Fast R-CNN 解决了上面的两个问题.

Fast RCNN 仍然采用传统方法获得 ROI, 但是把 Corp 的时机放到了 CNN 之后.

将整个图片过 CNN, 再在 feature map 上取出对应的 bbox. 图像在卷积的过程中分辨率会减小, 因此在 feature map 上取的时候也要对应地缩小 ROI, 如图 20.4(a)所示, 缩小过程中不在格点上的顶点被吸附到最近邻的格点上.

这样取得的 ROI 大小仍然不统一, 原论文采用了一种 max pooling 的方法, 将形式各异的 ROI 分割成合乎要求的子块, 对每个子块求最大值, 如图 20.4(b)所示. 图中为方便将最终的大小画成了 2×2 , 实际上为 7×7 .

这样做的好处是

- 经过 conv 后, 感受野也增大了, 也就不会产生上文提到的被裁剪而缺少上下文的问题了
- 因为最后特征图上的的 ROI 分辨率比较小, 可以将多出来的 ROI 数量这一维度作为 Batch 的一部分, 从而可以完全并行化. 使用 32bits 浮点数计算, 以每一个 ROI 对应的 $7 \times 7 \times 512$ 的特征图为例, 2000 个 ROI, 只需要 191MB 显存, 但是 RCNN 中 2000 个 ROI, $224 \times 224 \times 3$ 的原始图片, 需要 1148MB 的显存.

Fast RCNN 相比 RCNN 取得了可观的速度提升.

下图中 20.5 左图为训练的时间, Fast RCNN 的训练时间只有 RCNN 的约十分之一. 右图测试速度显著提高, 蓝色为总时间, 红色为不包含传统方法获得 ROI, 仅对 ROI 进行处理的时间, 可以看出 Fast RCNN 时间的限制在 ROI 的提出上, 不像 RCNN 的瓶颈在于卷积网络的计算上.

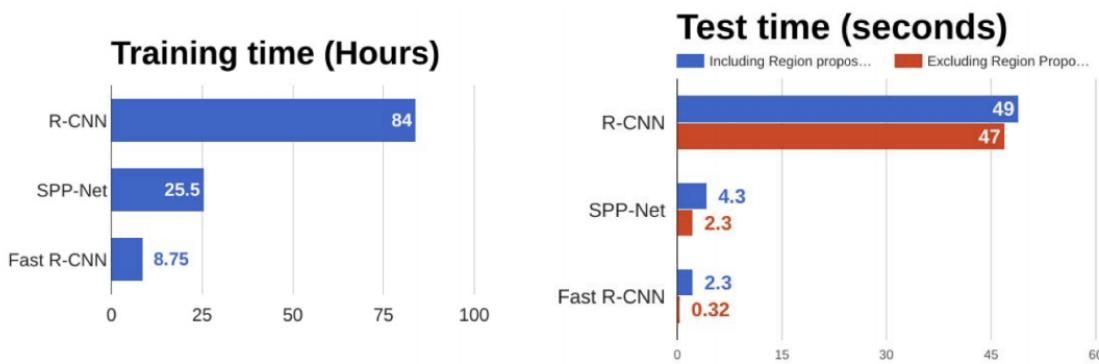


图 20.5: RCNN v.s. Fast RCNN

20.5 Faster R-CNN

经过 R-CNN 和 Fast R-CNN 的积淀, Ross B. Girshick 在 2016 年提出了新的 Faster RCNN.

在结构上,Faster RCNN 已经将特征抽取 (feature extraction),proposal 提取都整合在了一个网络中,使得综合性能有较大提高,在检测速度方面尤为明显.

但是依然不是端到端的网络,还是一个两阶段的网络,因为其中的 proposal 提取仍然需要首先训练.

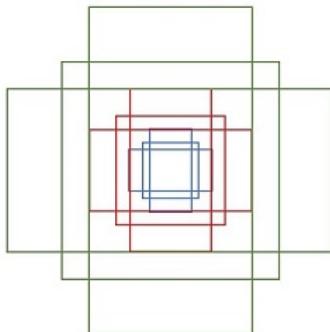


图 20.6: 不同形状的 anchor

首先引入 Anchor box 的概念, anchor box 是一些预先定义好大小的 bounding box, 数量为 K 个.

如何提取 proposal? 在 Fast R-CNN 中, 我们是通过 Region Proposal Network(RPN) 来提取的.

在 Fast R-CNN 中, 我们已经得到了 feature map, 而其中的每个 pixel 都可能作为不同形状的一个或多个物体的中心, 因此我们可以让每个 pixel 都给出自己是否是一些不同形状的 anchor 的中心(如图 20.6所示)

对于 $20 \times 15 \times 512$ 的一张 feature map, 得到 $20 \times 15 \times K$ 个概率值, 这些 300K 个概率中最高概率的 300 个 RoI 将被作为最终输送给下一步骤的 proposal. 所以第一步的 batch size 是图片数量, 第二步的 batch size 是图片数量 $\times 300$.

当然, 这样会获得数千个乃至更多的 bbox, 而图片里一般并没有这么多, 在第二步完成之后还要进行优化.

- 对于每个 proposal, 如果它在所有类上最高的概率小于某个阈值, 则直接舍弃.
- NMS. 先将这些 bbox 进行分类预测, 按照它们的分类进行分组, 随后对每个类型的分组内取分类概率最高的 RoI, 将同组之内和它 IoU 大于某个 threshold 的 RoI 全部去除. 这样做的原理是将某一类别概率最高的作为标准, 与其 IoU 较大的则认为是圈出了同一个物体, 全部去除; 对于剩余的 RoI 重复这一操作.³

当然,Faster R-CNN 之中还有非常多的细节, 比如 anchor 如何取定,

如何确定训练 RPN 的 positive/negative samples, 如何参数化 bbox 回归的过程等, 限于篇幅限制无法一一展开, 读者可参考 [**FasterRCNN**].

Faster R-CNN 中包含了四种 loss:RPN 对是否是物体的二分 loss, RPN regress box loss, final classification score, final box coordinates, 调参的过程自然是非常复杂的, 但是网络的效果也是显而易见的(如图 20.7). Faster R-CNN 的出现, 使得目标检测不再是学术界的 toy model, 而是真正进入了实用领域, 促进了安保等行业的发展.

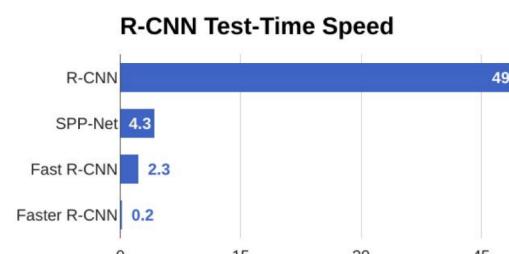


图 20.7: 几种网络的速度比较

20.6 two-stage detector and one-stage detector

如上图,Faster R-CNN 是一个分两步的目标检测网络. 有人提出: 既然在第一步里也做了 bbox 的 refine, 那么能否去掉第二步呢? 这就诞生了 single-stage detectors, 代表有 YOLO 系列, 它的特点就是非常快, 目前能达到 120 fps. 总体来说,two stage 准确率占优, 而 one stage 速度更快.

³举例来说, 假设猫分类的 RoI 中概率最大者圈住了一只猫的绝大多数, 因而以 90% 的 confidence 认为是猫, 则其他与它 IoU 大于 0.5 的 RoI 很可能只圈住了这只猫的半边身子, 所以把它们都去掉. 而 IoU 较小的可能是其他的猫. 当然, 这样做也存在一些问题, 比如有一个和它非常接近的 RoI 因为某种原因识别为狗, 那么这样做就不能去除这样的 RoI, 因为此处 NMS 只对同类的 RoI 进行操作. 后来也有工作同时预测 RoI 与 IoU(即“预测的预测”), 并证明这样做效果更优.

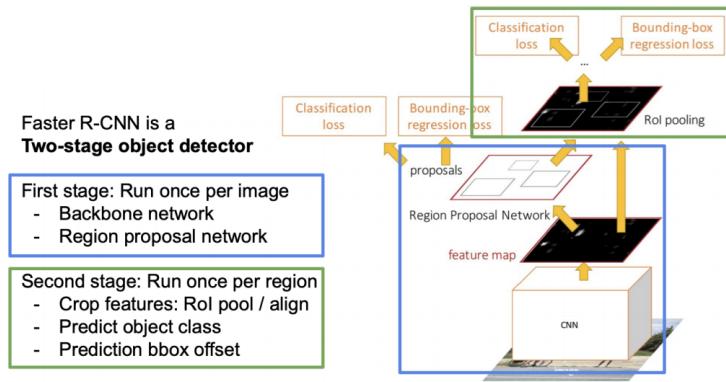


图 20.8: Two Stage 示意

20.7 Evaluation Metrics

无论是一步还是两步, 都需要面临的问题: 我们如何评估预测结果?

AP (Average Precision): Precision-Recall curve 下的面积

假如我们有一张图, 有 20 个 ground truth bounding box, 我们显然不可能要求其完全相符. 我们可以定义一个 IoU threshold. 首先其输出的类别要对, 其次 $\text{IoU} > \text{threshold}$. 另外一方面, 如果我乱猜了 5000 张, 显然也是不行的.

Trade-off between recall & precision.

画 PR 曲线: 先选出某个种类, 逐渐降低置信度阈值, 这样对于这一类的 precision 下降, recall 提升, 画在一个横轴是 recall, 纵轴是 precision 的二维坐标系上, 就画出了 PR 曲线.

AP: Average Precision, 即 PR 曲线下的积分面积.

AP at Different IoU Threshold: 对于不同的 IoU 阈值, 计算 AP.

mAP: 对于不同的 IoU 阈值 OR/AND 类别, 计算 AP, 然后取平均. 只对类别取平均的 mAP 可以记作 mAP@IoU, 比如 mAP@0.5 就是以 0.5 为 IoU 阈值的 mAP. mAP 对不同的 IoU 阈值取平均可以记作 mAP@[IoU1:IoU2], 例如 mAP@[0.5:0.95], 这种做法可以更全面地评估模型在不同 IoU 阈值下的性能, 尤其是在一些更严格的 IoU 阈值下.

Object detection 变量非常多. 若要准确度, 则 Faster R-CNN. 若要快速: YOLO, 但目前这一领域, 工业界已经占据了统治地位.

20.8 YOLO

现在已经出到 YOLO9 了, 但是这里只说最初的 YOLOv1

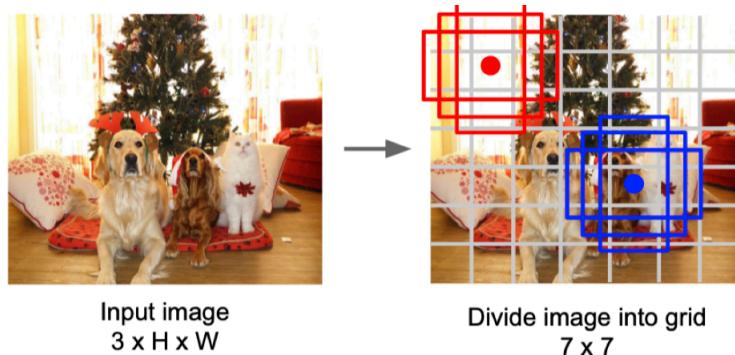


图 20.9: YOLO

将图片分为 $S \times S$ 个格子 (一般 S 为 7), 每个格子负责预测她为中心的 B 个 base boxes 和 C 类.

所以输出为 $S \times S \times (B \times 5 + C)$, 其中 5 个值分别为: 中心坐标, 宽高, confidence, 最后再加上 C 个类别的概率.

20.9 End-to-End Object Detection with Transformers (DETR)

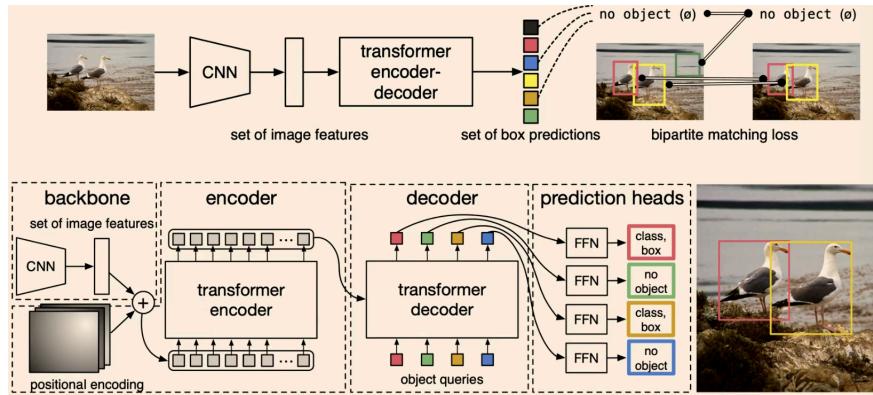


图 20.10: DETR

20.10 考什么

1. Transformer 和 Attention
2. NMS 的过程, YOLO 的 metrics

20.11 Mask R-CNN

在目标 bounding box 检测上再进一步, 输出哪个 pixel 属于哪个 segmentation. 直接从 box 里出 binary segmentation, 令人感觉 trivial.

两种方法: bottom-up, top-down. 目前在 2D 中前者较好, 因为 bbox 已经做得很好. 后者在 3D 中有用.

Top-Down Approach: Mask R-CNN

何恺明是如何让这个看起来大家都看不起的工作拿到了 ICCV 2017 best paper 呢? 他在这篇文章中提出了 RoI Align, Fast RCNN 中从原始图片到 feature map 中的吸附稍微丢失了信息, 所以边缘会很模糊.

首先, 经过 RoI pooling 之后, 分辨率可能下降. 但这是大家都知道的.

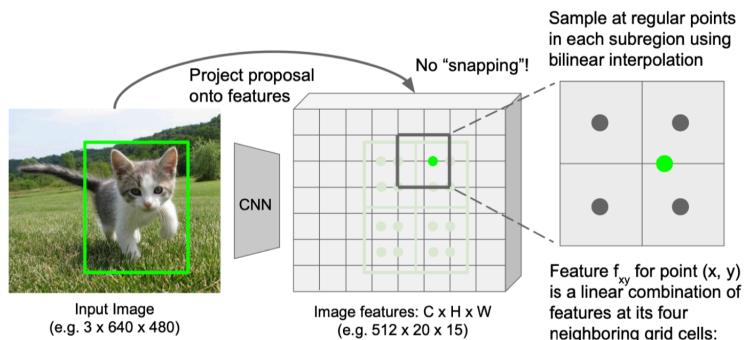


图 20.11: ROI align

最重要的在于, 何恺明指出, 我们不能进行最近邻的吸附, 否则会不匹配, 这是不可能被学习到的. 因此何恺明使用双线性插值进行处理, ROI align.

Ablation Study on RoI: AP75 提升比 AP50 高, 这是因为这样的方法对于高精度影响更大. 此外, 加入 align 后, bounding box 的表现也提升了, 这是因为 RoI align 后的特征更加精确.

Class-Specific vs. Class-Agnostic Masks

类别特定掩码是指为每个类别预测一个单独的 $m \times m$ 掩码. 也就是说, 对于每个候选区域 (RoI), 模型会为每个可能的类别生成一个掩码.

类别无关掩码是指预测一个单一的 $m \times m$ 掩码, 不管类别是什么. 然后, 通过分类分支来确定该掩码属于哪个类别.

Class-Specific 掩码更好, 可以通过例子理解, 猫的 mask 有猫的 prior, 人的 mask 有人的 prior, 因此通过预测所有类别的 mask 可以帮助前面的网络学习 bounding box 和分类.

Multinomial vs. Independent Masks

在多项式掩码中, 每个像素点的类别是通过一个多项式分布来预测的. 这通常通过一个 softmax 层来实现, softmax 层会输出每个像素点属于每个类别的概率, 最终的类别是概率最大的那个类别. 类别之间的竞争可能导致某些类别被忽略, 特别是在类别不平衡的情况下.

在独立掩码方法中, 每个类别都有一个独立的掩码, 这些掩码通过每个像素点的 sigmoid 函数来预测. 每个像素点可以同时属于多个类别, 这种方法没有类别之间的竞争.

20.12 3D Object Detection and Instance Segmentaiton

期末考试不考

什么模态都有 detection, 这里主要讲讲点云 detection, 对大模型之前的端到端自动驾驶很重要

三维 axis align 的 bounding box 自由度为 6, orientated 的为 9. rotated bounding boxes 一般指只有 y 轴旋转的
三维一般需要考虑方向.

从 2D 到 3D 往往难度高很多.

第 21 章 Generative Models

学习 $p_{model}(x)$ 来估计真实的数据分布 $p_{data}(x)$. 换言之假设 $\text{image} \in \mathcal{X} = \mathcal{R}^{3 \times H \times W} \sim p_{data}$. 我们想要习得这个分布.

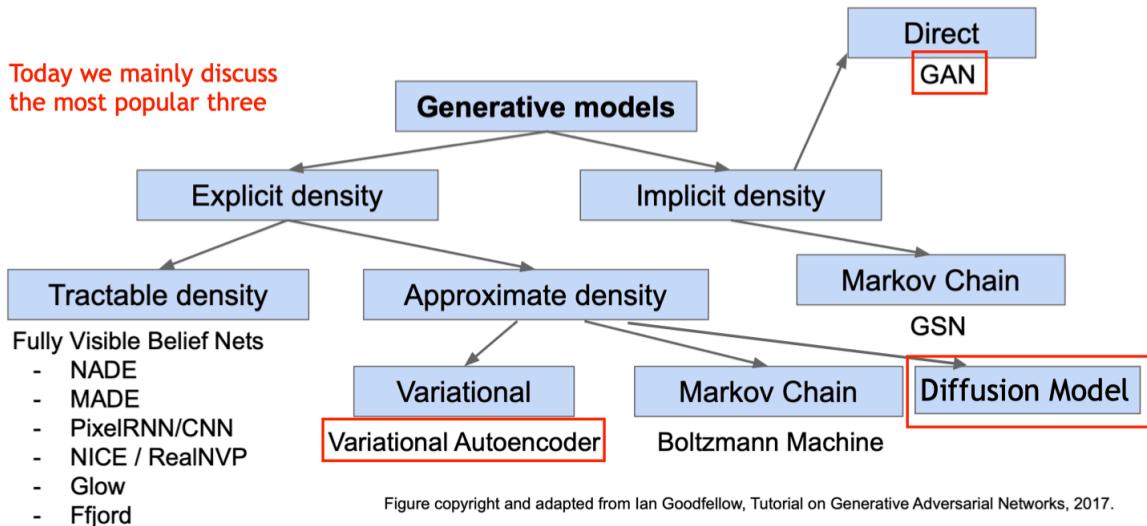


图 21.1: 生成模型分类

如图21.1所示, 整体来讲有两种生成模型: 隐式和显式. 显式模型是 explicit density model, or Fully Visible Belief Network (FVBN), 可以告诉你 $p_{model}(x)$; 而隐式模型则只能采样, 不能告诉你概率分布.

本门课接触三个生成模型: VAE, GAN, Diffusion Model

但是图中 Tractable Density 模型最容易理解, 先说一下

21.1 Tractable Density Model

假设我们的隐式概率模型是 $p(x) = p(x_1, \dots, x_n)$, 应用链式法则可得:

$$p(x) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1}) \quad (21.1)$$

什么样的神经网络能够处理这样的连续的条件概率?(显然我们希望获得一个 shared 网络)

PixelRNN, PixelCNN

优点: loss 直观, 容易优化, sample 方便

缺点: 由于存在序列关系, 所以慢, 并且序列关系对于图片是没有意义的

21.2 VAE

看过很多生成模型的博客, 这个博客是写的最好的: [From Autoencoder to Beta-VAE](#)

$$p_{\theta}(x) = \int p(z)p_{\theta}(x | z)dz \quad (21.2)$$

前面我们曾经讲过 AutoEncoder 的概念, 其实它可以视为一种降维手段, 将原数据通过一定处理(如多层神经网络)获得其另一种表示(或称编码), 这个过程就是 encode, 然后需要时可以进行解码恢复.

它本质上可以视为学习两个映射:

$$\begin{aligned}\phi : \mathcal{X} &\rightarrow \mathcal{F} \\ \psi : \mathcal{F} &\rightarrow \mathcal{X} \\ \phi, \psi = \arg \min_{\phi, \psi} &\| \mathcal{X} - (\psi \circ \phi) \mathcal{X} \|^2\end{aligned}\tag{21.3}$$

看起来在前面的 AE 当中, 我们只需要 decoder 部分就可以完成生成. 实果真如此吗? 如果只有 decoder, 那么 \mathbf{z} 服从何种分布完全不了解 (总不能是均匀分布吧).

然后考虑这个模型的 Loss:

$$p_\theta(x) = \int p(z)p_\theta(x|z)dz\tag{21.4}$$

如何计算积分? 积分里面的两项都容易算, 但是其积分不容易计算.

那么, Monte Carlo 方法可以吗:

$$p_\theta(x) = \mathbf{E}_{z \sim p(z)}[p_\theta(x|z)]\tag{21.5}$$

$$\log p(x) \approx \log \frac{1}{k} \sum_{i=1}^k p_\theta(x | z^{(i)}) , \text{ where } z^{(i)} \sim p(z)\tag{21.6}$$

如果使用蒙特卡洛方法估计, 维数太高, 方差太高, 需要 sample 太多了.

直接计算不行, Monte Carlo 不行, 那么试试 Bayes 公式, 如果能够引入后验概率 $p_\theta(z|x)$:

$$p_\theta(x) = \frac{p_\theta(x, z)}{p_\theta(z|x)} = \frac{1}{p_\theta(z|x)} p(z)p_\theta(x|z)\tag{21.7}$$

但是 $p_\theta(z|x)$ 还没有, 我们希望学一个神经网络近似 $p_\theta(z|x)$, 也就是学习一个 $q_\phi(z|x)$ 来近似 $p_\theta(z|x)$.

变分自编码器的 Variational 就是指这个近似, 使得 $q_\phi(z|x)$ 来近似 $p_\theta(z|x)$.

注意对于 VAE 的基本假设有如下三条

1. $p(z)$ 已知, 是一个高维标准正态分布
2. $p_\theta(x|z)$ 函数族已知, 一个均值和方差来源于神经网络的正态分布
3. $q_\phi(z|x)$ 函数族已知, 一个均值和方差来源于神经网络的正态分布

当然 $p(z)$ 是高维标准正态分布这个假设不是很合理, 这也是 VAE 的一个 limitation, 但是在一定程度上是可以接受的

再次 Formulate 一下这个工作:

假设所有已知数据 \mathbf{x} 来自一个未知的概率分布 $P(\mathbf{x})$, 我们希望用一组参数 θ 来确定一个参数分布 $p_\theta(\mathbf{x})$ 来拟合 $P(\mathbf{x})$. 我们假定 \mathbf{x} 与另一些隐变量 \mathbf{z} 有关, 那么依据边缘分布和条件分布的相关性质可知

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int_{\mathbf{z}} p_\theta(\mathbf{x} | \mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z}\tag{21.8}$$

这里 $p(\mathbf{z})$ 一般被称为先验分布, 一般取其为标准正态分布. 而 $p_\theta(\mathbf{z} | \mathbf{x})$ 则被称为后验分布. 编码器学习的就是 $p_\theta(\mathbf{z} | \mathbf{x})$, 因为它代表了如何从 \mathbf{x} 转换到隐变量. 解码器学习的则是 $p_\theta(\mathbf{x} | \mathbf{z})$. 先验的, 所以

上式并不能解决我们的问题. 尽管我们确定了先验分布, 而且 $p_\theta(\mathbf{x} | \mathbf{z})$ 可由解码器学习, 但上式的积分是难以计算的, 因为实际操作中我们不可能遍历 \mathbf{z} 所在的高维空间. 所以我们试图用贝叶斯公式曲线救国:

$$p_\theta(\mathbf{x}) = \frac{p_\theta(\mathbf{x} | \mathbf{z}) p(\mathbf{z})}{p_\theta(\mathbf{z} | \mathbf{x})} \tag{21.9}$$

这可能就是
经网络之...
凡是不容易
不好表示的
西, 通通丢
神经网络去
习...

那么 $p_\theta(z | x)$ 又怎么获得呢? 这是编码器负责的分布, 编码器就像一条辅助线, 这个分布不知道就引入一条辅助线来拟合这个分布, 因此我们用编码器来拟合它. 我们记编码器学习的分布为 $q_\phi(z | x)$.

那么如何度量学习的好坏呢? 两个分布的差异可以用 KL-divergence 度量

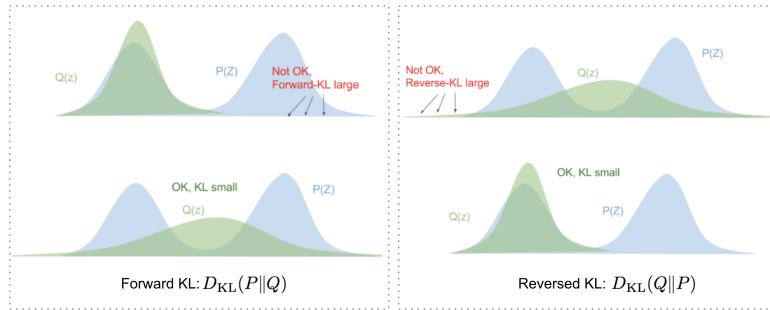


图 21.2: Forward and reversed KL divergence have different demands on how to match two distributions. Credit to [A Beginner's Guide to Variational Methods: Mean-Field Approximation](#)

最后概括一下 VAE 的工作, 然后进入形式化的推导: 在 VAE 当中, 输入数据 \mathbf{X} 是来自于一个特定的先验参数分布 $p_\theta(\mathbf{x})$, 随后同时训练 encoder 和 decoder, 使得在我们学习的后验分布 $q_\phi(z | \mathbf{x})$ 和真实后验分布 $p_\theta(z | \mathbf{x})$ 的 KL 散度 $D_{KL}(q_\phi \| p_\theta)$ 作为度量之下的误差变小.

当然这只是为了限制辅助线 q 别太歪, 让我们进行数学推导看看能不能变换一下这个式子, 毕竟我们的神经网络是 $p_\theta(\mathbf{x} | z)$, 算不出来 $p_\theta(z | \mathbf{x})$.

接下来推导 Loss:

计算学习的后验分布 $q_\phi(z | \mathbf{x})$ 和真实后验分布 $p_\theta(z | \mathbf{x})$ 的 KL 散度 $D_{KL}(q_\phi \| p_\theta)$ 得到:

$$\begin{aligned} D_{KL}(q_\phi(z | \mathbf{x}) \| p_\theta(z | \mathbf{x})) &= \int q_\phi(z | \mathbf{x}) \log \frac{q_\phi(z | \mathbf{x})}{p_\theta(z | \mathbf{x})} dz \\ &= \int q_\phi(z | \mathbf{x}) \log \frac{q_\phi(z | \mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(z, \mathbf{x})} dz \\ &= \int q_\phi(z | \mathbf{x}) \left(\log(p_\theta(\mathbf{x})) + \log \frac{q_\phi(z | \mathbf{x})}{p_\theta(z, \mathbf{x})} \right) dz \\ &= \log(p_\theta(\mathbf{x})) + \int q_\phi(z | \mathbf{x}) \log \frac{q_\phi(z | \mathbf{x})}{p_\theta(z, \mathbf{x})} dz \\ &= \log(p_\theta(\mathbf{x})) + \int q_\phi(z | \mathbf{x}) \log \frac{q_\phi(z | \mathbf{x})}{p_\theta(\mathbf{x} | dz) p_\theta(z)} dz \\ &= \log(p_\theta(\mathbf{x})) + E_{z \sim q_\phi(z | \mathbf{x})} \left(\log \frac{q_\phi(z | \mathbf{x})}{p_\theta(z)} - \log(p_\theta(\mathbf{x} | z)) \right) \\ &= \log(p_\theta(\mathbf{x})) + D_{KL}(q_\phi(z | \mathbf{x}) \| p_\theta(z)) - \mathbb{E}_{z \sim q_\phi(z | \mathbf{x})} (\log(p_\theta(\mathbf{x} | z))) \end{aligned} \quad (21.10)$$

将上式重写成

$$\log(p_\theta(\mathbf{x})) - D_{KL}(q_\phi(z | \mathbf{x}) \| p_\theta(z | \mathbf{x})) = -D_{KL}(q_\phi(z | \mathbf{x}) \| p_\theta(z)) + \mathbb{E}_{z \sim q_\phi(z | \mathbf{x})} (\log(p_\theta(\mathbf{x} | z))) \quad (21.11)$$

左侧第一项是我们希望最大化的输出的概率, 第二项是希望最小化的分布差异, 综合起来应该最大化左侧. 我们再来看右侧, 第一项是 $q_\phi(z | \mathbf{x})$ 和 $p(z)$ 之间的 KL 散度, 最小化这一项说明我们希望后验分布也符合正态. 最后一项最大化则是希望我们的解码器预测更加准确. 使用优化理论的常用手段, 我们将损失函数定义为

$$\mathcal{L}_{\theta, \phi} = -RHS = D_{KL}(q_\phi(z | \mathbf{x}) \| p_\theta(z)) - \mathbb{E}_{z \sim q_\phi(z | \mathbf{x})} (\log(p_\theta(\mathbf{x} | z))) \quad (21.12)$$

我们希望求得

$$\theta^*, \phi^* = \arg \min_{\theta, \phi} \mathcal{L}_{\theta, \phi} \quad (21.13)$$

我们的网络结构如图21.3所示, 它由两部分组成.

Variational Autoencoder (VAE)

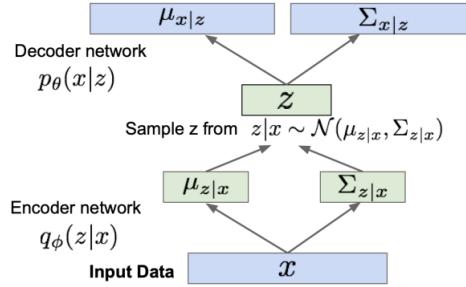


图 21.3: VAE 结构

现在我们来看看这两项具体如何运算.

第一项, 由于我们假定后验分布也是高斯分布, 两个高斯分布的 KL 散度有 Closed form solution

因为我们假
虽然我们添加这一项希望其接近高斯, 但这一项绝不可真正为 0, 否则就与 x 无关了, 这样就不含 x 的信息了, 后验分布
从而不可能进行重建.

$$D_{KL}(p(z|x)\|q(z)) = \frac{1}{2} \sum_{k=1}^d \left(\mu_{(k)}^2(x) + \sigma_{(k)}^2(x) - \ln \sigma_{(k)}^2(x) - 1 \right) \quad (21.14)$$

对于后验分布, 我们有

$$q(x|z) = \frac{1}{\prod_{k=1}^D \sqrt{2\pi\tilde{\sigma}_{(k)}^2(z)}} \exp\left(-\frac{1}{2} \left\| \frac{x - \tilde{\mu}(z)}{\tilde{\sigma}(z)} \right\|^2\right) \quad (21.15)$$

得到

$$-\ln q(x|z) = \frac{1}{2} \left\| \frac{x - \tilde{\mu}(z)}{\tilde{\sigma}(z)} \right\|^2 + \frac{D}{2} \ln 2\pi + \frac{1}{2} \sum_{k=1}^D \ln \tilde{\sigma}_{(k)}^2(z) \quad (21.16)$$

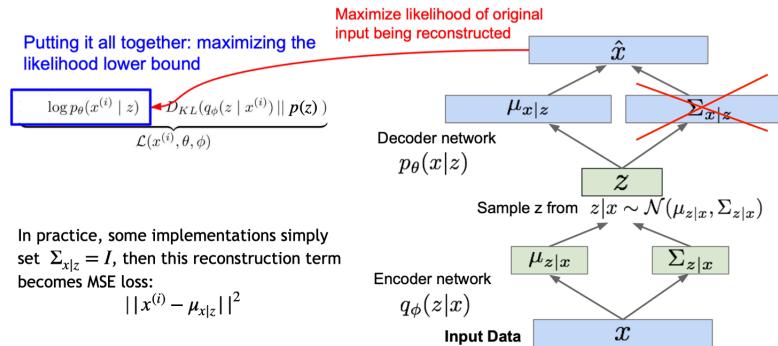


图 21.4: 简化的 VAE 结构

第二项, 也就是让网络输出的分布的接近真实的分布 (注意上式的 x 并不是网络输出, 而是输入)

因此, VAE 的 loss 非常有趣, 它的 ϕ 看似只出现在第一项, 但实际上还出现在第二项的 z 里面, 因此我们将 z 写成 $z = \mu_{z|x} + \epsilon\sigma_{z|x}$, 从而有梯度可以回传 (这就是重参数化技巧). 另外, ELBO 的计算也是 intractable 的, 因为第一项的 \mathbb{E}_z 这一步就计算不出来, 我们直接扔掉期望, 取它自己作为 Monte Carlo 的估计.

这里和最开始说 Monte Carlo 的地方相比较, 之前被估计的量是 $\log(\mathbb{E}_z [p_\theta(x^{(i)}|z)])$, 现在是 $\mathbb{E}_z \log p_\theta(x^{(i)}|z)$, 前者的 $z \sim p$, 后者则服从 q , 后者实际上更加集中, 方差更小.

实际上我们有时直接令 $\Sigma_{x|z} = \mathbf{I}$. 因为 loss 当中, 形式为

$$\exp^{-\left(\frac{x-\mu}{\sigma}\right)^2} \quad (21.17)$$

所以如图21.4所示,我们可以将 VAE 简化为一个简单的网络结构: 令 decoder 的 $\sigma = 1$, 即协方差矩阵为单位阵。否则对于上面这一项, 网络可以通过一直增大方差的方式来减小 loss, 如果 $\sigma = 1$, 损失函数就没有了分母, 变为 MSE。

为什么使用高斯分布?

1. 高斯分布简单, 可以通过均值和方差完全描述
2. 高斯分布的 KL 散度有解析解, 满足线性性, 可以进行缩放方便地进行重参数化
3. 高斯分布的采样很容易

这东西跟变分 (variational) 究竟有什么关系?

要说起变分, 先讲泛函. 泛函简单地说就是函数的函数, 或广义的函数.

变分, 是指自变量函数发生的变化, 为与自变量的变化 dx 区分, 我们一般用 δ 表示. 例如我们想要求解空间中 A, B 两点之间最短的曲线, 设任意一条曲线为 $f(x, y)$, 其长度为

$$l = \int_A^B f(x, y) ds. \quad (21.18)$$

当自变量函数 f 发生微小变化 δf 时, 长度也发生了 δl 的变化. 运用 EL 方程, 我们可以求得 $\frac{\delta l}{\delta f} = 0$ 时的函数 f .

在 VAE 当中, 我们实际上也是希望找到 q_θ 来近似 p_θ , 这是在说两个函数之间的距离很小; 这就类似于变分的时候找一个函数去逼近最速降线, 使得降落时间不变, 也是在考虑两个函数之间的距离.

但是实际上我们并没有使用任何变分法的技术, 因为变分法是不知道连续性之外任何函数的先验知识的, 但是这里对于神经网络我们已经加入了神经网络进行参数化, 从而将搜索空间限制为神经网络的参数空间上, 而非更高阶无限维的函数空间.

现在提到变分 (Variational), 都是在说这个工作引入了 ELBO 进行近似.

21.3 GAN

有了 GAN, 不代表 VAE 没用了, 在不要求清晰度的情况下, VAE 还是有很大发挥空间的.

$$\min_{\theta_g} \max_{\theta_d} [\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z)))] \quad (21.19)$$

交替训练, loss function 如下所示, 其中 D 是判别器, G 是生成器.

判别器梯度上升:

$$\max_{\theta_d} [\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z)))] \quad (21.20)$$

生成器梯度下降:

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z))) \quad (21.21)$$

但是有一点值得注意: 训练之初, discriminator 可以轻易辨别出 generator 生成的图片为假, 上式中 $\log(1 - x)$ 这一函数在 x 接近 0 时梯度又非常小, 导致 generator 的梯度又太小, 所以刚开始优化的时候生成器梯度很小, 优化效果不好.

所以这个 loss 对于 generator 是很不利的, 改变一下 loss 的形式, 可以进行凹凸性反转, 对生成器的目标函数进行改进, 使得生成器的目标函数在刚开始优化的时候梯度更大, 优化效果更好.

得到了 Non-Saturating Loss functions.

生成器梯度上升:

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z))) \quad (21.22)$$

为什么每一个 iteration 多次训练 Discriminator 但是只训练一次 Generator?

因为训练 Generator 的时候梯度要流过 Discriminator, 如果 Discriminator 训练的次数太少, Discriminator 找到的错误就不对, 因此 Generator 的梯度也不对, 会导致训练不稳定.

所以一个好老师很重要.

GAN 在单一分布上效果很好, 但是在大规模数据集很差, 为什么?

Mode chasing → Mode collapse: 生成器一直选择和输入一样的图片输出, 辨别器一直说这是假的, 最后导致生成器生成的图片都是同样的, 这就是模式崩塌.

VAE 要求所有类型都为真, 而 GAN 则只需要保证“我生成的”图片真.

这个问题最后也没解决.

Metrics

如何衡量网络的表现? 很难有客观的度量标准, 但是可以通过一些方法来评估生成器的性能.

1. 最近邻: 检测过拟合, 生成的样本和训练集中其最近邻进行对比
2. 用户研究: 参与者被要求在短时间内(例如 100 毫秒)区分生成的样本和真实图像或者根据生成图像的逼真度对模型进行排名
3. 模式丢失和模式崩溃: 对于具有已知模式的数据集通过测量生成数据到模式中心的距离来计算模式大小, 这也是我们之前 interpolation 要求渐变的原因, 防止你只学了个记忆功能. 如果是这样, 那么连续变化会有非常明显的不连续

有没有定量的 GAN Generator Evaluation? FID, a Quantitative Measurement, 实际上就是将 gt 和生成图片都转换到一个特征空间, 随后认为其符合正态, 度量其统计参数.

$$FID(r, g) = \|\mu_r - \mu_g\|_2^2 + \text{Tr} \left(\Sigma_r + \Sigma_g - 2 (\Sigma_r \Sigma_g)^{\frac{1}{2}} \right) \quad (21.23)$$

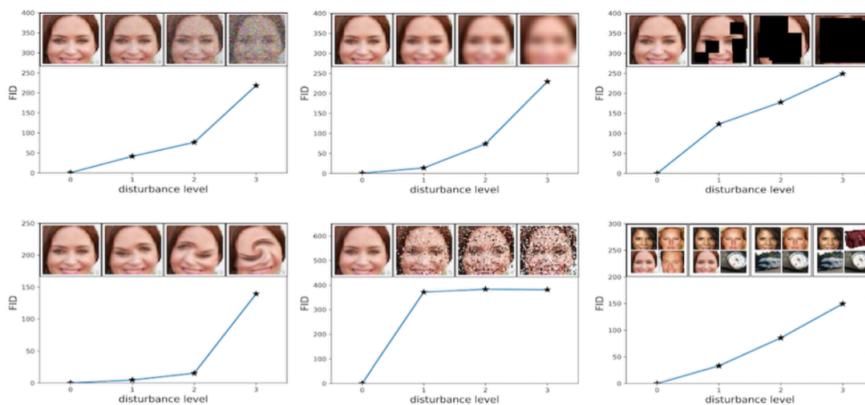


图 21.5: image distortions 对 FID 的影响

如图 21.5 所示, FID measure 对图像失真敏感. 从左上到右下: 高斯噪声、高斯模糊、植入的黑色矩形、旋转图像、salt and pepper 噪声, 以及被 ImageNet 图像污染的 CelebA 数据集.

GAN 在语义上的加减: 生成好的图片的网络, 其隐变量也必然学习了一些好的特征. latent space structure 有一定的线性代数结构.

21.4 Diffusion Model

不会考很深.

本质就是解决 VAE 的 latent space 是固定的标准正态分布的问题.

向大家推荐这个博客:[What are Diffusion Models?](#).

第 22 章 Pose and Motion

这一章节在 2024 年教学中已经被删去, 为了让有兴趣的读者了解, 故保留

22.1 Beyond Detection: Pose

Pose 是物体刚性运动的表征. 二维 bbox 拥有四个自由度. 三维有六个. 若有转动角 θ , 则为七个.
rotation 包含物体的朝向信息. 定义六维物体姿态 (6d object pose) 为 3 平动 (translation), 3 转动 (rotation).
旋转矩阵 \mathbf{R} 满足 $\mathbf{R}^\top \mathbf{R} = \mathbf{I}$ 且 $\det \mathbf{R} = 1$. 它属于群 $\text{SO}(3)$ ¹.
旋转矩阵只有三个自由度, 但却包含九个元素, 这使得神经网络难以预测. 我们需要其他的表达方式.

22.2 Euler Angle

$$R_x(\alpha) := \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix} \quad R_y(\beta) := \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix} \quad R_z(\gamma) := \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (22.1)$$

当然, 具体使用的时候, 有不同的 convention: 比如, 按照什么顺序依次旋转轴? 旋转是使用变换前的轴, 还是变换后的轴?

但两个操作复合时, 并不能简单将两个角相加.

22.3 Axis Angle

寻找瞬时转轴和角度 e, θ . 得到 θe , 三个变量可以取任意值.

应用 Rodrigues' rotation formula, 可以将 axis-angle 转变为 rotation matrix:

$$R = \mathbf{I} + (\sin \theta) \mathbf{K} + (1 - \cos \theta) \mathbf{K}^2 \quad (22.2)$$

其中 $\mathbf{K} = [e]_\times$.

Axis Angle 非常好地表示了旋转的特征, 且其 θ 是不随坐标系选取而变化的. 但是它仍然存在问题: 当我们获知两个 AA 的 θ, \mathbf{K} 时, 其复合也不能由它们简单运算得到. $\text{SO}(3)$ 在李群, 李代数当中有很漂亮的形式, 以及与 AA 的联系.

22.4 Quaternion

Quaternion 即四元数, 表达形式是由一个实部和三个虚部组成:

$$q = w + xi + yj + zk. \quad (22.3)$$

¹SO 的含义是 Special Orthogonal, 前者代表行列式为 1.

其中

$$\begin{aligned}
 \mathbf{i} * \mathbf{i} &= -1 \\
 \mathbf{j} * \mathbf{j} &= -1 \\
 \mathbf{k} * \mathbf{k} &= -1 \\
 \mathbf{i} * \mathbf{j} &= -\mathbf{j} * \mathbf{i} = \mathbf{k} \\
 \mathbf{j} * \mathbf{k} &= -\mathbf{k} * \mathbf{j} = \mathbf{i} \\
 \mathbf{k} * \mathbf{i} &= -\mathbf{i} * \mathbf{k} = \mathbf{j}
 \end{aligned} \tag{22.4}$$

运算律:

$$\begin{aligned}
 \mathbf{q}_1 * \mathbf{q}_2 &= (w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2) \\
 &\quad + (w_1 x_2 + x_1 w_2 + y_1 z_2 - z_1 y_2) \mathbf{i} \\
 &\quad + (w_1 y_2 - x_1 z_2 + y_1 w_2 + z_1 x_2) \mathbf{j} \\
 &\quad + (w_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 w_2) \mathbf{k}
 \end{aligned} \tag{22.5}$$

共轭: $q = w - i - j - k$. 性质为

$$\|\mathbf{q}\| = \sqrt{\mathbf{q} * \mathbf{q}'} = \sqrt{w^2 + x^2 + y^2 + z^2} \tag{22.6}$$

若模为 1, 则是单位四元数. $\mathbf{q}^{-1} = \mathbf{q}'$. 乘法满足结合律但不满足交换律.

如何表达旋转? scalar+vector 的表达方式:

$$\mathbf{q} = (s, \mathbf{v}) \tag{22.7}$$

得到

$$\mathbf{q}_1 * \mathbf{q}_2 = (s_1 s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, s_1 \mathbf{v}_2 + s_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2) \tag{22.8}$$

一个单位四元数对应一个旋转. $s = \cos \frac{\theta}{2}, \mathbf{v} = e \sin \frac{\theta}{2}$.

对于一个向量 \mathbf{x} , 其运算方式为将 \mathbf{x} 补成四元数 $\mathbf{x} = (0, \mathbf{x})$, 求 $\mathbf{x}' = \mathbf{q} \mathbf{x} \mathbf{q}^{-1}$. 计算复合只需要四元数相乘.

四元数实际上是一个四维空间上的超球面 S^3 . 很遗憾, 它也不是欧式的.

How to Estimate Rotation use Neural Network?

方法一: Use a neural network to regress a rotation representation.²

方法二: Predict object coordinate or correspondence and then solve rotation.

Orthogonal Procrustes Problem:

$$\widehat{\mathbf{A}} = \underset{\mathbf{A} \in \mathbb{R}^{p \times p}}{\operatorname{argmin}} \|\mathbf{M} - \mathbf{N} \mathbf{A}\|_F^2 \quad \text{subject to} \quad \mathbf{A}^T \mathbf{A} = \mathbf{I}, \tag{22.9}$$

The solution can be expressed in terms of the SVD of a special matrix³.

$$\mathbf{M}^T \mathbf{N} = \mathbf{U} \mathbf{D} \mathbf{V}^T, \text{ then } \widehat{\mathbf{A}} = \mathbf{V} \mathbf{U}^T \tag{22.10}$$

SVD is very sensitive to outliers. For fitting rotations, we need to use RANSAC.

How many pairs of 3D-3D correspondence do we need for hypothesis generation? 2 pairs(如果连线不与转轴平行).

²在此祝愿助教 Jiayi Chen 的论文被顺利接收.

³这里为了保证行列式为 1, 可以令 $\widehat{\mathbf{A}} = \mathbf{U} \Lambda \mathbf{U}^T$, 其中 $\Lambda = \operatorname{diag} \{1, 1, \det \mathbf{V} \mathbf{U}^T\}$

第 23 章 Instance-Level 6D Object Pose Estimation

这一章节在 2024 年教学中已经被删去, 为了让有兴趣的读者了解, 故保留
Instance-level: a small set of known instances.

Pose is defined for each instance according to their CAD model.

Input: RGB/RGBD. 如果有相机内参, 那么没有 D 也可以. 有 D 可以做得更好.

2D center localization. 先预测 2d 图片的中心位置和深度. 随后利用相机内参得到 translation.

PoseCNN: Translation Estimation: Voting. 每个 pixel 给出一个指向中心的向量, 得到 center.

PoseCNN: Rotation Estimation. ROI?

loss: $\mathcal{L}(\mathbf{q}, \mathbf{q}^*)$. 我们发现 \mathbf{q} 和 $-\mathbf{q}$ 在旋转意义上是相同的, double coverage. 因此一种可行的 regression loss 是取两者的最小值.

PoseCNN 则采用了另一种 loss:

$$\text{PLoss}(\tilde{\mathbf{q}}, \mathbf{q}) = \frac{1}{2m} \sum_{\mathbf{x} \in \mathcal{M}} \|R(\tilde{\mathbf{q}})\mathbf{x} - R(\mathbf{q})\mathbf{x}\|^2 \quad (23.1)$$

对称性:(表示旋转的等价类)

$$\text{SLoss}(\tilde{\mathbf{q}}, \mathbf{q}) = \frac{1}{2m} \sum_{\mathbf{x}_1 \in \mathcal{M}} \min_{\mathbf{x}_2 \in \mathcal{M}} \|R(\tilde{\mathbf{q}})\mathbf{x}_1 - R(\mathbf{q})\mathbf{x}_2\|^2 \quad (23.2)$$

PoseCNN 的 translation 表现尚可, 但是 rotation 的表现一般, 这受限于四元数的性能.

6D pose 要求已知物体的 cad 模型, 这在现实中不太可能.

category-level 6D pose. 希望能够泛化, 输入 3d 输出 6d pose, Without the need to use CAD model.

王鹤老师的论文:Normalized Object Coordinate Space for Category-Level 6D Object Pose and Size Estimation, CVPR2014 与照片
oral.

Detecting and estimating 6D pose and 3D size of previously unseen objects from certain categories from RGBD images.

为什么要 depth 呢? 因为对于未知的物体来说, 仅有 rgb 而没有 depth 是无法确定其大小的. 有了 depth 和相
机内参, 才能消除 scale 的不确定性.

问题的主要难点是 rotation 的估计. 前面我们看到 PoseCNN 即使对于已知的物体, 做得也相当不好.

间接法.Representation: Normalized Object Coordinate Space(NOCS)

简而言之, 我们需要对一张图片的像素预测其在 CAD model 中的位置. 你可能会问: 不是没有 CAD model 吗?
在此我们建立了一个 reference space:NOCS.

step 1: rotation Normalization: align object orientations. 将所有物体对齐成同样的姿态, 如马克杯的方向都向左,
此时旋转矩阵为 0.

Step 2 (translation normalization): zero-center the objects. 对于新物体, 将其紧 bbox 的中心作为原点.

Step 3 (scale normalization): uniformly normalize the scales. 将 bbox 的对角线长度设置为 1. 这样所有的都可以
放入一个对角线长为 1 的正方体里了.NOCS = Reference frame.

NOCS = Reference frame transformation from NOCS to camera space.

23.1 Beyond Object Pose

human/hand pose estimation. 人体可以按照关节活动, 并不是刚体.

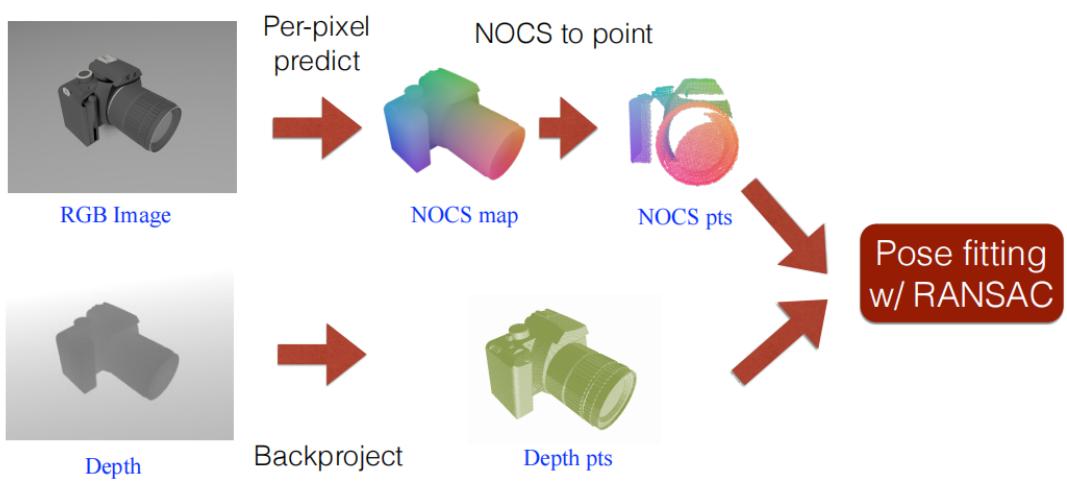


图 23.1: From Image to NOCS map to Pose.

第 24 章 Motion

这一章节在 2024 年教学中没有讲授, 为了让有兴趣的读者了解, 故保留

Today let's focus on motions between two consecutive frames!

Optical Flow 光流.

图片的亮的部分在两帧之间的表象运动.

几个假设: 亮度相对稳定, 小移动, 一个点的运动与其邻居相似.

$$\begin{aligned} I(x+u, y+v, t) &\approx I(x, y, t-1) + I_x \cdot u(x, y) + I_y \cdot v(x, y) + I_t \\ I(x+u, y+v, t) - I(x, y, t-1) &= I_x \cdot u(x, y) + I_y \cdot v(x, y) + I_t \\ \text{Hence, } I_x \cdot u + I_y \cdot v + I_t &\approx 0 \rightarrow \nabla I \cdot [uv]^T + I_t = 0 \end{aligned} \quad (24.1)$$

那么, 这个方程足够解出所有 (u, v) 吗? 我们有 n^2 个方程, 但有 $2n^2$ 个未知数, 因此不够.

The Aperture Problem. 单纯从图像来看, 运动可能并不完整. Barberpole Illusion. 沿着线的方向不容易观测, 垂直的容易被观察到.

更多约束: Spatial coherence constraint. 1981 年 Lucas 和 Kanade 提出了假设在每个 pixel 的 5*5window 当中 flow 相同.

$$\begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{bmatrix} \quad (24.2)$$

即 $\mathbf{A}_{25 \times 2} \mathbf{d}_{2 \times 1} = \mathbf{b}_{25 \times 1}$

得到

$$\mathbf{A}^\top \mathbf{A} \mathbf{d} = \mathbf{A}^\top \mathbf{b} \quad (24.3)$$

什么时候可解?

1. 可逆
2. 特征值不能太小
3. 良态

FlowNet: 最简单的想法: 两张三通道图 merge 在一起, 卷.dense regression.early fusion.

或者: 分别提取 feature. 两个网络 share weight. 然后结合到一起.middle fusion.

过早 fusion 会使得问题空间变大. 过完 fusion 会使得微观细节缺失.

这和我们之
的 Harris C
ner Detec
非常相似.
流当中最容
被捕获的也
corner.corne
与光流紧密
关.

第 25 章 Embodied AI

一路走来我们的工具是逐渐齐全的, 任务是逐渐丰富的. 但是这些任务都是基于被动的数据输入, 还没有接触过机器自主的与环境交互, 现在希望能具备在 physical world 进行交互的能力.

25.1 Embodied AI

具身智能是什么?

这个词语出现非常久了, 但是中国在 23 年左右才开始提. Turing 提到, 人工智能有两种和人竞争的方法, 第一种是使用搜索等方法, 解决类似下棋的问题; 第二种就是具身智能, 使用传感器, 涵盖了知识传递的过程.

经典智能, 只具有人类大脑的能力, 计算能力; 但是人在大脑之外还具有交互能力, 这就是 embodied AI.

具身智能相比基于数据的智能近两年才火起来, 也是因为具身智能不像网络上有那么多数据.

Embodiment 是什么?

Embodiment 就允许了你去探索环境, 和环境交互, 得到经验. 这个过程既给了你探索的能力, 也对你作出了限制, 车是不能飞的.

有人讲, Embodied AI 是通向 AGI 的必经之路, 道理就是如果一个 agent 没有 interaction 的能力, 就不能认为它是 AGI 的.

比如 Hand-Eye Coordination 任务, 人类也不是天生具有的, 需要后天学习, 甚至很多 Hand-Eye Coordination 任务现在也不会, 比如 Juggling ball.

你觉得大自然中有没有 AGI?

思考的智能可能不具备, 但是 embodied 智能是很强的.

从上述讨论中抽象出来一个高于 Perception-only 的 paradigm, 也就是 Perception-Action Loop¹.

人类的学习大部分就是 Perception-Action Loop, 只有少部分, 比如学习考试, 是 supervised learning. Perception-Action Loop 是我们现在模型中很欠缺的.

Scaling law 是不是通向 AGI 的唯一方法?

For sure answer is no.

Data driven 是有用的, 但是纯互联网数据是不够的, 现在的 ChatGPT-4o 等大模型基本上已经穷尽了互联网数据(但是还没有将算力使用枯竭), 每周再爬取几十 Terabytes 数据就是现在唯一的数据来源了, 或许还有一些侵犯版权的数据. 并且互联网上的数据只占每人每天的数据的一小部分, 相信你不会把自己的所有行为都放在互联网上.

但是现在还有大量的数据还没有利用, 就是 Perception-Action Loop 的数据, 这样的数据暂时还不存在很多, 比如自动驾驶数据可能有一些, 但是只能在特定环境下使用.

一般来讲我们相信 Perception-Action Loop 是下一个阶段的 AGI 的重要数据源泉.

Now and Future for Embodied AI

1. Industrial Robots and Autonomous Driving for now.

但是不好说 Industrial Robots 是一个 AI. 因为这些机器人是在一个非常特定的环境下工作的, 所有运动路径都是 hard code 的.

¹ 强化学习-like

另外一个就是 Autonomous Driving, 比如 Tesla 准备在 2024 年 8 月在美国进行 robotaxi 服务测试, 使用 FSD(Full Self-Driving) 技术, 他们宣称这是一个 end2end 的 imitation learning. 这就是一个典型的 embodied AI, 并且是 data-driven, 也符合 Perception-Action Loop 的 paradigm.

但是显然这样的数据 DOF 还是太低了, 控制还是比较简单, 但是需要安全度很高, 99% 或者 99.9% 也不算绝对安全, 至少需要像人一样开车上万公里才出现一次失误才足够.

对于 Tesla 这样的技术, 如果一个季度没创死人, 就算是 revolutionary.

2. Home robots for future.

制造一个像人一样的通用机器人. 但是这样的机器人需要具备很多能力, 比如抓取、操作、导航等等.

老师上课提到了转笔, 这可能是在说 2023 年的这篇论文 [ma2023eureka], 这篇论文采用 LLM driven 的 reward fine-tuning, 在物理仿真中实现了非常灵巧的转笔任务.

同时 locomotion 这也是一个非常难的任务, 这样的下肢任务是落后于上肢运动的, 容错率也很低.

老师认为 embodied AI 可以支持 AI 再发展 20 年.

25.2 Object Grasping

prehensile: 可理解的, 可抓取的.

不同的抓取稳定性

Non-force closure: 靠力保持稳定.

Force closure: 靠力保持稳定, 能容忍任意一个方向的加速度.

Form closure: 靠形状保持稳定, 任意一个方向的速度都不会让物体脱离.

怎么做呢?

主流方法是使用 detection 方法, 跟 RCNN 一样把所有抓取方法都检测一遍, 可以像 object detection 一样算 recall 和 precision.

但是需要数据集, 多少数据够呢? 很多, 所以需要 synthesize 很多数据.

这篇论文 [fang2020graspnet] 在 88 个物体上使用 Force closure 的 constraint 上 synthesize 了十亿个数据, 结果是可以泛化到所有物体上.

为什么使用点云? 只在意形状, 不在意颜色.

泛化性来源于什么? 点云中的局部相似性, 也就是说, 一个点云可以抓取的局部形状是可以泛化到其他物体上的.

Cons?

透明物体或者说不是漫反射的很难抓取.

可以通过 synthetic data 训练模型来解决这个问题.

一个发现

Synthesize 大量数据集同时进行 domain randomization, scale 很大之后, 就可以泛化到真实世界了.

曾经 Google 在真实世界使用 7 台机械臂 +RL 在 RGB 模态下训练 Grasping, 用一年达到了 87% 的准确率, 但是不能泛化到其他环境, 抓取台一旦改变就不 work 了. 问题就在于 RGB 模态对于环境是不具有泛化性的.

25.3 Object Manipulation

老师上课提到的 OpenAI 拧魔方是这篇工作 [openai2019solving], 试图在物理仿真中进行策略学习然后泛化到真实世界.

其中最重要的 idea 是使用 Automatic Domain Randomization (ADR), 让物理仿真中的 Domain Randomization 自动化改变, 而不是人类手动改变环境参数, 从而使得模型可以泛化到更加复杂的环境中.

作为一个魔方爱好者, 其实这样的结果距离真正的 One-handed Rubik's Cube 还有较大差距, 但是对于这样一个 Training from scratch 的工作, 可以实现这样的效果已经是非常了不起的了.

但是更重要的事情是只学这一个远远不够的, 这样的 policy 是不具有到其它任务上泛化性的.

Dexterous Hand

手的自由度是非常高的, 一只手可能就有 22 到 25 个自由度, 赋予了人极高的自由度, 但是学习难度也非常高.

Shadow Hand 100w 人民币. Dexterous hand 是达到人类灵巧度的希望.

Related work

Learning Category-Level Generalizable Object Manipulation Policy: 基于强化学习的抓取.

GAPartNet: 基于监督学习的抓取.

DexGraspNet: 大型抓取数据集.

25.4 Locomotion and Navigation

四个轮子已经解决, 四条腿比较难, 两条腿基本上还没解决.

传统的 locomotion 机器人不使用视觉, 但是现在希望结合视觉信息.

Navigation Tasks

Point Goal: Go 5m south, 3m west relative to start

Image Goal: Go where the photo was taken.

Object Goal: Go to the red chair.

Navigation Methods

Classical Modular Navigation: Mapping, Planning, Execution

End-to-end Reinforcement Learning: imitation learning (behavior cloning), RL

Related work

考试不要求相关论文

Online 3D Recon. and Navi. (CVPR 2023): RL

Mobile Manipulation (ICRA 2024): RL, loco-manipulation

25.5 Embodied Multimodal Large Model

真正的 AGI 的可能形态.

RT-1: 采集 13 万条数据, 以 97% 的成功率抓取一个厨房里面的物体, 做了一年半, 使用 imitation learning 学习的.

RT-2: 问题也在于泛化性不好, 数据也不够.

估计使用上万亿个小时的视频才可以做到真正的泛化 embodied AI.

老师认为使用合成数据是好的.

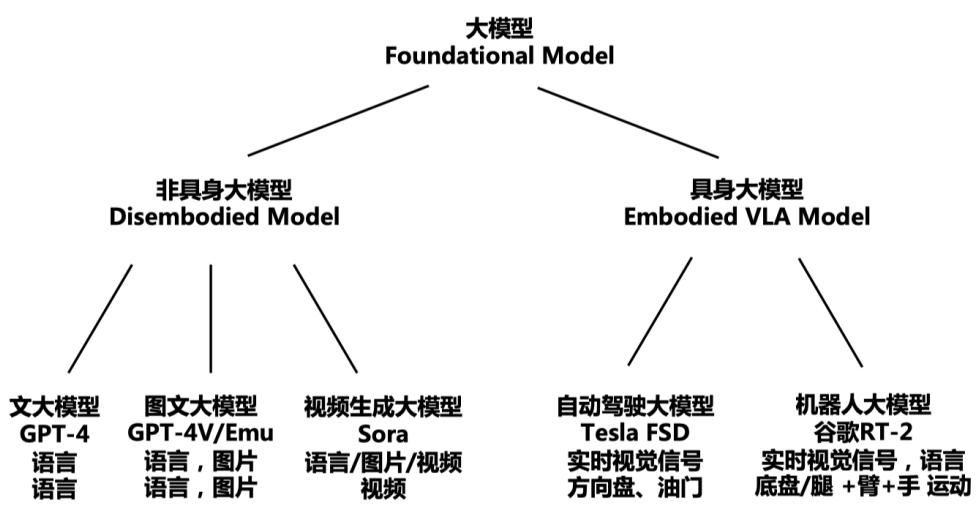


图 25.1: Multimodal Large Model

第 26 章 Summary of Computer Vision

Compared to human vision, computer vision deals with the following tasks:

1. visual data acquisition (similar to human eyes but comes with many more choices)
2. image processing and feature extraction (mostly low-level)
3. analyze local structures and then 3D reconstruct the original scene (from mid-level to high-level)
4. understanding (mostly high-level)
5. generation (beyond the scope of human vision system)
6. and further serving embodied agents to make decisions and take actions.

谢谢老师的 **wonderful lecture!**

谢谢看到这里的同学! 欢迎在 [github](#) 上 star 本项目或者访问我的[个人网站](#)上关于课程的测评!

附录 A Condition Number

问题的条件数是数值分析中常见的概念, 是导数的一种推广. 简单来说, 就是对于输入发生微小变化时, 输出的变化程度的大小. 如果一个问题的条件数较小, 那么就称其是良置的 (well conditioned), 否则称为病态的 (ill conditioned).

考虑一个线性系统 $\mathbf{A}\mathbf{x} = \mathbf{b}$, 那么若 $\det \mathbf{A} \neq 0$, 则 $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. 若输入变化 $\delta\mathbf{x}$, 则输出变化为 $\mathbf{A}\delta\mathbf{x}$. 考虑相对变化之比的上界:

附录 B Transformation in \mathbb{R}^n

在这一小节中我们简单介绍各种变换.

等距变换顾名思义, 就是保持距离的变换. 在其基本形式当中可以表示为平移加旋转, 这也是我们已经接触过的. 矩阵表示为

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (\text{B.1})$$

其中的 \mathbf{R} 为旋转矩阵, 也是正交矩阵.

相似变换是指形状不变, 但可以改变大小和位置的变换. 直观地说就是等距变换加上缩放. 矩阵表示为

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{S}\mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \mathbf{S} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \quad (\text{B.2})$$

其中的 \mathbf{S} 表示缩放. 它保持边的长度比例和角度不变. 理解仿射变换的一种有益方法是将线性变换 \mathbf{A} 视作旋转和非均匀缩放的组合. 这是因为我们可以对 \mathbf{A} 进行 SVD:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top \quad (\text{B.3})$$

由于 $\mathbf{U}, \mathbf{V}^\top$ 都是正交矩阵, 可以视作旋转, 而 $\Sigma = \text{diag}\{\sigma_1, \sigma_2\}$ 则可以视为非均匀缩放.

仿射变换则是一种保持了点, 线和平行性的变换. 它可以表示为一个线性变化加一次平移. 也就是

$$T(\mathbf{v}) = \mathbf{A}\mathbf{v} + \mathbf{t} \quad (\text{B.4})$$

同样在齐次坐标下我们可以写作

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (\text{B.5})$$

只不过这里的矩阵 \mathbf{A} 可以代表任意的线性变换.

射影变换则只保留了将线映射成线, 而不保证平行性. 它表示为

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{v} & b \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (\text{B.6})$$

不难看出它包含了上述所有的变换种类, 添加了额外的自由度 \mathbf{v} . 注意并不总能够通过缩放矩阵使得 $b = 1$, 因为 b 可能为 0. 当 $b \neq 0$ 时, 射影变换可以做如下分解:

$$\mathbf{H} = \mathbf{H}_S \mathbf{H}_A \mathbf{S}_P = \begin{bmatrix} s\mathbf{R} & t/v \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{K} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{v}^\top & b \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{v} & b \end{bmatrix} \quad (\text{B.7})$$

其中

$$\mathbf{A} = s\mathbf{RK} + \frac{1}{v}\mathbf{tv}^\top \quad (\text{B.8})$$

且 \mathbf{K} 是满足 $\det \mathbf{K} = 1$ 的归一化上三角矩阵. 如果限定 s 的符号, 它是唯一的, 只需做一次 QR 分解即可得到.

不难看出这个分解是将射影变换分解为相似变换 \mathbf{H}_S , 仿射变换 \mathbf{H}_A 和一个有约束的透视变换 (特殊的射影变换) \mathbf{H}_P 组成.

在射影变换下, 四个点的交比仍然保持不变. 四个点 P_1, P_2, P_3, P_4 的交比定义为

$$\text{cross ratio} = \frac{\|P_3 - P_1\| \|P_4 - P_2\|}{\|P_3 - P_2\| \|P_4 - P_1\|} \quad (\text{B.9})$$

附录 C DOF and rank in essential matrix and fundamental matrix

所谓矩阵的自由度, 实际就是指矩阵中有多少个元素可以独立变化. 例如, 一个 $m \times n$ 的矩阵在不加任何限制的情况下有 mn 个自由度, 而对于 n 阶上三角矩阵, 其自由度为 $n(n + 1)/2$.

从三维的物体投影成二维的图像, 这个过程其实就是射影变换. 在射影空间当中的单应矩阵 \mathbf{H} (可以理解为我们的投影变换矩阵) 天然少一个自由度, 因为自原点出发位于同一条线上的两个点, 在射影之后无法区分, 所以 $\mathbf{H} \sim \alpha\mathbf{H}$.

测地线距离:geodesic distance.

附录 D QR Decomposition

矩阵的 QR 分解就是将矩阵分解为正交矩阵和上三角矩阵的乘积, 它可以对任意形状的矩阵进行. 常用的方法有 Gram–Schmidt process, Givens rotations 和 Householder reflections 等. 我们用最容易理解的施密特正交化方法来推导方阵的情形.

我们先将分解后的形式写出:

$$\begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{bmatrix} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_n \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & r_{nn} \end{bmatrix} \quad (\text{D.1})$$

也就是满足

$$\mathbf{a}_i = \sum_{j=1}^i r_{ji} \mathbf{e}_j \quad (\text{D.2})$$

由此可以定出

$$\begin{aligned} \mathbf{u}_1 &= \mathbf{a}_1, & \mathbf{e}_1 &= \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|} \\ \mathbf{u}_2 &= \mathbf{a}_2 - \text{proj}_{\mathbf{u}_1} \mathbf{a}_2, & \mathbf{e}_2 &= \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|} \\ &\vdots \\ \mathbf{u}_n &= \mathbf{a}_n - \sum_{j=1}^{n-1} \text{proj}_{\mathbf{u}_j} \mathbf{a}_n, & \mathbf{e}_n &= \frac{\mathbf{u}_n}{\|\mathbf{u}_n\|} \end{aligned} \quad (\text{D.3})$$

以及

$$r_{ij} = \langle \mathbf{e}_i, \mathbf{a}_j \rangle \quad (\text{D.4})$$

我们考虑一个方阵 \mathbf{P} , 其副对角线上的元素为 1, 其余为 0. 不难验证 $\mathbf{P}\mathbf{P}^\top = \mathbf{I}$, $\mathbf{P} = \mathbf{P}^\top = \mathbf{P}^{-1}$. 左乘矩阵 \mathbf{P} 会使得矩阵上下翻转, 右乘会使得矩阵左右翻转. 将一个上三角矩阵上下翻转后左右翻转, 即变为下三角矩阵. 记 $\tilde{\mathbf{A}} = \mathbf{PA}$, 对 $\tilde{\mathbf{A}}^\top$ 进行 QR 分解, 得到

$$\tilde{\mathbf{A}}^\top = \tilde{\mathbf{Q}} \tilde{\mathbf{R}} \quad (\text{D.5})$$

由此得到

$$\mathbf{A} = \mathbf{P} \tilde{\mathbf{R}}^\top \tilde{\mathbf{Q}}^\top = (\mathbf{P} \tilde{\mathbf{R}}^\top \mathbf{P}) (\mathbf{P} \tilde{\mathbf{Q}}^\top) \stackrel{\text{def}}{=} \mathbf{R} \mathbf{Q} \quad (\text{D.6})$$

同样的, 对于 \mathbf{A}^\top 进行 QR 分解就可以得到 \mathbf{A} 的 LQ 分解, 可以用同样的方法得到 QL 分解. 不过在使用时要注意: \mathbf{P} 不一定是旋转矩阵.