Learning LOG

郭治焱
信息科学与工程学院, 湖南大学
长沙市, 岳麓区麓山南路
zhiyanguo@hnu.edu.cn www.gzyzq.site

最新版 - 2021 年 10 月 1 日

Abstract

这份文档用于记录郭治焱在研究生期间，学习、做研究的笔记、所感所悟等。文档正式开始于 2020 年暑假（研一开学前），但是由于当时做的还比较散乱，就不当作文档的起始时间了。文档正式纪录始于 **2020-09-18**。

近期逐步将一些部分的内容剥离了这个文档，将论文阅读笔记、想法分别放在了新的文档中。修改记录于 **2020-10-14** 凌晨。顺便感叹一下，想点子好难！

mybox-InfoColor mybox-ImportantColor mybox-NormalColor mybox-OtherColor
mybox-HintColor xmybox-InfoColor xmybox-ImportantColor
xmybox-NormalColor xmybox-OtherColor xmybox-HintColor commandbox
> commandbox*

关 键 词: GNN, Machine Learning, Knowledge Graph, Recommender System, Learning To Rank，学习记录, 科研记录

目录

1 Concepts about ML

1.1 Basics

1.1.1 Auto-regressive model—AR

自回归模型，“自”体现为：使用某个变量不同时期的值（如时间序列的值）来预测该变量下一个值。AR 从线性回归发展而来，假设该变量的不同步之间的值为线性关系。

1.1.2 Variational Auto Encoder-VAE

变分自编码器。

1.1.3 Spectral cluster

谱聚类。将每个样本视作某个空间中的点。

参考资料：

1. 谱聚类原理总结

1.1.4 Spectral graph convolution

1.1.5 Semi-supervised learning

1.1.6 t-SNE

t-distributed Stochastic Neighbor Embedding。一种数据降维的方法。

参考资料：

1. sklearn 中关于 tSNE 的介绍
2. tSNE 降维原理

1.1.7 L1/2 regularization

1.1.8 mini/full-batch gradient descent

1.1.9 预训练，使用自编码器进行预训练

1.1.10 机器学习中常见的数据分布

1.1.11 early stopping strategy

1.1.12 Jacobean Hessian

1.1.13 Automatic differentiation

1.1.14 不同的优化器原理

可参考：从 SGD 到 NadaMax，十种优化算法原理及实现

1.1.15 Teacher forcing

RNN 训练时使用的一种训练方法。在训练时，使用模型的输出值用来计算误差，但是使用真实的数据作为下一次的输入。

参考资料：

1. 自动微分简介

1.1.16 熵、相对熵、交叉熵、互信息

✓2020-10-08

这些概念来自信息论 [15]。简单来说，熵指的是不确定性或者信息量，熵越大不确定越大。相对熵也叫 KL(Kullback-Leibler divergence) 散度，用来比较两个概率分布之间的差异。不论是熵、相对熵、交叉熵，都可以看作针对某个（或多个）随机变量，对该随机变量的概率分布的一个某种熵（熵、相对熵、交叉熵）的计算。接下来就从数学上来对其进行描述。

熵：先介绍自信息的概念。对于某个随机变量 X，当 X 取值为 x_0 时的自信息为 $I(x_0) = -log\ p(x_0)$，即事件 x_0 发生时所带来的信息量，如果一个事件发生的概率越大，则其带来的信息量越小。熵是自信息的均值。即 X 取任一值时所能带来的期望信息量，故 X 的信息熵 $H(X) = -E_{x\sim p}I(x)$（其中 p 是 X 所服从的分布），即 $H(X) = -\sum_{x\in X} log\ p(x)$。**因为随机变量 X 会服从某个分布，假设是 p，则 $H(X)$ 也可以看作是概率分布 p 的信息量的期望**。

相对熵：也称作 KL 散度。KL 是在信息熵的基础上定义的，用来衡量两个分布的差异，其实由上述熵的含义可知，其实 KL 也可以看作是随机变量 X，其可能服从的两个分布之间的差异。假设可能服从的两个分布分别是 p, q，则以 q 去接近 p 时的 KL 散度表示为：$D_{KL}(p||q) = \sum_{x\in X} p(x)log\ \frac{p(x)}{q(x)} = E_{x\sim p}log\ \frac{p(x)}{q(x)}$。经过化简可得 $D_{KL} = -H(p) + \sum_{x\in X} p(x)log\ q(x)$。

交叉熵：cross-entropy，与 KL 散度相似，也是用来衡量随机变量 X 可能服从的两个分布 p, q 之间的差异的。其数学上的定义为：$Cross - Entropy(p, q) = -E_{x\sim p}log\ q(x) = -\sum_{x\in X} p(x)log\ q(x)$。很显然，交叉熵比 KL 散度多了一个 $H(p)$，即 $Cross - Entropy(p, q) = D_{kL}(p||q) + H(p)$。

机器学习中常使用交叉熵，既然 KL 散度和交叉熵都可以达到相同的目的，那为什么不使用 KL 散度呢？在机器学习中，上述的分布 q 常作为数据的真实分布，而 q 作为数据的预测分布，此时 $H(P)$ 则可以视作一个常数（因为给定数据后，其真实分布是确定的），在优化模型的参数时，$H(P)$ 并不会对参数的优化做出贡献（不会影响优化过程），故使用交叉熵即可。

1.1.17 线性回归 & 逻辑回归

✓2020-09-30

线性回归研究的问题是多个变量中某个变量和其他变量之间存在的线性关系，相当于用多个变量线性表示某个变量，某个变量就称为因变量，其他变量就称为自变量。用数学语言来描述的话就是这样的：

$$y = b + \sum_{i=1}^{n} w_i \cdot x_i$$

在 n 等于 1 时，相当于根据数据拟合一条直线；在 n 大于 1 的时候，就是多元线性回归了，此时拟合一个平面。自变量也可以称为特征。构建线性回归模型时，重要的有这几个点：发现相关性较高的特征，发现与因变量无关的特征，得到最后实际使用的 n 个特征，对于实值特征的范围进行约束，对于类别特征的类别进行处理。其实，以上这些主要是针对数据的初步分析和预处理。得到处理后的数据，接下来可以通过随机梯度下降的方法得到最优的权重。在线性回归中常使用的目标函数是均方误差函数。为了避免模型过拟合，目标函数中还可以加入正则化项，对特征权重进行限制。

其实线性回归模型很像神经络中的一部分————一个激活函数为恒等映射的神经元，也可以看做一个神经网络模型————一个只有一层的神经网络模型。

逻辑回归（Logistic regression, LR），也是在线性回归的基础上的一个分类模型。如果把线性回归看做神经网络单元，那么把激活函数替换为非线性的激活函数就是 LR 了。从数学形式来看，LR 是这样的：

$$z = b + \sum_{i=1}^{n} w_i \cdot x_i$$

$$\bar{y} = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z}$$

针对一个样本 x，LR 模型得到的就是 $\bar{y}$，很明显这是一个 0 到 1 之间的值，即一个概率值，当对样本进行类别划分后（划分为 0、1），$\bar{y}$ 是类别为 1 的概率。二分类 LR 的目标函数定义如下：

$$L(w_1, ..., w_n) = \prod_{i=1}^{m} (\bar{y^i})^{y^i} \cdot (1 - \bar{y^i})^{1-y^i}$$

其中 y^i 为样本 $\boldsymbol{x^i}$ 的实际类别，$y^i \in \{0, 1\}$。显然这是一个关于权重和偏置的最大似然函数，对其取对数后得到：

$$loss = logL = \sum_{i=1}^{m} \left(y^i log(\bar{y^i}) + (1 - y^i) log(1 - \bar{y^i}) \right)$$

很明显，线性回归和逻辑回归的目标函数是不一样的，那么为什么 LR 不适用均方误差损失函数呢？理论上 LR 也是可以使用均方误差损失函数的，但是均方误差损失在进行 SGD 时存在一个问题，当预测值与真实值相差越大时，参数变化的越小，训练的越慢（这个可以通过均方误差损失函数进行 SGD 时对参数的梯度可以看出）。上述的对数似然函数也可以叫做交叉熵。

对于多类别（例如类别数为 $\mathcal{L}$）的 LR，可以训练 $\mathcal{L}$ 个二分类的 LR，每个 LR 只输出样本属于某个类别的概率，最后进行集成得到最终的输出结果。此时的目标函数则会发生一点变化：

$$loss = \sum_{i=1}^{m} \sum_{l=1}^{\mathcal{L}} \mathbb{1}\{y^i = l\} log \frac{e^{\boldsymbol{w^l} \cdot \boldsymbol{x^i}}}{\sum_{j=1}^{\mathbb{L}} e^{\boldsymbol{w^j} \cdot \boldsymbol{x^i}}}$$

其中 $\boldsymbol{w^l}$ 是第 l 个 LR 模型的权重向量（包括了偏置）。

1.1.18 metric learning

度量学习。学习如何衡量两个对象之间的相似度/距离。

1.1.19 n-fold cross validation

即 n 折交叉验证。

1.1.20 xavier innitializatino

Xavier[8] 一种参数初始化方法。神经网络模型的参数初始化时很重要的，对模型最终的效果、收敛速度都有很大的影响。

1.1.21 批训练的过程，原因

在训练神经网络时，通常训练数据都是很大的，无法一次性加载进内存。可以把数据分为很多个 batch，使用每个 batch 训练模型后再进行参数调整。

1.1.22 Contrastive loss, Triplet loss

都是一种损失函数。

Triplet loss 用于训练差异较小的数据，常用于人脸识别中。以这种函数为损失函数时，输入的一个样本是一个三元组 (anchor, positive, negative)，anchor 是随机选择的一个样本，而 positive 和 negative 分别于 anchor 为同类/异类数据。在学习时，Triplet loss 的目的就是让 anchor 的表征与 positive 的表征尽量靠近，与 negative 的表征尽量疏远。那么对于一个样本来说，Triplet loss 写成公式：

$$\mathcal{L} = max(||f(a) - f(p)||_2^2 - ||f(a) - f(n)||_2^2 + \alpha)$$

公式的含义也很明显，尽量使类内数据相近，类间数据相离。

Contrastive loss 是对比损失，也叫 zero-one 损失，主要用来处理孪生网络中的 paired data 的关系。通常 Contrastive loss 的输入是两个样本，且各自都有一个标签，Contrastive loss 的目标就是：如果两个样本同类，则 loss 更小，否则 loss 更大。写成公式：

$$\mathcal{L} = d_{ij}^2 \cdot Y_{ij} + (1 - Y_{ij}) max(margin - d_{ij}, 0)^2$$

其中，d_{ij} 就表示样本 i, j 之间的距离，这个可以有多种形式的定义，Y_{ij} 表示样本 i, j 的标签是否相同，margin 是一个阈值，如果 d_{ij} 超过阈值则应该尽量把它们划分开。

1.1.23 Siamese Network

孪生网络。

1.1.24 Batch Normalization

在神经网络的训练过程中，前一层的输出相当于后一层的输入，当对参数进行更新的时候，前一层的输出会发生变化，可能前一层---后一层的输入的分布就发生了变化，这会降低模型的收敛速度或使得模型难以收敛。为了解决这个问题，BN 通过对每一层的输出进行标准化，使其分布尽量稳定下来，加快模型的收敛速度。实际情况在进行 BN 时，可能是在通过激活函数之间进行 BN 或者在通过激活函数后再进行 BN。通常是在激活函数之前进行 BN，因为当输入较大时，通常激活函数的变化都较小，梯度变化不明显，故在激活函数之间就对数据进行 BN，使其分布尽量稳定。

1.1.25 判别模型、生成模型

- 判别模型：直接对判别函数或者条件概率分布函数进行建模，不考虑样本的产生模型，直接研究预测模型
- 生成模型：学习联合概率密度 $P(Y,X)$，然后求出条件概率分布 $P(Y|X)=\frac{P(X,Y)}{P(X)}$，不仅要求出联合分布，还要求出训练数据的分布 $P(X)$（不一定要计算 $p(X)$，因为对于同一个样本，计算它属于不同分类时，其 $p(X)$ 是一样的，对判别没有帮助）。生成模型表示了输入 X 产生输出 Y 的生成关系

对于生成式模型，可以这样理解：
$p(x|y)=p(y)p(x|y)$ 表示的是，从 $p(y)$ 中采样一个 y，然后根据 $p(x|y)$ 采样一个 x。生成式模型希望找到那个能够使 $p(x,y)$ 最大的 y。1

生成模型从统计的角度表示数据的分布情况。判别模型不能反映训练数据本身的特性，但它不断寻找不同类别之间的最优分类面。

也可以从 **决策函数 $Y=f(X)$ 或条件概率分布 $P(Y|X)$** 的角度来看待判别模型和生成模型：从数据中学习一个分类器时，希望通过给定的输入 X 输出相应的 Y。这个模型的一般形式为：

- 决策函数 $Y=f(X)$：输入一个 X 就输出一个 Y，可以将 Y 于阈值比较得到 X 的类别
- 条件概率分布 $P(Y|X)$：输入一个 X，输出 X 属于各个类的概率，如 $P(c_1|X),P(c_2|X)$。取其中最大的作为 X 的类别

实际上 $P(Y|X)$ 是隐含了或者说可以转化为决策函数形式 $Y=f(X)$ 的。例如，将条件概率分布改写为 $Y=\frac{P(c_1|X)}{P(c_2|X)}$。

参考资料：判别模型 (Discriminative model) 和生成模型 (Generative model)、Background: What is a Generative Model?。

1.1.26 变分贝叶斯

用来近似计算复杂积分，在这类模型中一般包含三类变量：观测变量、未知参数、隐变量，其中位置参数和隐变量统称为不可观测变量。变分贝叶斯的目的主要有两个：

- 近似估计不可观测变量的后验概率，以便通过这些变量做出推断
- 对于一个特定的模型，给出观测变量边缘似然函数的下界，作为模型选择的依据。一般认为似然概率越高，模型效果越好

通常情况下，我们会有一组观测数据（D），那么怎么获得不可观测变量（Z））的后验概率 P(Z | D) 呢?

通常不可观测变量的后验概率是很复杂的，难以直接计算之，但我们可以先假设一个分布 Q(Z) 与 P(Z|D) 是近似的。对于衡量两个分布的差异，可以使用 KL 散度，即：KL(Q(Z) | P(Z|D)) 。

$$
\begin{aligned}
KL(Q(Z)||P(Z|D)) &= \sum_{z\in Z} Q(z)\log\frac{Q(z)}{P(z|D)} \\
&= \sum_{z\in Z} Q(z)\log\frac{Q(z)P(D))}{P(z,D)} \\
&= \sum_{z\in Z} Q(z)(\log\frac{Q(z)}{P(z|D)} + \log P(D)) \\
&= \log P(D) + (\sum_{z\in Z} Q(z)\log\frac{Q(z)}{P(z,D)})
\end{aligned}
$$

显然只要最小化 KL 散度即可，因为 $\log P(D)$ 是一个常数，所以只要最小化 $\sum_{z\in Z} Q(z)\log\frac{Q(z)}{P(z,D)})$ 即可。将 $\sum_{z\in Z} Q(z)\log\frac{Q(z)}{P(z,D)})$ 记为 $\mathcal{L}$，则 $\log P(D) = -\mathcal{L} + KL(Q||P)$。显然 KL 散度一定是非零的，所以 $\log P(D)$ 的下界就是 $-\mathcal{L}$。

1.1.27 自动编码器/变分自动编码器

自动编码器是一种无监督的神经网络，用于学习输入数据的低维表示，并能够根据低维表示重建输入数据，也是一种将为的手段。

变分自动编码（Variational Auto Encoder）器运用了变分贝叶斯的思想，也继承了自动编码器的结构，但是存在一定的差别。在 VAE 中，默认将输入数据的低维表示（即隐变量）服从某个分布，一般认为服从正态分布。在 encoder 阶段学习到隐表示的分布 — 通常是学习到假定分布的参数，得到分布后就从该分布中进行采样得到隐表示；decoder 时，则将隐表示输入到 decoder 中。

参考资料：

- 变分自动编码器
- vae_tutorial

1.1.28 Koopman 分析

1.1.29 Model Collapse

模型坍塌，很形象啊，就是模型出现了一个漏洞，不管你输入什么东西都会漏进这个洞里。

这个主要出现在 GAN 的模型中。GAN 通过生成器 G 和判别器 D 来使 G 捕捉到真实数据的分布。在训练 GAN 模型时会出现 model collapse 的现象，即 G 只捕捉到了真实数据的部分分布。为什么会这样呢？简单的介绍一下：当 G 捕捉到了真实数据的部分分布后，被 D 识破了，于是 G 就改变，从某个分布跳到了另外的分布，并且抛弃了原来的分布，于是就成了“猫鼠游戏”，D 一直追着 G 跑，G 最终并没有完全捕捉到真实数据的分布。参考：GAN——ModeCollapse。

1.1.30 Inductive Bias

归纳偏置，使用某个算法解决问题时所基于的假设，类似于贝叶斯中的先验（prior），与先验不同，归纳偏置在学习过程中不会被更新，而先验会不断被更新。机器学习中常见的归纳偏置：奥卡姆剃刀、CNN

中的局部性、KNN 中假设相似样本在特征空间中也是相邻的、SVM 假设好的分类器应该是类别边界距离最大的等。

1.1.31 Covariate Shift

指机器学习中训练集和测试集样本分布不一致的现象。通常在机器学习中假设训练数据和测试数据的分布是一致的，通过训练数据习得的一组最优参数能否使得模型在测试数据上也有很好的表现呢？当训练集和测试集的分布不是那么相似时，covariate shift 就出现了。

怎么解决呢？训练集和测试集的分布不一致导致模型的参数不能很好地应用到测试集上，可能是测试集中地某些样本在训练集中被“轻视”或“过度重视”了，因此可以通过附加一个权重来解决该问题。可参考：covariate shift 现象的解释。

1.1.32 关于交叉熵损失函数的一点理解

交叉熵损失函数可以从多个角度进行理解。从概率论的角度，在极大似然概率估计中，希望参数能够使得已有样本出现的概率最大。

在分类任务中，由于不同样本是有标签的，样本出现的概率应该与标签对应，例如 p 表示 1 样本出现的概率，则 1-p 表示 0 样本出现的概率，那么对于 1 样本极大化的应该是 p，对于 0 样本极大化的应该是 1-p。

样本的极大化后的概率取对数后为相加的形式。则对于一个样本，其极大化的表示可以写成 $logp$ （1 样本），或者 $log(1-p)$ （0 样本）。如果用一个统一的式子表示的话，可以写成 $ylogp+(1-y)log(1-p)$，其中 y 取 0 或 1。

通常是对损失函数最小化，所以可以在极大化的表示前加个符号就变成了交叉熵损失函数：$-ylogp-(1-y)log(1-p)$。将一个 batch 里的样本的损失累加起来就是常见的形式了。并且，在用模型计算 p 时，p 通常通过一个函数来表示 $f(x)$，x 即为输入的样本，$f(x)$ 可以是各种机器学习模型，如逻辑回归、神经网络等。

除了从极大似然的角度来理解交叉熵损失函数，当然也可以直接从交叉熵1.1.16的角度来理解。

1.1.33 关于 CNN 的一点理解

于 DNN 相比，CNN 有两个特点：1）局部感知；2）权值共享。

局部感知。在 CNN 中每个 Feature map 中的一个值只于上一层的 Feature map 的一小部分有关。如果将 CNN 展开成 DNN 的形式，则一层中的一个神经元的输入只是上一层的某几个神经元。与全连接不同，全连接将上一层的所有信息作为输入，捕捉整体的信息。但局部感知以某个区域内的信息作为输入，捕获上一层数据中存在的局部信息 — 特定的模式。

权值共享。CNN 中的每一个 filter 都可以得到一个 Feature map，同一个 Feature map 中的元素之间通过同一个 filter 卷积得到。或者说，同一层的神经元由一个 filter 产生，共享一套参数。见 Fig.1。图来源于李宏毅老师的 ppt。有一点需要注意，CNN 模型的输入图像可能是单通道的也可能是三通道的，此时**每个 filter 不仅由长和宽还有深度**，filter 的深度与图像的通道数是相等的，所以一般来说有多少个 filter 就有多少个 Feature map。

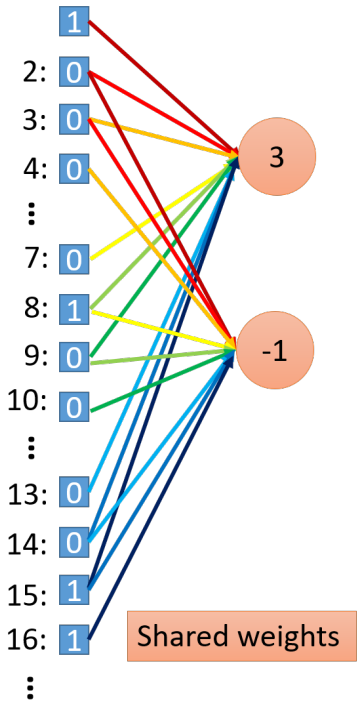


图 1: CNN 的权值共享

CNN 的应用非常广泛，不仅是图像，在 Speech、NLP 等很多领域都有非常多的应用。在决定是否要使用基于 CNN 的模型时，需要考虑：1）数据中存在一些较小的模式，这些模式经常出现在数据中 — 对用卷积核；2）模式与整个数据相比比较小，更大的语义信息是由很多小的模式组成的；3）进行 pooling 时不会破坏数据本来的含义。

在探索 CNN 到底学习到了什么的时候，可以在训练好模型后，通过不断优化输入的数据，来检验是什么样的输入使得基于 CNN 的模型中的神经元能够得到最大的激活 — 什么样的输入使神经元最兴奋。这些主要涉及到对 CNN 的解释和可视化。

1.1.34 DL 中的不可微操作

1.1.35 batch normalization、layer normalization

1.1.36 Global Max Pooling

1.1.37 Attention 机制与 CNN

1.1.38 学习率衰减与参数正则化

1.1.39 深度学习中常见的参数初始化方法

Glorot initialization

1.1.40 机器学习中常用的算法指标及其应用场景

Accuracy, Recall, Precision, F-score $ACC = \frac{TP+TN}{TP+FN+TN+FP}$ $R = \frac{TP}{TP+FN}$ $P = \frac{TP}{TP+FP}$ $F(\beta) = (1+\beta^2)\frac{P \cdot R}{\beta^2 P+R}$。混淆矩阵（Confusion Matrix）见表.1。F-score 是对 R，P 的一个综合评价，β 度量了 R 相对于 P 的重要性，可以理解为 $\beta = \frac{importance(R)}{importance(P)}$，则表示 β 越大，我们越看重 R。

表 1: 二分类混淆矩阵

	Actual class (observation)	
Predicted class (expectation)	tp (true positive) Correct result	fp (false positive) Unexpected result
	fn (false negative) Missing result	tn (true negative) Correct absence of result

关于 *Precision*，*Recall* 的选择？ 这几个指标在很多任务中都有应用，，但是不同的指标侧重于不同的方面，比如 P、R 都有不同的侧重，但看一个指标是比较片面的，并不能反映出模型真实的效果。有可能 P 很高，但是 R 很低，而在一些场景下 R 是很重要的。

比如在金融风控等领域，我们希望算法能够尽量识别出所有有可能有风险的用户，这时候就侧重于 Recall，即希望算法把所有的正样本（通常有危险的，需要被找出来的被标记为正样本）都筛选出来，即使将所有样本都标为正（这是 Recall=1）。因为这种情况下，可能漏掉一个正样本带来的代价是极大的，通常筛选完之后还需要交给人工进行判断。类似的场景还有癌症检测，将一个没有癌症的判断为癌症没有很大关系，但是将一个有癌症的判断为没有癌症则是很严重的，会出人命的!!!

又比如在垃圾邮件分类中，我们可能更侧重于 Precision。我们希望算法在识别垃圾邮件时不要把正常邮件错分了，这个时候希望 Precision 尽可能高，即使将所有样本标为负也没关系（TP=0 时可认为 Precision=1），或者说算法只把自己十分确信为垃圾邮件的标为正，尽量降低 FP，即尽量不要把正常邮件视为垃圾邮件，不然错过了 offer 那可咋整!!!

通常，将需要识别出的类别，或者简单的说坏的一类为正样本，为什么呢? 因为好的漏掉一般不会产生啥大的影响，但是坏的跑了课就不行了！当然，也需要不同场景下选择合适的指标!!!

我们是贪心的，因此就有了一些综合的指标，比如 F-score。Precision 是以被分类的所有样本为分母，Recall 则是以原本所有的 positives 元素为分母。二者之间并没有建立直接联系，如果一个分类器，Precision 很高但是 Recall 很低，或者 Recall 很高但是 Precision 很低，这两种分类器都是不好的，都是我们不希望的。所以我们采用 F1-Score 来建立 Precision 和 Recall 的联系。

在数学中，调和平均数是永远小于等于算术均值平均数的，当用于求两个数的平均数时，如果直接用算术平均作为结果，那么两数之间的差异将被大的值削平，而调和平均数则不会极大削平这种大的差异，得到的结果更倾向于小的值。

Micro-F1 & Macro-F1 基本的 F1 使针对二分类任务而言的，在多分类中中，Micro-F1 和 Macro-F1 是两种求多类别 F1 均值的方式。

- Micro-F1：分别计算每个类别的 TP, FN, FP，再求整体的 $Recall$，$Precision$，再以整体的 P, R 来求 $F1$，得到 Micro-F1。在计算公式中考虑到了每个类别的数量，所以适用于数据分布

不平衡的情况；但同时因为考虑到数据的数量，所以在数据极度不平衡的情况下，**数量较多的类（即常见的类）会较大的影响到 F1 的值**

- Macro-F1：分别计算每个类别的 F1，再求平均，得到 Macro-F1。没有考虑到数据的数量，所以会平等的看待每一类（因为每一类的 precision 和 recall 都在 0-1 之间），**会相对受高 *Precision* 和高 *Recall* 类（即稀有的类）的影响较大**

$Micro-F1$ 公式如下所示：

$$
\begin{aligned}
Recall_{mi} &= \frac{\sum_i TP_i}{\sum_i TP_i + \sum_i FN_i} \\
Precision_{mi} &= \frac{\sum_i TP_i}{\sum_i TP_i + \sum_i FP_i} \\
Micro-F1 &= \frac{Recall_{mi} \times Precision_{mi}}{Recall_{mi} + Precision_{mi}}
\end{aligned}
$$

$Macro-F1$ 如下所示：

$$
Macro-F1 = 2\frac{\sum_i F1_i}{N}
$$

ROC、AUC 接收者操作特征曲线（recevier operating characteristic curve），用于反映一个而分类器的灵敏度（sensitivity）和特异度（specificity）之间的关系。

$$
\begin{aligned}
Sensitivity &= \frac{TP}{TP + FN} \\
Specificity &= \frac{TN}{TN + FP}
\end{aligned}
$$

其中 Sensitivity 也就是 TPR（True Positive Rate），也就是 Recall，Specificity 是 TNR（True Negtive Rate）。ROC 的横坐标是 $1 - Specificity$，即 FPR（False Positive Rate），纵坐标是 Sensitivity。横坐标表示的是负样本中被预测为正样本的比例，纵坐标表示的是正样本中被预测为正样本的比例。

ROC 如 Fig.2所示。

ROC 的绘制过程：对于一个二分类问题，使用一个分类器对样本集进行预测后，可以得到每个样本属于正样本的概率，此时我们还需要一个阈值来确定那些为正样本。每选取一个阈值，就可以得到一个（TPR，FPR）数值对。当阈值从 1 到 0 不断减小时，被确定为正样本的样本数不断增大，其中 TP 和 FP 都会不断增大，由于正负样本的数量是固定的（即 TPR，FPR 的分母是固定的），则 TPR 和 FPR 都会不断增大。那么，

- 当阈值为 1 时，（几乎）所有样本都为负样本，则 TPR 约为 0，既然都为负样本那么 FP 也为 0，则 FPR 也为 0
- 当阈值为 0 时，所有样本都为正样本，那么肯定所有的正样本都找出来了，则 TPR 为 1，由于所有样本都预测为正样本，那么肯定所有的负样本都预测为了正样本，则 FPR 为 1

因此，在阈值从 1 到 0 的过程中，（TPR，FPR）不断增大，从坐标（0，0）到（1，1），如 Fig.3(a)所示，Fig.3(a)上部分为负样本为正样本的概率的分布图（即横坐标为正样本概率值，纵坐标为对应的样

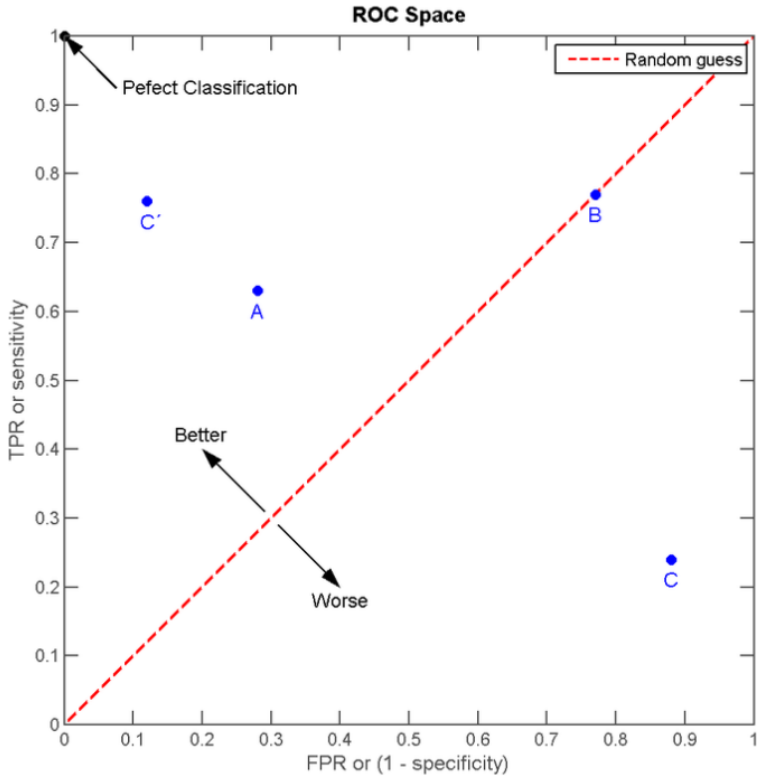
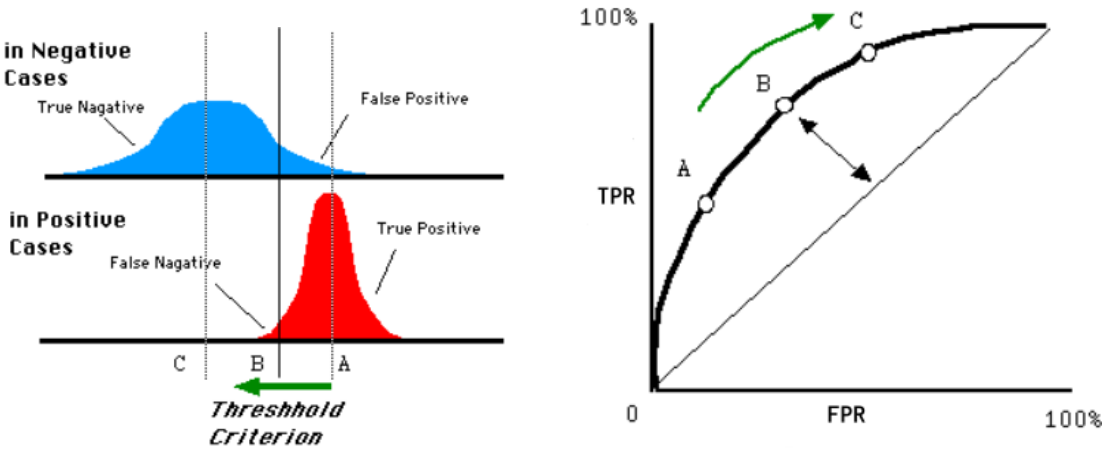


图 2: ROC

本数量），下部分为正样本为正样本的概率的分布图，可见，当阈值为 B 时，大部分正样本都被分为了正样本（TP），小部分负样本被分为了正样本（FP）。由 Fig.3(b)可见，当阈值减小时，（TPR，FPR）的变化过程。



(a) 阈值变化与预测正负样本的分布

(b) 阈值变化与 ROC

图 3: 阈值变化

ROC 是一个曲线，那怎么作为一个指标呢? — 取 ROC 与坐标轴围成的面积，即 **AUC (Area Under Curve)**。由于 ROC 的绘制过程，我们希望当阈值为接近 0 时，TPR 尽量高，FPR 尽量低（其实不管阈值为何值，都希望有这个效果），一个好的分类器的 ROC 的 AUC 应该尽量大。

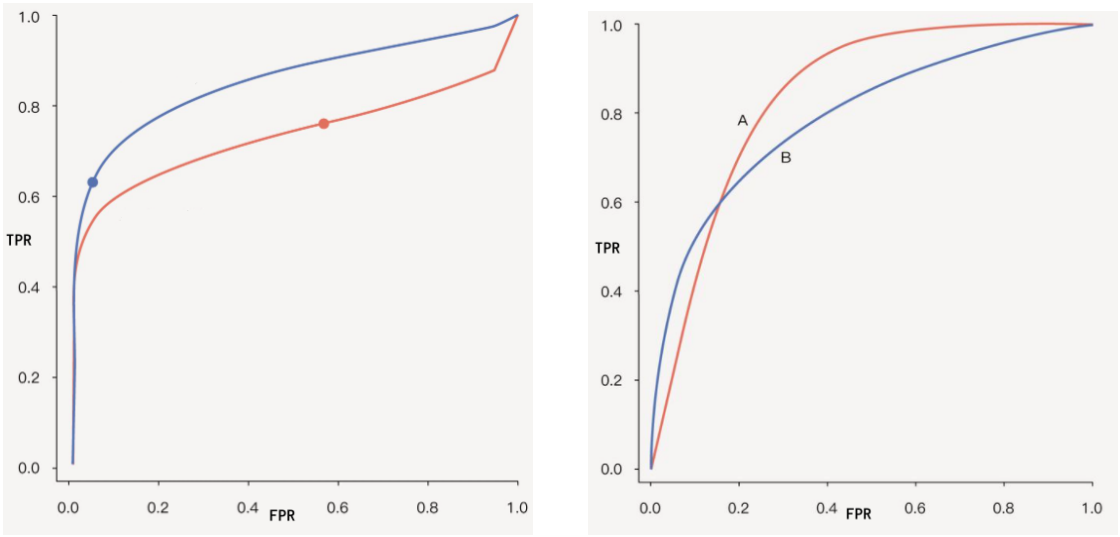
AUC 的含义：随机挑选一个正样本、一个负样本，分类器分别给出一个分数，正样本的分数大于负样本的分数的概率。

为什么要用 ROC/AUC 呢？

因为 ROC 曲线有个很好的特性：当测试集中的正负样本的分布变化的时候，ROC 曲线能够保持不变。在实际的数据集中经常会出现类不平衡 (class imbalance) 现象，即负样本比正样本多很多 (或者相反)，而且测试数据中的正负样本的分布也可能随着时间变化。roc 曲线不变原因：TPR 和 FPR 是实际 label 内部的操作，看混淆矩阵和 tpr、fpr 计算公式，无论实际 label 比例怎么变化，tpr、fpr 计算公式都是在实际为 p 或者 n 的内部计算的

如何使用 ROC 来选择模型？

当我们有多个分类器时，给定一个数据集，可以得到多条 ROC 曲线，那么怎么来选择模型呢? 一个很直观的想法是直接比较 AUC。但是在不同场景下，我们要结合更看重的指标选择模型。如 Fig.4所示，当 ROC 不交叉时，可以直接选择 AUC 高的，当 ROC 交叉时则需要慎重考虑了。当需要高的 Sensitiviy 时，选择 A，需要高 Specificity（即低 FPR）时选择 B。



(a) ROC 不交叉　　(b) ROC 交叉

图 4: ROC 比较

参考资料：

- 分类模型评估之 ROC-AUC 曲线和 PRC 曲线
- ROC 曲线

mIoU　Mean Intersection over Union(MIoU，均交并比)，为语义分割的标准度量。其计算两个集合的交并比，在**语义分割**的问题中，这两个集合为真实值（ground truth）和预测值（predicted segmentation）。

令 p_{ij} 表示实际类别为 i，预测类别为 j 的数量，则

$$mIoU = \frac{1}{C}\sum_{i=1}^{C}\frac{p_{ii}}{\sum_{j=1}^{C}p_{ij} + \sum_{j=1}p_{ji} - p_{ii}}$$

如下图 Fig.5所示：在**语义分割**中，被分类的对象为每个像素，真实标签为该像素所属的类别，预测标

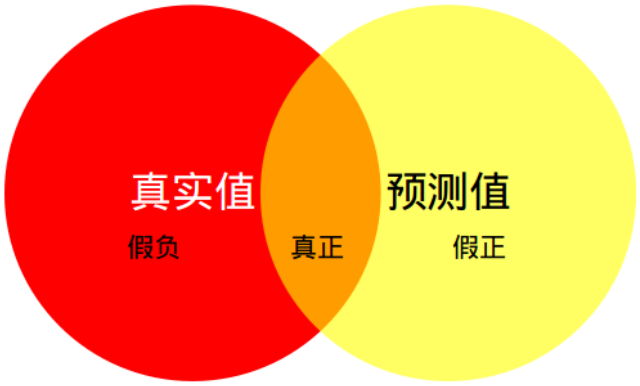


图 5: mIoU

签为预测的类别。计算时，可以先计算出混淆矩阵，将对角线上的元素的值之和除以混淆矩阵中所有元素的和，再除以类别数就是 mIoU 了。注意，在实际计算中，要注意除零的情况。

Dice　有 Dice 系数和 Dice loss 之分。Dice 系数是一种集合相似度度量函数，通常用于计算两个样本的相似度，取值范围在 [0,1]，计算公式如下：

$$dice(x, y) = \frac{2|x \cap y|}{|x| + |y|}$$

在**语义分割**中，x, y 可以分别代表预测的分割结果、真实的分割，分别以矩阵的形式表示。那么，计算模型的分割效果可以为：

$$dice(pred, ground) = \frac{2(pred \cdot ground).sum()}{pred.sum() + ground.sum()}$$

其中，$\cdot$ 和$.sum()$ 分别表示矩阵的逐元素乘积、逐元素求和。Dice loss 则是：$1 - dice(pred, ground)$，Dice loss 首次在 VNet 中提出。

在图像分割实践中，可以用 Dice loss 或者交叉熵损失函数作为目标函数，但是由于交叉损失函数的梯度形式更优，更倾向于选择交叉熵损失函数。Dice Loss 特点：

- 训练误差曲线非常混乱，很难看出关于收敛的信息。尽管可以检查在验证集上的误差来避开此问题
- Dice Loss 比较**适用于样本极度不均的情况**，一般的情况下，使用 Dice Loss 会对反向传播造成不利的影响，容易使训练变得不稳定
- Dice 对 mask 的内部填充比较敏感

作为 Dice loss 的一个替代，可以使用 cross entropy loss。原因：

使用交叉熵做损失函数时，在反向传播时，计算得到的梯度的形式是类似于（p-t）的，其中 p, t 分别是预测值和标签。而 dice loss，如果将其写成 $\frac{2pt}{p^2+t^2}$ 或 $\frac{2pt}{p+t}$ 的形式，则在反向传播时，梯度大概时这个样子的：$\frac{2t(t^2-p^2)}{(p^2+t^2)^2}$ 或 $\frac{2t^2}{(p+t)^2}$。这样的梯度有什么问题呢？当 p, t 都很小时，梯度可能会变得很大，这也是 **dice loss 在训练过程中不稳定的原因**了。

Rand Error Rand Error 是以 Rand Index（兰德系数）为基础的。Rand Index 用于衡量两个数据簇之间的相似性。Rand Index 的定义：对数据点集进行分类时，用 a 表示实际为同一类，预测时也为同一类的数据点对（pair）的数量，b 表示实际为不同类，预测时也为不同类的数据点 pair 的数量，则

$$RI(RandIndex) = \frac{a+b}{\mathrm{C}_n^k}$$

Rand Error 定义为：

$$RE = 1 - RI$$

RE 可以用于衡量图像分割算法的效果。可以参考：Rand Index 计算。

Hausdorff 距离 可以用于衡量两个点集之间的距离，定义如下，其中 $\boldsymbol{X}, \boldsymbol{Y}, d(x, y)$ 表示两个点集和点之间的距离度量函数：

$$d_H(X, Y) = \max(d_{XY}, d_{Yx}) = \max\left\{\max_{x\in X}\min_{y\in Y} d(x, y), \quad \max_{y\in Y}\min_{x\in X} d(x, y)\right\}$$

Dice 缺陷在于对边界的刻画不敏感，注意力主要集中在 mask 的内部。而 Hausdorff 距离作为形状相似性的一种度量，能够为 Dice 做出较好的补充。

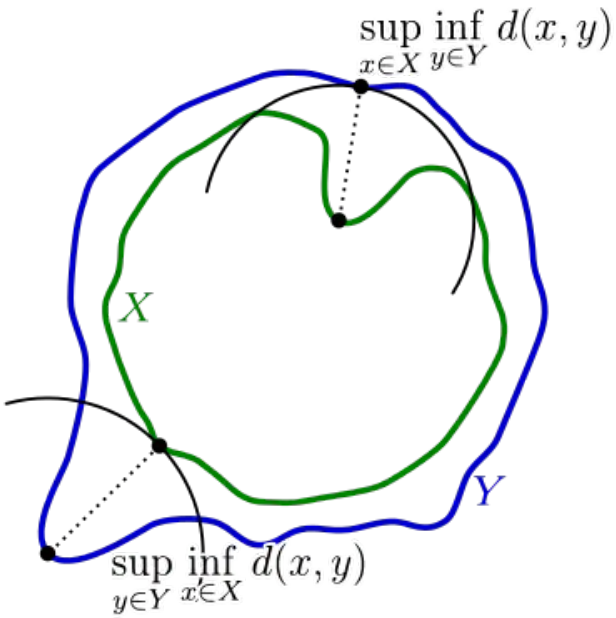


图 6: Hausdorff Distance

MRR Mean Reciprocal Rank，常用来衡量搜索算法效果的指标，目前被广泛用在允许返回多个结果的问题，或者目前还比较难以解决的问题中（由于如果只返回 top 1 的结果，准确率或召回率会很差，所以在技术不成熟的情况下，先返回多个结果）。在这类问题中，系统会对每一个返回的结果给一个置信度（打分），然后根据置信度排序，将得分高的结果排在前面返回。核心思想很简单：返回的结果集的优劣，跟第一个正确答案的位置有关，第一个正确答案越靠前，结果越好。定义如下：

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

其中 Q 为查询集合，$rank_i$ 是第 i 个查询的结果集中正确结果的排名。

AP Average Precision，平均精确率。对于二分类问题，给定样本真实标签 $\{y_1, ..., y_n\}$ 和模型预测的正样本置信度 $\{c_1, ..., c_n\}$，计算 AP：

1. 按照置信度从大到小对样本进行排序，令排序后的样本的置信度为 $\{c_1, ..., c_n\}$
2. 令 $i = 1$，重复执行：
   (a) 以 c_i 为阈值，得到预测的正负样本，即前 i 行预测为正样本，之后的均为负样本
   (b) 计算当前阈值下的 Recall 和 Precision
   (c) $i+ = 1$
   (d) 当 $i > n$ 则结束循环
3. 排序后的每个样本都对应一个 (Recall, Precision) 对，即 $\{(r_1, p_1), ..., (r_n, p_n)\}$
4. 上一步得到的 Recall 列表 $\{r_1, ..., r_n\}$ 中的元素 r_i 与 r_{i-1} 做差，设 $r_0 = 0$，则可以得到 $\{d_1, ..., d_n\}$，其中 $d_i = r_i - r_{i-1}$
5. 求和：$AP = \sum_{i=1}^{n} d_i \cdot p_i$

如 Fig.7所示，该表是按照置信度排序后的样本，correct 列表示该样本的真实标签，P、R 列是按照上述方式计算出来的 Recall 和 Precision。

MAP Mean Average Precision，是 AP 的均值。AP 通常是针对单个类别而言的，当有多个类别时，分别计算每个类别的 AP，再进行算术平均。

注意： AP 和 MAP 常用在目标检测和信息检索领域中。

- 目标检测领域中，对于一个类别，可能会检测出多个检测框，每个检测框可以根据 IoU 来判断检测框的真实标签是 1 还是 0（针对一个类别的检测而言，**因为目标检测中的 Ground Truth 也是一个边界框，所以不需要和边界框完全一致才作为正样本，只要 IoU 大于一定阈值即可**），每个检测框还对应一个置信度。此时即可按照上述方式计算该类别的 AP
- 在信息检索领域，对于一个查询 q，通常希望模型能够给出 top K 个结果（**有序的**）。若 q 真实的有序搜索结果列表为 $\{d_1, ..., d_m\}$，则可以对这 K 个结果的标签，由于模型的输出已经是排好序的，故可以直接计算 AP。对于多个查询，则计算每个查询的 AP 后再取平均

rank	correct	P	R
1	right	1/1	1/5
2	right	2/2	2/5
3	wrong	2/3	2/5
4	right	3/4	3/5
5	wrong	3/5	3/5
6	wrong	3/6	3/5
7	right	4/7	4/5
8	wrong	4/8	4/5
9	wrong	4/9	4/5
10	wrong	4/10	4/5

图 7: AP 计算示例

总而言之，根据某种方法判定样本的真实标签（如目标检测中通过 IoU 判定、分类任务中给定的标签、搜索中给定的真实搜索列表等），再根据置信度进行排序（如目标检测中的置信度、分类任务中输出的分数、搜索中直接给出的排序等），计算每个位置处的 $(recall, precision)$，按照上述方式即可算出 AP。

DCG、NDCG Discounted Cumulative Gain，折扣累计增益。在介绍 DCG 之前有必要先介绍一下 CG，Cumulative Gain，即累计增益。这两个指标主要用于搜索领域。对于模型返回的 p 个结果（**有序的**），每个位置处的结果与查询的相关性为 rel_i，则该结果的 CG 为：

$$CG_p = \sum_{i=1}^{p} rel_i$$

很明显，CG 没有考虑结果的先后顺序，在搜索中，结果的顺序是至关重要的，因此产生了 DCG：同一个相关度，排名越后则增益越小，即与所处排名成反比。

$$DCG = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} \qquad or \qquad \sum_{i+1}^{p} \frac{2^{rel_i}-1}{\log_2(i+1)}$$

通常，不同的查询对应的结果列表是不一样长的（**不同长度的搜索结果对应的 DCG 值范围不一样，无法直接比较，例如长的结果列表 DCG 最大可为 10，短的最大可为 5，前者的 DCG 为 4 和后者的 DCG 为 4 并不代表二者的结果列表质量一样**），因此不能将 DCG 用于评价结果列表长度不同的查询效果，因此需要对 DCG 进行归一化，即 NDCG（Normalized DCG，归一化折扣累计增益），

$$NDCG = \frac{DCG}{IDCG}$$

其中 IDCG 为理想情况下的折扣累计增益，表示真实的结果列表的 DCG，计算方式为取真实结果列表的前 p 个结果计算 DCG，即

$$IDCG = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

注意： MAP 和 NDCG 都可以用于衡量搜索结果的质量，但是 MAP 只支持两种相关性：{相关，不相关}，NDCG 可以支持多种相关性得分，如 1-5。

1.1.41 深度学习模型中特殊的结构

Residual learning block 跳跃连接，基本的残差块如 Fig.8所示（当然残差块不止有这一种形式，可以根据需求定义不同的残差块）。残差学习由何凯明基于以下问题提出：给定一个学习问题后，逐渐加深网络的层的时候，模型的效果应该是逐渐提升的或者不能低于原来模型的效果，但是在实验中发现通常加深后模型的效果反而变差了。按理来说就算不能提升了，额外增加的层也可以学习到一个恒等映射来保持效果不变啊，但是为什么反而下降了呢？这便是**模型退化问题**。

假设原本要学习的问题是 $\mathcal{H}(x)$，之前的想法是直接学习它，在残差学习中，将它进行分解 $\mathcal{H}(x) = \mathcal{F}(x) + x$，由于 x 是已知的，那么只需要学习 $\mathcal{F}$ 就好了，$\mathcal{F}$ 也就是所说的残差。

为什么这样会有效呢？由于神经网络中通常都会使用非线性函数来拟合复杂的函数，但是对于线性关系却有点力不从心（可能这就是为什么不能学习到恒等映射的原因吧）。残差学习不仅保留了学习非线性函数的能力，也提高了线性函数的学习能力 — $\mathcal{F}$ 为 0 即可。Resudial Learning 的这种能力使得更深的网络称为可能。

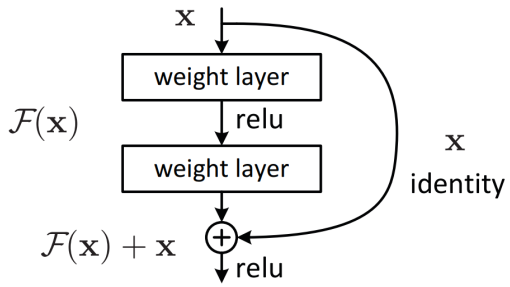


图 8: Residual

Dense Connection 稠密连接，如 Fig.9所示。每一层与之前的所有层都有连结。用 H_l 表示第 l 层，$\boldsymbol{X}_l$ 表示第 l 层的输出，则 $\boldsymbol{X}_l = H_l(\boldsymbol{X}_0, \boldsymbol{X}_1, ..., \boldsymbol{X}_{l-1})$。

有以下优点：1）减轻了梯度消失问题；2）加强了特征的传播，能够有效的利用学习到的特征；3）能够利用多层次的特征。

Dilated Convolution 空洞卷积。在 pooling 时，可以减小 feature map 的尺寸，也能增大每个元素的感受野但是也损失了空间信息。在进行分割时，pooling 损失的那部分信息是难以复原的。dilated 卷积是一种特殊的卷积，与通常的卷积不同，dilated 卷积会在**卷积核元素之间插入空格**，其实相当于一个更大的卷积核，而那些插入的卷积核的值一直为 0。如 Fig.10所示。Dilation 卷积可以在不做 pooling

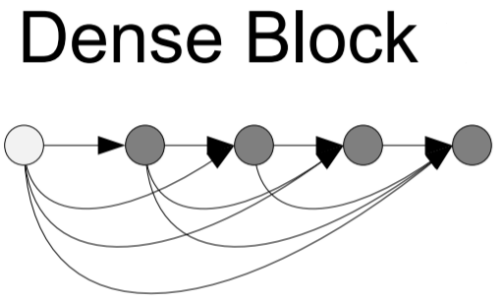


图 9: Dense Block

损失信息的情况下增大感受野，pooling 虽然可以增大感受野但是失去了位置信息，难以从 pooling 后的层恢复到原来的信息，而 dilation 卷积不仅增大了感受野，还保留了特征图中元素的相对空间信息。

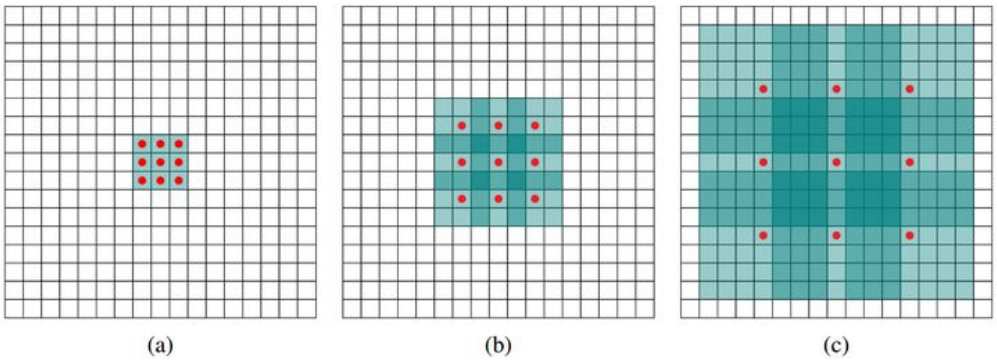


图 10: Dilation Convolution

1×1 Conv 1×1 卷积很显然，就是卷积核的长宽均为 1，用 c_i 和 c_o 分别表示输入和输出通道的数目，为了获得多个通道的输出，可以为每个输出通道创建一个形状为 $c_i \times 1 \times 1$ 的卷积核张量，这样卷积核的形状是 $c_o \times c_i \times 1 \times 1$。

由于 1×1 卷积的长宽为 1，无法关注到周围像素的信息，只能关注到同一位置不同通道上的信息，其作用主要由以下几点：

- 融合多个通道间的信息
- 在不改变 feature map 尺寸的情况下改变 feature map 的通道数，例如在图像分割中
- 降低参数量
- 在以每像素为基础应用时，1×1 卷积相当于全连接层

更多解释可参考动手学深度学习：1×1 卷积层

1.1.42 常用数据增强手段

增强之前，先想一想：真的需要增强数据吗（通常来说是的）？需要增加的多少数据？需要增加什么样的数据（并不是什么样的数据都可以，主要考虑应用场景中一般会出现的数据即可）？

仿射变换 仿射变换（Affine Transformation）是指在二维向量空间中进行一次线性变换 (乘以一个矩阵) 和一次平移 (加上一个向量)，变换到另一个向量空间的过程。

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

图解，放射变换的种类也如 Fig.1.1.42所示：

弹性形变 [17] 最早是从 UNet 中了解到弹性形变，在细胞分割中，弹性形变发挥了重要作用；弹性形变也用在手写数字识别中。可以发现，在这两种任务中，任务所涉及的对象并不是刚体，简单的仿射变换并不能满足我们的需求。弹性形变所针对的数据特点：对象不是刚体，可能在不同的场景下会有形变。

弹性形变流程：

- 对图像 imageA 进行仿射变换，得到 imageB
- 对 imageB 图像中的每个像素点随机生成一个在 x 和 y 方向的位移，Δx 和 Δy。其位移范围在 (-1, 1) 之间，得到一个随机位移场 (random displacement fields)
- 用服从高斯分布的 $N(0, \delta)$ 对 step2 中生成的随机位移场进行卷积操作 (和 CNN 中的卷积操作一样，说白了就是滤波操作)。我们知道 越大，产生的图像越平滑。下图是论文中的不同 值对随机位移场的影响，下图左上角为原图，右上角为 δ 较小的情况 (可以发现，位移方向非常随机)，左下角和右下角为较大的不同 δ 值
- 用一个控制因子 α 与随机位移场相乘，用以控制其变形强度
- 将随机位移场施加到原图上，具体是**怎么施加的呢**？首先，生成一个和 imageB 大小一样的 meshgrid 网格 meshB，网格中的每个值就是像素的坐标，比如说 meshgrid 网格大小为 512x512，则 meshgrid 中的值为 (0, 0), (0, 1), ..., (511, 0), (511, 511)，然后将随机位移场和 meshB 网格相加，这就模拟了 imageB 中的每个像素点在经过随机位移场的作用后，被偏移的位置，meshB 与随机位移场相加后的结果记做 imageC
- 弹性变形最终输出的 imageC 中每个位置的灰度值大小，组成一副变形图像，现在 imageC 中每个像素点存储的是 $(\mathrm{x}+\Delta\mathrm{x}, \mathrm{y}+\Delta\mathrm{y})$，如下图中的 A′，那怎么转化成灰度值呢，依据论文，作者是根据 imageB 中的 B 位置的双线性插值灰度值作为 A′ 点的像素灰度值大小 (如 Fig.1.1.42所示)，最终将 imageC 输出得到变形图像

参考：数据增强：弹性变形 (Elastic Distortion)。

增加噪声 如椒盐噪声。

GAN

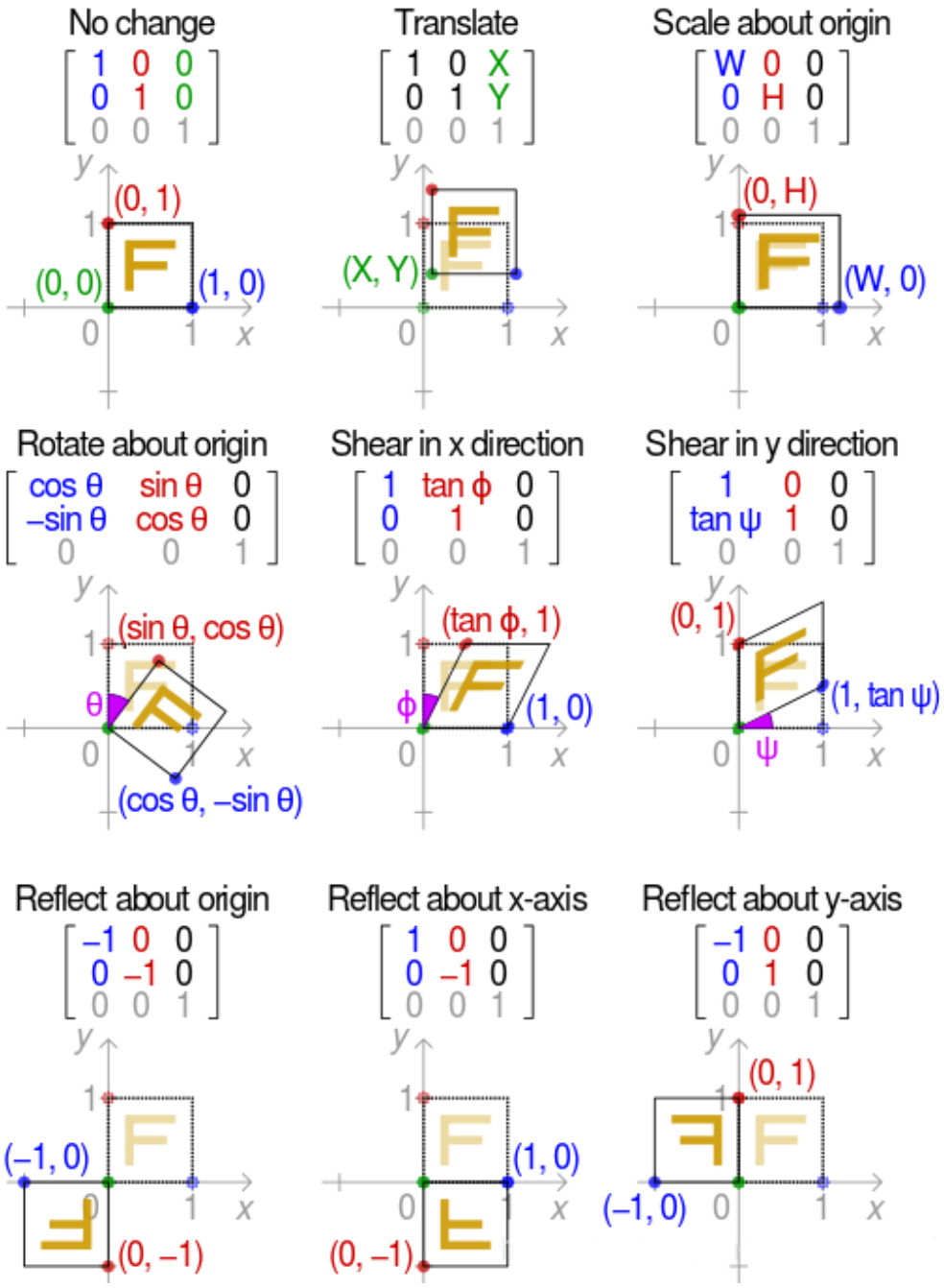


图 11: 仿射变换

1.1.43 方差与偏差

偏差（Bias） 讲模型的方差的时候，通常是指：对于一个模型，它的预测结果与真实值之间的差距。方差度量的是学习到的函数与真实函数的差距的期望，即 $E[\hat{f} - f]$。

方差（Variance） 选用一种模型（如 KNN，SVM，NN 等），使用不同的数据可以得到该种模型的多个实例（即训练好参数的模型）。**这些模型**对于同一个输入给出的预测值的方差，即 $E[(\hat{f} - E[\hat{f}])^2]$。

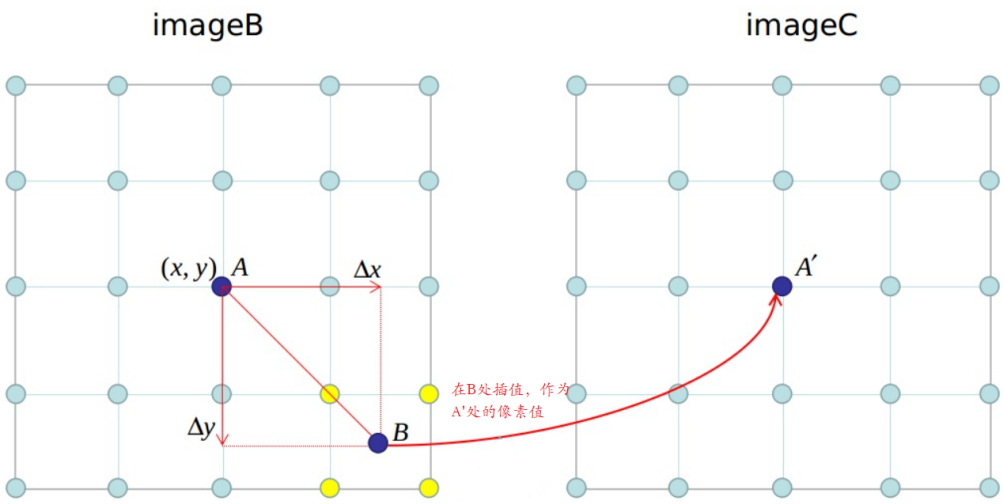


图 12: 弹性形变

方差是受所使用的训练集影响的，我们使用不同分布的数据集训练出来的模型会导致学出来的模型参数变化，这反映出来的就是针对同样的输入会产生不一样的预测值，这些预测值的方差就反映了模型的方差。

在模型很复杂的时候，在训练集上学习的模型能够准确地预测，产生较小的偏差，但是更容易产生大方差，因为模型有更多的参数，能够“死记硬背”记下输入与输出的映射关系，这个时候模型可能记住了一些没用的东西甚至是噪声，而模型是没有见过测试集的，稍微有些不同的数据点的输出可能会有较大的差别，从而产生大的偏差。— **过拟合**

当模型较简单的时候，可能无法充分地利用训练集，在测试集上会产生较大的偏差，但是因为模型参数少，模型关注地特征较少，光从这些特征来看的话，测试集中的数据与训练集中的数据会有更高的相似性，因此就算不同的数据输入，但是由于某些特征被忽略后数据反而是相似的，所以模型给出的输出是类似的，因而方差小。— **欠拟合**

参考：Understanding the Bias-Variance Tradeoff。

注意：一种模型相当于一个函数空间，当我们选择一种模型进行训练，得到了该种模型的各个参数，就相当于得到了一个实例化的模型。模型种类相当于类，训练好的模型相当于对象。喂数据训练模型时相当于在某个特定的函数空间中找到一个合适的函数 — 即我们要的模型。有时候训练后的模型效果不好，可能不是这种模型不行，可能是没有在这个函数空间中找到合适的函数。

1.1.44 类别不平衡问题

指的是监督学习中，不同类别的样本数目具有较大的差异（样数据分布与均匀分布差异较大）。

类别不均衡可能造成的问题

- 一些评价指标可能失效。例如在癌症检测中，可能 99% 的样本都是 0，只有 1% 的样本为 1，这个时候即使将所有样本预测为 0 也能有很好的 acc，但是漏检是很严重的！类别不平衡使得一些评价指标并不能反映模型真是的能力

-

决解办法

数据角度 中心思想就是直接改变数据分布。

- 获取更多数据，使数据分布趋向于均衡
- 上采样。通过一些方法，使得占少数的类别（minority 类）的样本数增加，常用的方法：
  - 重复采样 minority 类，使其样本数增加
  - 合成的方法。根据已有数据集生成新的样本，如 SMOTE 方法及其变体
  - 基于聚类。分别对 major 和 minority 类进行聚类，再通过过采样的方法使得 major 和 majority 中各个簇的数量相等，例如原本 major 类聚类后样本数目比为 1:2:3，minority 类聚类后为 1:2，通过过采样的方法，先使 majority 与 minority 样本数相等，再使类内部各个簇的数目相等。这样不仅可以解决类别间的不平衡，还可以解决类内部的不平衡
- 下采样。通过一些方法把占多数的类别（major 类）的样本数降低。下采样的方法有很多
  - 随机下采样，从 major 类中随机保留一部分样本
  - 基于临近样本，来选择保留哪些 major 类样本
  - 基于聚类，对 major 类进行聚类，使其具有 N（minority 类样本数）个簇，用这 N 个簇的中心作为 major 类采样后的样本

算法角度

- 选择对数据倾斜相对不敏感的算法，如树模型等
- 在下采样时会损失一部分信息，可以从 major 类中采样多个数据集来学习不同的模型，即集成学习（Ensemble 集成算法）。首先从多数类中独立随机抽取出若干子集，将每个子集与少数类数据联合起来训练生成多个基分类器，再加权组成新的分类器，如加法模型、Adaboost、随机森林等
- 将任务转换成异常检测问题。譬如有这样一个项目，需要从高压线的航拍图片中，将松动的螺丝/零件判断为待检测站点，即负样本，其他作为正样本，这样来看，数据倾斜是非常严重的，而且在图像质量一般的情况下小物体检测的难度较大，所以不如将其转换为无监督的异常检测算法，不用过多的去考虑将数据转换为平衡问题来解决

评价指标角度

- 混淆矩阵
- F-score

1.1.45 交叉验证

在训练模型的时候，通过会把数据集划分为训练集和测试集，测试集的作用用于学习模型参数，测试集是为了检验模型在未见过的数据上的效果。但是只将数据集划分为测试集和训练集有以下问题：

- 最终模型与参数的选取将极大程度依赖于你对训练集和测试集的划分方法。不同的划分情况，学习出来的参数是不一样的；固定模型参数，模型在不同划分上的表现也是不一样的。这种情况使得我们无法准确对模型的能力进行评估，不利于我们选择最优的模型（指何种模型）以及最优的模型参数（一般是指超参数，而不是模型学习到的参数）；
- 该方法只用了部分数据进行模型的训练，无法充分利用已有的数据。测试的效果只是针对某个划分，并不是针对整个数据集，不够有说服力；

为了解决以上为题，即 1）选择最好的模型与参数；2）充分利用数据，使模型的测试结果有说服力，就有了交叉验证（Cross Validation）。常见的交叉验证方法：

- LOOCV(Leave-one-out cross-validation),即留一验证。对于有 N 个样本的数据集,重复 N 次,每次选择其中一个作为测试集，其他的作为训练集，这样就得到了 N 个 $(train_i, test_i), i = 1, ..., N$ 训练集、测试集对儿。分别用这 N 个训练集-测试集对儿来训练模型，这样就可以学习到 N 个模型。每个模型都可以得到一个测试得分，进行平均后就作为这类模型在这个数据集上的测试分数
- K-fold CV，即 K 折交叉验证。与 LOOCV 类似，但是对数据集的划分不同，是把数据集划分为 K 份，每次取其中一份作为测试其，其余作为训练集，这样就可以得到 K 个测试得分，进行平均后作为最终的测试分数

注意：给定模型种类和一组超参数，这样就确定了一个模型，但是模型的参数需要通过数据来学习（如 Fig.1.1.45所示），比如线性回归中的权重，上述的 N 个或 K 个模型是针对某个模型种类和一组超参数组合而言，K 或 N 个不同的训练集学习到的 K 或 N 个模型，但它们的超参数都是一样的，具体过程就是：**确定模型类别 ==> 确定超参数 ==> 喂数据进行学习（喂 K 次不同的数据）==> 得到 K 个模型（但是超参数都一样）==> 计算平均得分 ==> 得到这种模型在这种超参数下的得分**，这也就是 CV 可以用于选择模型（**选择模型最优的超参数**）的原因。

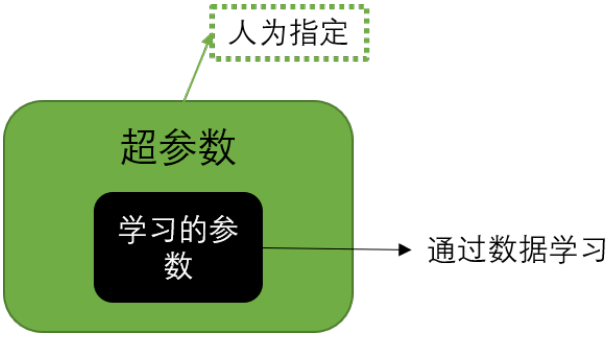


图 13: 模型的超参数与学习参数

一些值得注意的问题：

- LOOCV 计算成本太高而且不同训练集之间的重合度太高
- K 的选取。K 太大，投入的训练集也大，得到的模型可能会有较小的偏差，但是由于训练集之间的重叠度较高，会存在较高的方差

1.1.46 归一化 vs 标准化定量的分析

参考：https://mp.weixin.qq.com/s/lO3Li7dWAvtzefVA_M0dJg

2 NLP

2.1 常见 NLP 任务

2.1.1 命名实体识别

NLP 中的命名实体识别（Named Entity Recognition）指的是将识别出文本中描述实体的词汇，比如人名、地名、组织机构名、股票基金、医学术语等，更一般地讲，不同的领域会有不同的实体，用于描述该领域内的实体对象。NER 的目的就是识别出文本中实体与其他文本的边界和实体的类别。主要以下几个难点：

- 实体的数量是无穷的。不同的领域会有不同的实体，且实体并不是定量的，会不断的增加
- 实体的构词灵活。同一个实体可能会有多个名字，比如实体的简写；实体也可能是嵌套形成的
- 类别模糊。同一个实体在不同的语境下可能会有不同的类别

命名实体的识别，从某个角度来看可以视作一个序列标注问题。具体做法是将命名实体附着到 B, M, E, S 标签。其实，词性标注问题也可以以这样的方法来处理。更一般地看，词性标注与命名实体识别是同一个问题，它们都需要对文本进行分词，词性标注中的词性和 NER 中的实体类别是等价地。

2.1.2 语法分析

分析句子中的语法结构并将其表示为容易理解的结构（通常为树形结构）。

短语结构树 根据上下文无关文法将句子/短语分解为树状结构。短语结构语法（上下文无关文法）描述了如何自定向下地生成一个句子，同样句子/短语也可以通过短语结构语法进行分解。这其实和编译原理中的上下文无关文法是很类似的，也是通过很多产生式进行推导，那么是否可以使用编译原理中的词法分析来解决这个问题呢?

依存句法树 关注的是句子中词语之间的语法关联系，并将其约束为树形结构。在句子中，如果一个词语修饰另一个词语，则称修饰词为从属词，被修饰词为支配词，两者之间的语法关系称为依存关系，在可视化时，由支配词指向从属词。依存句法树描述了句子中的各个词之间的依赖关系，一般约定同一个词不能依存于多个词。

复合性原理：一个复杂表达式的意义是由其各组成部分的意义以及结合它们的规则决定的。通过将句子分解为短语、分解短语为单词，下游应用会得到更多更深层次的结构化信息。

2.1.3 序列标注

通常也可以看作是 token 级别的分类问题：对每一个 token 进行分类。token 级别的分类任务通常指的是为为文本中的每一个 token 预测一个标签结果，如命名实体识、词性标注等。

- NER (Named-entity recognition 名词-实体识别) 分辨出文本中的名词和实体 (person 人名, organization 组织机构名, location 地点名...).
- POS (Part-of-speech tagging 词性标注) 根据语法对 token 进行词性标注 (noun 名词, verb 动词, adjective 形容词...)
- Chunk (Chunking 短语组块) 将同一个短语的 tokens 组块放在一起。

2.1.4 问答系统

问答式信息检索是一种允许用户以自然语言方式询问，系统从单语或多语文档集中查找并返回确切答案或者蕴含答案文本片断的新型信息检索的方式。问答系统分类如 Fig.14所示。

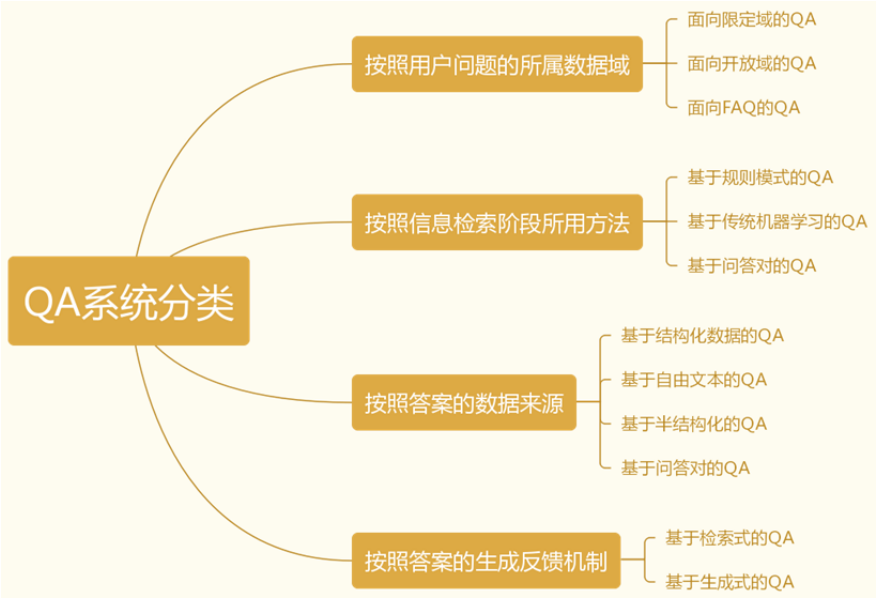


图 14: 问答系统分类，源图出处

2.2 统计自然语言处理

2.2.1 马尔科夫链/隐马尔科夫链

马尔可夫模型主要用于研究时间序列的分布的，若已有一个是时间序列：$X_0, X_1, X_2, ..., X_n$，马尔可夫模型要解决的就是这些随机变量的取值是如何随着时间而变化的、每个随机变量取值的概率 — 这些随

机变量取值的分布。随机变量的取值可以称为状态。那么问题就成了：

$$P(X_0 = s_0, X_1 = s_2, ..., X_n = s_m) =?$$

或者说，下一个时刻的随机变量的取值问题：

$$P(X_n|X_0 = s_0, X_1 = s_2, ..., X_{n-1} = s_{m-1}) =?$$

为了简化这个问题，有了马尔科夫假设。

(1 阶）马尔可夫假设：当前状态的取值只取决于前一个时刻的状态。即：

$$P(X_n|X_0 = s_0, X_1 = s_2, ..., X_{n-1} = s_{m-1}) = P(X_n|X_{n-1} = s_{m-1})$$

满足这样性质的一系列随机变量串联在一起就是马尔科夫链了。

那隐马尔科夫链与马尔科夫链又有什么关系呢？隐马尔科夫模型描述的是两个时序序列的联合分布：$p(\boldsymbol{X}, \boldsymbol{Y})$，其中 $\boldsymbol{X}, \boldsymbol{Y}$ 均为时序序列，通常称 $\boldsymbol{X}$ 为观测序列，$\boldsymbol{Y}$ 为状态序列 — 不可观测序列。其中观测序列/状态序列均可视为随机变量序列（注意，同一时刻的观测变量和状态变量并不是独立的），观测变量序列的取值为可观测值，状态变量序列的取值为状态值 — 与马尔可夫模型中的状态含义一致。隐马尔科夫模型的假设：

- 当前状态变量 $\boldsymbol{Y}_t$ 的取值仅以来与前一个状态变量 $\boldsymbol{Y}_{t-1}$ 的取值有关，连续多个状态变量的取值则构成隐马尔科夫链
- 任意时刻的观测变量 $\boldsymbol{X}_t$ 取值仅依赖于该时刻的状态变量的取值 $\boldsymbol{Y}_t$

一个隐马尔科夫模型可以表示为：$\lambda = (\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$，含义如下：

- $\boldsymbol{\lambda}$：初始状态概率向量，即初始时刻的状态变量 $\boldsymbol{Y}_0$ 取值为各个状态的概率
- $\boldsymbol{A}$：状态转移概率矩阵，即上一时刻的状态变量取某值时，当前时刻状态变量取值的概率分布：$p(\boldsymbol{Y}_t = s_j|\boldsymbol{Y}_{t-1} = s_i)$
- $\boldsymbol{B}$：发射概率矩阵，即当前状态变量取某值时观测变量取某观测值的概率分布：$p(\boldsymbol{X}_t = o_j|\boldsymbol{Y}_t = s_j)$

其实，可以看出马尔可夫模型是隐马尔科夫模型的一个特例 — 状态变量序列与观测变量序列相同，状态值即为观测值，每个状态只对应一个观测值即本身。

隐马尔可夫模型的三个应用：

- 样本生成问题：给定模型 $\lambda = (\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$，生成满足模型约束的样本 — 观测序列即对应的状态序列
- 模型的训练：给定训练数据 — 观测序列即对应的状态序列，估计模型参数 $\lambda = (\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$
- 序列预测：已知模型参数 $\lambda = (\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$，给定观测序列，求状态序列

怎么解决上述的三个问题呢?

对于样本生成问题，其实很简单，指要逐步采样状态，得到一个状态序列，在根据每一步的状态取值采样得到观察值，就可以得到观测序列，样本生成就完成了。

对于模型的训练问题，需要估计的参数为：$(\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$，对于 $\boldsymbol{\lambda}$ 则统计所有的状态序列中，计算以每个状态为开头的序列的频率即可，其他两个概率矩阵的训练类比即可。

对于序列预测问题可以通过维特比算法解决。给定一个观测序列，求解最有可能的状态序列。本质上这是一个搜索问题，搜索最有可能的状态序列，使观测序列的似然概率最大。简要地说一下。

维特比算法通过动态规划的方法解决这个问题。假设我们已经有最优的状态路劲，那么其中一条从起点开始的子路径也是最优的子路径。因此可以通过维护两个动态规划的矩阵来记录路径的选择和其概率。假设有两个矩阵 $\boldsymbol{\sigma}, \boldsymbol{\psi}$。其中 $\boldsymbol{\sigma}_{ti}$ 表示在时刻 t 时以 s_i 结尾的所有局部路径的最大概率，$\boldsymbol{\psi}_{ti}$ 表示在时刻 t 时末状态为 s_i 的前驱状态。

应用 使用隐马尔科夫模型可以解中文分词问题。给定训练数据 — 每个样本为一个句子，对于每个样本，目标是给每个词分配一个标签（如 B, M, E, S），然后可以根据序列对句子进行分词。为达成这个目的，需要训练隐马尔科夫模型，获得模型的各个参数，根据训练数据是否被标注可以使用不同的方法进行训练。获得训练数据后即可使用模型进行序列预测任务，再根据序列进行分词。

注：本节内容主要参考何晗所著《自然语言处理入门》。

2.2.2 概率图模型

将联合概率分布以图的形式来表示，可以分为有向概率图模型和无向概率图模型。概率图模型中，将随机变量表示为结点，相关联的随机变量之间会有边连接。

有向图模型中按变量的因果关系进行连接，可以刻画变量之间的因果关系。有向图表示的联合概率可以根据图中的变量依赖关系分解成多个条件概率的积。(听着有点像拓扑图)

无向图模型没有体现出因果关系，强调的是变量之间的相互关系。无向图模型将联合概率分布分解为“无向图中-最大团上-随机变量的函数-的乘积”。

2.2.3 TF-IDF

词频-倒排文档频次。通常用于衡量一个词语在文档中的重要程度。单单使用词频评价词语重要程度是不全面的，有些词可能会在文档中出现很多次，但是如果其在很多文档中都出现，却又体现不了其重要性。因此，需要一个对词频进行扩充，希望得到的重要的词应该是这样的：词频高，同时又不是出现在大部分文档中。

$$TF - IDF(t, d) = \frac{TF(t, d)}{DF(t)} = TF(t, d) \cdot IDF(t)$$

其中，$TF(t, d)$ 表示词 t 在文档 d 中出现的次数，$DF(t)$ 表示包含词 t 的文档数。实际中计算 $TF - ID$ 时会加入一些平滑操作（如加一平滑、对 IDF 取对数）防止结果下溢等。IDF 表示 inverse document-frequency，计算方法：

$$IDF(t) = \log \frac{1 + n}{1 + DF(t)}$$

或

$$IDF(t) = \log \frac{n}{1 + DF(t)} + 1$$

计算得到一个矩阵，每一行表示一个文档的 tf-idf 向量，每个元素表示对应的词（term）在该文档种的重要程度，一般还会进行 $l2$ 正则化。

2.2.4 词袋模型

用于表示文档/句子（其实句子可以看作只有一个句子的文档）的一种方法。词袋模型一般会先构建一个词表，每个文档经过分词后，可以使用不同的统计量来构建文档/句子的向量，词表之外的词不予考虑。可以选用的不同的统计量有：

- 词频
- 布尔词频
- TF-IDF
- 词向量

2.2.5 使用朴素贝叶斯分类器进行文本分类

首先，给定类别数为 c，每个文档的特征向量长度为 n。

朴素贝叶斯分类器是一个生成式的分类器，其计算的是 $p(Y|X)$，对于生成模型，重要的是要学习联合分布 $p(X,Y)$，朴素贝叶斯通过训练数据学习该联合分布：$p(X,Y) = p(Y)p(X|Y)$。对于给定样本 X^i，对其进行分类可形式化为：

$$y = \underset{c_k}{argmax}\, p(Y = c_k|X = X^i) = \underset{c_k}{argmax}\, \frac{p(X = X^i|Y = c_k)p(Y = c_k)}{p(X = X^i)}$$

朴素贝叶斯中假设样本的特征之间是相互独立的，即 $p(X) = p(X_1 = x_1, ..., X_n = x_n) = p(X_1 = x_1)...p(X_n = x_n)$。那么为了得到样本 X^i 的类别，只需要计算出 $p(X = X^i|Y = c_k), p(Y = c_k), p(X = X^i)$ 即可。而对于某一个样本的分类，$p(X^i)$ 对分类是没有帮助的，在 Y 取不同类别时是不会变化的，故可以不计算 $p(X^i)$。那么剩下的需要计算的只有出 $p(X = X^i|Y = c_k), p(Y = c_k)$ 了。

$p(Y = c_k)$ 是很好计算的，只需要统计训练数据中类别为 c_k 的样本所占的比例即可。由于特征之间独立的假设，所以 $p(X^i|Y = c_k) = \prod_{j=1}^{n} p(X_j^i = x_j|Y = c_k)$。这个概率也只需要从训练数据中统计即可，统计每个类别下，特征向量的第 j 维取 x_j 的比例即可。

关于样本的特征向量：通常可以采用词袋模型的方法来表示一个文本/句子。

2.2.6 预训练

预训练在很多领域上都有应用，如 CV、NLP。当用深度学习来解决一个任务时，使用的深度模型通常会包含很多参数，我们需要使用大量的数据来更新我们的参数使之能够在特定的任务上取得不错的效果。一般，我们会对模型的参数初始化，然后使用大量带标签的数据来更新模型参数。

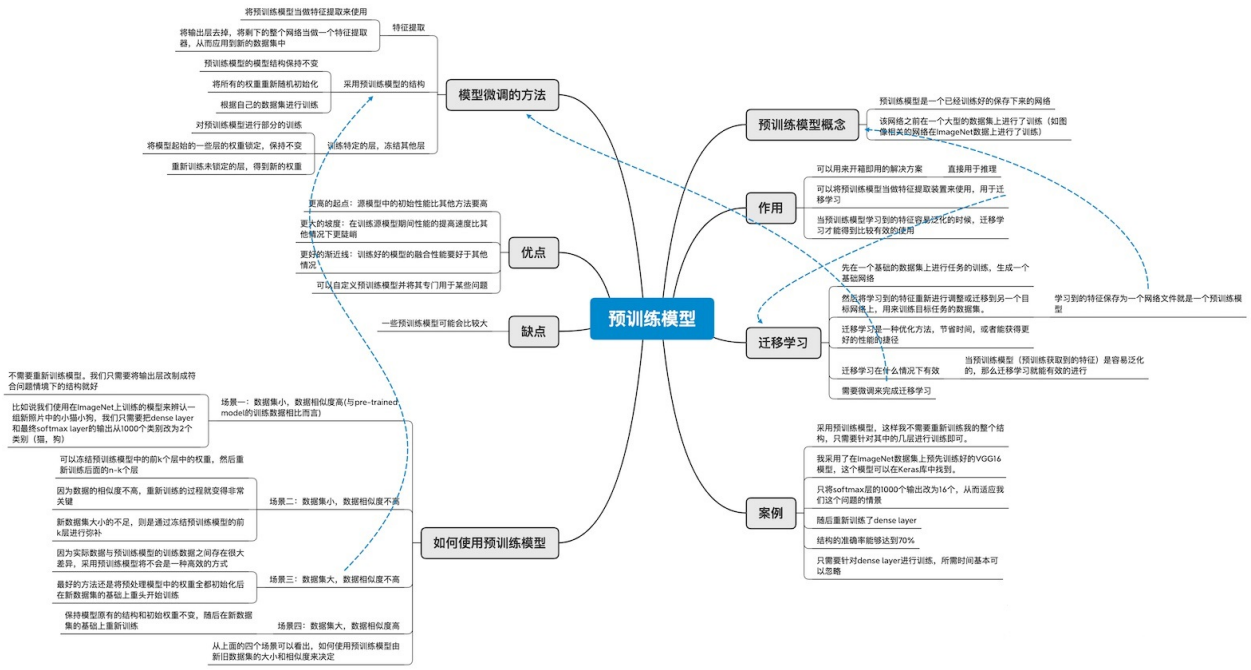


图 15: 预训练模型（出处）

但实际情况是，带标签的数据总是很少的。

预训练通常在大量无监督的数据上学习（可以是无监督的学习任务、简单的有监督学习任务），用无监督的数据帮助模型的参数收敛，后续再在监督数据上进行调整。

2.3 Deep Model for NLP

2.3.1 Transformer

在 NLP 中有很多输入为序列输出为序列的 **seq2seq** 任务，例如机器翻译、文本摘要、序列标注、文本生成等等。早期的 seq2seq 任务 RNN-based[18, 5] 模型为主，但 RNN-based 的模型容易出现以下问问题：

- RNN 处理 seq2seq 任务时通常将输入序列转换成一个 context 向量，再基于 context 来得到输出的序列。单个向量难以包含输入序列的所有信息
- RNN 在递归的解码过程使得其难以处理长文本序列
- RNN 的反向传播容易出现梯度弥散/爆炸问题
- 序列的处理是串行的，难以并行化，计算效率低

Attention[1, 14] 技术 — 帮助 RNN-based 模型解决了一部分困难。具体是怎么做的呢? 在 RNN-based 模型的基础上主要有以下几个变化：

- 编码器传给解码器的不再只是一个 context 向量，会利用编码过程中的所有隐向量
- 解码时，会根据所处的时间步，为每一个隐向量计算一个分数，归一化后再分别与隐向量相乘得到当前时间步的 context 向量

有了 Attention，RNN-based 模型的一部分缺点得到了解决（处理长文本序列的问题），但还是有一些问题：不能并行化处理输入序列，且能不能直接在 Attention 上构建 seq2seq 模型呢?

Transformer[19] 来了!!! Transformer 采用的也是编解码的结构：先对输入序列进行编码，再根据编码的输出进行解码。与 RNN-based 模型不同的是，其不是递归的处理输入，Transformer 可以并行地处理整个文本序列。

Transformer 的模型结构如 Fig.16所示。具体的计算过程可以参考论文 [**?**] 的论文笔记。

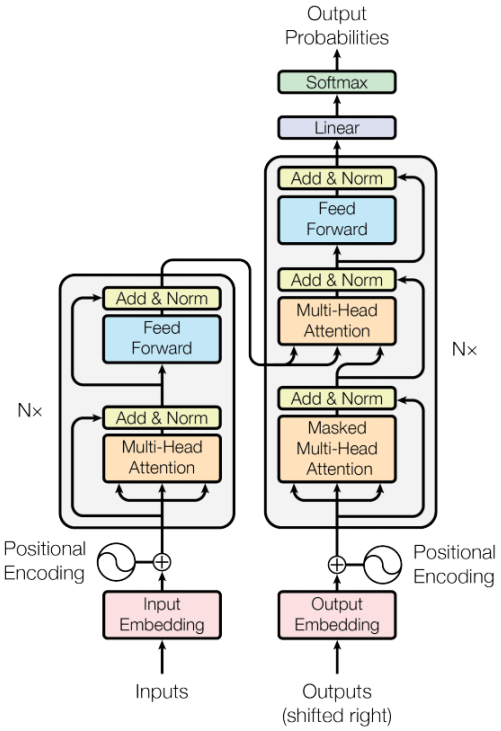


图 16: Architecture of Transformer

- 由于 Transformer 没有循环结构，但是加入了 Positional Encoding
- 编码部分最终的输出。会作为 Decoder Block 中 Attention 的 K, V，解码器的输入作为 Q
- Decoder Block 中的 Masked Multi-head Attention，处理当前输入时只允许看到当前位置之前的输入
- 在编码器中，也有 mask。因为通常对输入进行 padding

2.3.2 一些关于 Transformer 的问题

1. 为什么输入 X 要经过变换得到 Q, K, V？为什么不直接使用 X？

如果直接使用 X，则 X 直接承担了三种角色：查询、键和值。难以学习到满足要求的 X。经过变换后得到的 Q, K, V，Q 负责表示查询的问题，V 表示输入的信息，K 表示与输入相关的 “关键词”，Q, K 相乘用于衡量查询的问题与各条信息之间的相似性。

2.3.3 BERT

Motivation BERT（Bidirectional Encoder Representations from Transformers）[7] 缘自 Transformer 模型。通常的语言模型都是单向的，例如从左到右，这使得在处理每个 token 时只能利用其之前的 token。单向的模型处理 token-level 的任务时，难以应用到上下文的信息。

HOW? BERT 在大量无标签的文本数据集进行预训练，得到模型的参数后，再添加具体任务相关的神经网络层，微调后即可得到适用于具体任务的模型，即：pre-training + fine-tuning。

Pre-trainingBERT 使用两个预训练任务。

Fine-tuning

title

2.4 词嵌入模型

2.4.1 Word2Vec

2.4.2 ELOM

3 Graphs

3.0.1 图基本概念

Motif 形式上是一个连通子图，强调这个 Motif 的拓扑结构。Motif 可能是某种经常出现在图中的结构，可以认为时一种反复出现的结构模式。相同结点数的 motif 可能会有多种结构。

Weisfeiler-Lehman isomorphism test 图同构测试。对于两个图 G_1, G_2，WL test 能够判断它们是否同构，但是也存在一些特殊的例子，WL test 会将非同构的两个图判定为同构的。算法过程如下：

1. 对 G_1, G_2 中的结点都赋予一个标签，都为 1；
2. 对于每个结点 v 重复执行：
   (a) 汇集 v 的邻居结点的标签，放在一个列表里，$S_v = \{u_l | u \in \mathcal{N}_v\}$
   (b) 将 v 的标签 v_l 和 S 组合成一个 pair—(v_l, S_v)
3. 利用一个 hash 函数将每个结点的 (v_l, S_v) 映射成一个新的标签，将新的标签赋予给对应的结点。至此一轮迭代就结束了，如果图中结点的标签都没发生变化则结束，否则返回步骤 2.

上述过程是针对一个图来说的，可以同时分别对 G_1, G_2 执行上述算法。当迭代结束后怎么判定两个图是否同构呢？WL test 中是这样做的：
分别计算图中每个标签的数量（也就是标签的分布），可能会得到类似这样的一个结果 (1: 2, 2:3, ...)，这表示标签 1 出现了 2 次，标签 2 出现了 3 次。如果最后两个图这个结果是一致的，那么不排除它们同构的可能性。

基于 WL test 算法，WL subtree kernel[16] 构建以某个结点为根的子树，来代表该结点的特征。

图算法（GNN）中常用的 benchmar 以及数据集 常用的图数据集主要可以分为这几类：

- 生物信息类的图数据。如 MUTAG, PTC, NCI1,PROTEINS 分子，药物的一些图数据
- 社交网络。如 Facebook, Reddit 等
- 引文网络。如 Cora, PubMed, Citeseer 等

斯坦福大学的 SNAP 组有很丰富的图网络数据集：Stanford Large Network Dataset Collection

Heterogeneous graph, Attributed graph 同构图和属性图，同构图中，有多种类型的结点和边。Attributed graph 中结点会有较丰富的属性（个人感觉可以理解为特征）。一个简单的分类：

Graph Types	Homogeneous	Multi-Relational	Heterogeneous
#of node types	1	1	>1
#of edge types	1	>1	>=1

Graph Kernel

Dynamic graph, Streaming graph 动态图和流图。动态图主要指在给定的（static）的结点集上，随着时间，边发生变化；而流图则是结点集和边都会随着时间而变化。

除了结点和边的变化，可能结点/边的标签也会发生变化，更具体的，结点/边的属性也会变化。

图的同构 对于两个图 $\mathcal{G}_1 = (V_1, G_2), \mathcal{G}_2 = (V_2\ E_2)$，如果两图之间的结点能够一一对应，令该映射为 f，

若 $\forall v_i^1, v_j^1 \in V_1, v_i^2, v_j^2 \in V_2$，$f(v_i^1) = v_i^2, f(v_j^1) = v_i^2$，当 $(v_i^1, v_j^1) \in E_1$ 有 $(f(v_i^2), f(v_j^2)) \in E_2$，则称 $\mathcal{G}_1, \mathcal{G}_2$ 是同构的，存在同构映射 f。

3.0.2 结点邻居采样

在图机器学习中，生成结点表征可以说是最基本的事儿了。不管是随机游走、矩阵分解还是图神经网络的方式，最终都会得到结点的表征。其中不可绕过的一个就是结点的邻居。在随机游走中，会类似于语言模型中词语的上下文环境一样为结点构造上下文，每个结点产生的随机游走路径就是一个上下文。游走的过程可以看作是对结点邻居进行采样的过程。矩阵分解中，通常是构造图的 Laplacian 矩阵，再进行分解，用特征向量来表示结点表征，对这个不是很了解，就不多说了。到了图神经网络中，节点邻

居更是成了主角。从 GCN 原论文 [12] ==> GraphSage[9] ==> FastGCN[3] ==> VR-GCN[2] ==> ... 。GCN 中结点的邻居采样方式五花八门。

1. GCN 原论文中，再进行图卷积时需要把整个 graph 全部加载进内存，为了求一个结点的表征需要对整个 graph 进行卷积
2. GraphSage 中，并不使用结点的所有邻居，而是对每个结点的邻居进行采样，得到固定数目的采样后的邻居
3. FastGCN 中引入了 importance sampling
4. 审稿时发现的根据 attention distribution 和 uniform distribution 的 KL 散度选 top-k 的采样。这种方法可以发现比较重要的结点，而不是均匀地参与到其邻居的结点

3.0.3 DeepWalk & Node2Vec

二者都是学习结点表征的方法，都是通过游走的方式来生成一个节点的 context，在通过 SkipGram 的方式来学习结点表征。二者关键的差别就在生成 context 的方式，即游走方式上。

- DeepWalk 不适用于有权图，不能学习权重对 graph、结点带来的影响；Node2Vec 能够处理带权图
- DeepWalk 中使用的是 RandomWalk。给定一个结点 v，下一步访问的结点通过均匀采样 v 的邻居结点来得到
- Node2Vec 中用了两个参数来控制游走，即如何采样下一个结点

重点介绍以下 Node2Vec 中的 randomwalk，如 Fig.3.0.3所示。Node2vec 中的随机游走是二阶的随机

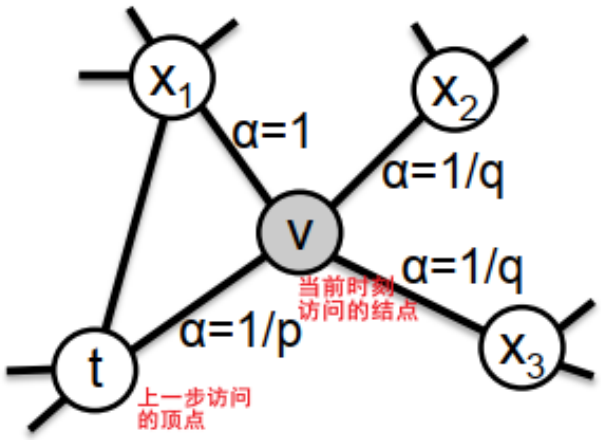


图 17: Node2Vec 中的随机游走

游走，为什么叫二阶的呢？在 Deepwalk 中，随机游走只与当前时刻访问的结点有关，直接从当前节点

的邻居中进行均匀采样，这叫一阶。Node2vec 中的二阶是指：当前访问的结点 v，下一步访问的某个节点概率将由上一时刻访问的结点 t 和 v 与邻居的边的权重决定，这是一个二阶马尔科夫链。

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v,x) \in E \\ 0 & \text{otherwise} \end{cases}$$

其中 $P(c_i = x \mid c_{i-1} = v)$ 表示当前时刻访问 v，下一时刻访问 x 的概率，Z 用于归一化，$\pi_{vx} = \alpha_{pq}(t,x) \cdot w_{vx}$，$\alpha_{pq}(t,x)$ 为：

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

可见 $\alpha_{pq}(t,x)$ 是一个由 p,q 参数化的函数，函数的自由变量是 t,x 即上一时刻访问的结点和下一时刻访问的结点，其中 $d_{tx} \in \{0,1,2\}$ 表示 t,x 间的最短距离。

p,q 就是控制游走的两个参数：

- Return Parameter，p，该参数控制下一时刻访问 t 的可能性，如 Fig.3.0.3所示，p 越小，越有可能访问上一时刻已经访问过的结点 t，当然还要考虑 w_{vt}
- In-Out Parameter，q，该参数能够控制是 BFS 还是 DFS。当 $q > 1$ 时，在 p 不变的情况下，访问 $d_{tx} = 1$ 的结点的概率会增大，这就是 BFS，当 $q < 1$ 时，则会增大访问 $d_{tx} = 2$ 的结点的概率会增大，这就是 DFS

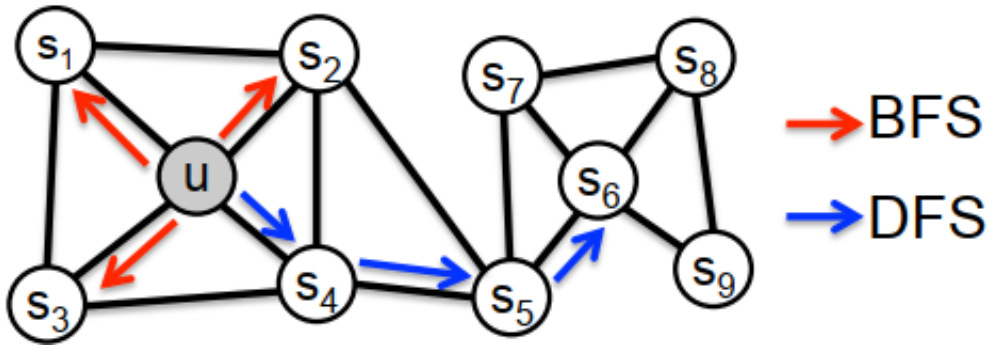


图 18: Homophily 与 Structure Equivalence

论文中将这叫做 Biased random walk，通过调整游走的参数 p,q 能够使学习到的表征在同质性（Homophily）和结构相似性（Structure Equivalence）间平衡。BFS 能够探索探索 homophily，即距离相近的结点，如 Fig.3.0.3所示，s_1, s_2, s_3, s_4 是与 u 相邻的结点，它们的表征应该相似；DFS 使得探索到更远的地方，能够发现与 u 结构类似地结点，如 s_6，它俩地表征也应该相似。所以学习到的表征既能体现结点地同质性，也能体现结点地结构相似性。

4 Knowledge Graph

4.0.1 KG 中的文本信息抽取

从纯文本数据中获取有价值的数据，组织成结构化的形式或半结构化的形式。主要包含这几个过程：实体识别、实体消歧、关系抽取及事件抽取。

实体识别 从文本中识别出实体信息。

实体消歧 消除指定实体的歧义。可以分为两类：

- 实体链接：将给定文本中的实体指称项链接到已有知识图谱中的某个实体上
- 实体聚类：假设已有知识图谱中并没有已经确定的实体，在一个给定一个语料库的基础上，通过聚类的方法消除语料中所有同一实体指称项的歧义，具有相同所指的实体指称项应被聚为一类

关系抽取 获取两个实体之间的语义关系。

事件抽取 从描述事件信息的文本中抽取出用户感兴趣的事件信息并以结构化的形式呈现。**kg 中的事件能否看作一个 subgraph ?**

5 Recommender System

5.1 基本概念

5.1.1 历史背景

Motivation 互联网的发展，人们接受的信息越来越多，从信息稀缺时代逐渐过渡到了信息爆炸时代。面对数据的海洋，我们越来越希望我们感兴趣的信息能够直接呈现在我们面前。**把用户想要的信息推荐给用户** — 推荐系统的宗旨。

发展过程 ：

- 1994 年，明尼苏达大学 GroupLens 研究组推出第一个自动化推荐系统 GroupLens，提出将协同过滤作为推荐系统的重要技术
- 1995 年，卡耐基梅隆大学的 Robert Armstrong 等人提出个性化导航系统 Web Watcher；斯坦福大学的 Marko Balabanovic 等人退出了个性化推荐系统 LIRA
- 1997 年，Resnick 等人首次提出 Recommender System 一词
- 1998 年，Amazon 上线了基于物品的协同过滤算法，并在千万级用户和百万计商品的规模上进行了应用。Amazon 于 2003 年发表论文 [13] 公布了基于物品的协同过滤算法
- 2001 年，IBM 在其电子商务平台 Websphere 中增加个性化功能

- 2003 年，Google 开创 AdWords 盈利模式，通过用户的广告词来提供相关的广告。2007 年 Google 为 AdWords 添加个性化元素，通过对用户一段时间内的搜索历史进行记录和分析，以便更精准呈现广告
- 2006 年，Netflix 宣布一项竞赛，任何人只要能将其现有的电影推荐算法 Cinematch 的预测准确度提高 10% 就能获得 100 万美金
- 2007 年，Yahoo 提出 SmartAds 广告方案，通过分析用户信息以及用户搜索、浏览行为为用户呈现个性化广告
- 2007 年，第一届 ACM 推荐系统大会举行
- 2015 年，Facebook 在其官网公布了其推荐系统原理、性能及使用情况（Recommending items to more than a billion people），相关论文 [11]
- 2016 年，YouTube 发表论文 [6] 介绍 Youtube 如何向用户推荐个性化的视频
- 2016 年。，Google 发表论文 [4] 介绍 App 商店中的推荐系统，即 Wide&Deep 模型

形式上来看，推荐系统的任务是这样的：给定一个输入，从数据集中搜索出一系列数据对象，并排序后输出。

输入通常是关于某个用户的表示，可以是该用户的特征、向量化的表征等，除此之外用户的行为数据，如用户的浏览记录、搜索记录、与商品的交互数据、用户的一些动态变化的数据等都可以与用户特征一起作为输入，而待搜索的数据集中通常是商品的数据，如商品的特征等，基于以上丰富的信息，将输入的信息与数据集中对象进行匹配，给出推荐的顺序。

乍一看，这和搜索很像。其实这么说也没错，都是一个**Learning To Rank** 的任务。其实很多领域研究的问题都很相似，通过进一步的抽象可以看作是同一个问题。但随着研究的深入，为了在一个问题上取得更好的效果，会相应地结合该领域地特点，如领域内特有的信息、领域内特有的数据形式等。因此，虽然问题之间有重叠，但为了做得更好，除了基础的方法，还需要更深入地挖掘领域地特点！例如，在推荐系统中，用户的需求和兴趣是隐含的（隐含在历史数据中，可能用户都不知道自己喜欢什么），而搜索中搜索语句是显式的（用户需求是明确的）。

推荐系统主要元素

- 物品集合：被推荐的物品或内容
- 用户：用户的基本信息，如基本信息、行为信息、兴趣爱好等
- 场景：用户所处的环境，如网络环境、所处位置等
- 推荐引擎：根据用户对物品的偏好与用户的画像数据进行拟合，学习什么样的用户会喜欢什么样的物品。引擎包含以下重要模块：
  - 召回模块：根据用户和场景特征，从整个物品数据集（上百万物品）中挑选用户可能感兴趣的物品，**挑选出一个较小的候选集**（几百至几千）。召回模块中，通常使用简单的特征进

行快速查询，比如用户最近点击的物品的相似物品、根据用户兴趣召回物品等。常用的算法：Word2vec、LDA、LSTM、ItemCF、UserCF、DNN 等

- 排序模块：针对召回模块找到的**候选集进行精排**，得到用户对候选物品集的评分。常用的算法：LR、FM、XGBoost、GBDT+LR、Wide&Deep、FNN、PNN、DeepFM、NFM、DIN 等
- 后排模块：得到用户对候选集的评分后，可以根据一些规则对排序进行调整，如运营干预、优先级调权等

- 推荐结果集：推荐结果，或推荐结果的有序排列

5.1.2 共现矩阵

显然，共现矩阵是一种矩阵，**关键是它描述了一种什么事实**？$M, N(|M| = m, |N| = 2)$ 表示两个相同或者不同的集合，共现矩阵可以用于表达两个集合笛卡尔乘积的元素对的某种关系，(m_i, n_j) 间的这种关系可以用共现矩阵的元素 CM_{ij} 表示。

在具体的应用场景中，

- 推荐系统：M 表示用户集，N 表示商品集，CM_{ij} 表示用户 m_i 对 n_j 的喜爱程度（如评分）、是否点赞、是否分享等
- 自然语言处理：$M = N$ 表示词汇表，CM_{ij} 表示词 m_i 与词 m_j 出现在同一个句子中的次数
- 社交网络：$M = N$ 表示作者集合，CM_{ij} 表示 m_i 与 m_j 间存在某种关系，如朋友关系、合作关系等

共现矩阵是两个集合的元素间关系的一个很直白的表达。正是因为直白，共现矩阵通常很稀疏，空间需求大。

5.1.3 召回方法分类

召回即从海量的数据集中找出尽可能包含真实结果的候选集。常见的分类：

- 行为相似召回：通过用户与物品的交互行为，发现行为指向的物品的相似物品
- 相似用户召回：通过用户画像和用户行为等，计算用户之间的相似性，根据相似用户的行为进行召回物品
- 内容相似召回：通过对物品内容进行分析，得到物品之间的相似性，根据与用户产生交互的物品来召回

5.1.4 常用相似度计算方法

同现相似度 物品 A 和物品 B 的同现相似度：

$$w_{A,B} = \frac{|N(A) \cap N(B)|}{|N(A)|}$$

其中，$A(A), N(B)$ 分别表示喜欢 A 和 B 的用户集合。但是这种相似度有个问题，当 B 是热门物品时，喜欢 A 的用户中可能绝大部分也会喜欢 B，那么 $w_{A,B}$ 就会接近于 1，即任何物品与热门物品的相似度都会接近于 1。除此之外，还有个问题，这个相似度不是对称的，即 $w_{A,B} \neg w_{B,A}$，这看起来是有点不合理的，因此，其改进如下：

$$w_{A,B} = \frac{|N(A) \cap N(B)|}{\sqrt{|N(A)| \cdot |N(B)|}}$$

欧几里得距离 众所周知，欧氏距离是欧氏空间中两点之间的距离。两个物品 A，B 的向量表示为 $\boldsymbol{V}_A, \boldsymbol{V}_B$，则欧式距离为：

$$d(A, B) = ||\boldsymbol{V}_A - \boldsymbol{V}_B||_2$$

以欧式距离计算 A，B 间相似性时可如下计算：

$$sim(A, B) = \frac{1}{1 + d(A, B)}$$

或许还有其他的形式，例如：

$$sim(A, B) = e^{-d(A,B)}$$

皮尔逊相关系数 介于-1 和 1 之间，它度量两个序列（可以是用户对物品的偏好/评分序列）之间的线性相关程度。它度量数字一起按比例改变的倾向性，也就是说两个数列中的数字存在一个大致的线性关系。当该倾向性强时，相关值趋于 1。当相关性很弱时，相关值趋于 0。在负相关的情况下一个序列的值很高而另一个序列的值低---相关趋势趋于-1。

使用皮尔逊相关系数计算用户/物品相似度时，通常取共现矩阵中的行（用户对各个物品的偏好）或列（各个用户对一个物品的偏好）作为数列来计算皮尔逊相关系数。两个物品/用户 A，B 的向量表示为 $\boldsymbol{x}, \boldsymbol{y}$，则皮尔逊相关系数为：

$$r_{\boldsymbol{xy}} = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2} \sqrt{\sum_i (y_i - \bar{y})^2}}$$

其实，仔细一看，皮尔逊相关系数和余弦相似度很像，对 $\boldsymbol{x}, \boldsymbol{y}$ 进行标准化后再进行余弦相似度计算。皮尔逊相关系数用于计算相似性时有以下问题：

- 没有考虑序列的长短。例如两个用户交互的物品的交集可能会比较大或较小，通常交集较大的情况下更可靠，但是交集更小时计算出的皮尔逊系数可能会更大
- 若序列长度为 1 时，根据定义则无法计算皮尔逊系数
- 若序列中的值都相同时也无法计算皮尔逊系数，因为方差为 0

余弦相似度 两个物品/用户 A，B 的向量表示为 $\boldsymbol{V}_A, \boldsymbol{V}_B$，则余弦相似度为：

$$sim(A, B) = \frac{\boldsymbol{V}_A \cdot \boldsymbol{V}_B}{||\boldsymbol{V}_A||_2 \times ||\boldsymbol{V}_B||_2}$$

Jaccard 系数 两个物品/用户 A，B 的向量表示为 $\boldsymbol{V}_A, \boldsymbol{V}_B$，则 Jaccard 相似度为：

$$sim(A, B) = \frac{\boldsymbol{V}_A \cdot \boldsymbol{V}_B}{||\boldsymbol{V}_A||_2 + ||\boldsymbol{V}_B||_2 - \boldsymbol{V}_A \cdot \boldsymbol{V}_B}$$

5.1.5 基于人口统计学的推荐

特征相似的人喜欢的东西应该也类似。根据**用户画像**（以用户的基本信息作为相似度计算的基础），找出与目标用户相似的用户，将相似用户喜欢的物品推荐给目标用户。这种方法需要构建用户画像。

优点

- 不涉及当前用户的历史喜好，所以解决了“**用户冷启动**”问题
- 不依赖于物品本身的数据，故而无论这个物品是“书籍”、“音乐”还是“短视频”都可以使用，即它是领域独立的

缺点

- 用户画像所需要的有些数据难以获取
- 以人口统计信息计算相似用户不可靠。特别是“书籍”“音乐”这种涉及到个人喜好的商品，单用这种方法更是难以达到很好的效果

5.1.6 基于内容的推荐

把用户可能喜欢的物品类型进行推荐，即要找到相似的物品。需要构建物品的特征，如对物品进行标签化。

优点

- 不存在稀疏性和“**项目冷启动**”问题
- 简单有效，推荐结果具有可解释性，不需要领域知识
- 基于物品本身特征推荐，不存在过度推荐热门的问题
- **解决了基于人口统计学对个人兴趣建模的缺失**，能够很好的建模用户的喜好，实现更精确的推荐

缺点

- 推荐的结果**没有新颖性**
- 由于需要基于用户的兴趣偏好进行推荐，故而存在“**用户冷启动**”问题
- 该方法受推荐对象特征提取能力的限制。由于是根据物品相似度进行推荐，故而，物品特征构建模型的完善和全面决定了最后推荐的质量，然而像图像、音频等这种类型的特征难以提取

5.1.7 基于协同过滤的推荐

Collaborative Filtering，通过群体的行为来寻找相似性（用户或物品的相似性），通过该相似性来做推荐。协同过滤算法可以分为以下几类：

基于用户的协同过滤 User-based CF（UserCF），根据用户对物品的偏好，发现与当前用户口味和偏好相似的“邻居”用户群，并推荐近邻所偏好的物品。与基于人口统计学的推荐的联系与区别：

- 都是基于相似用户来推荐
- 不同之处在于如何计算用户相似度：基于人口统计学只考虑用户本身的特征，而 UserCF 是在用户的历史偏好的数据上计算用户的相似度，它的基本假设是，喜欢类似物品的用户可能有相同或者相似的偏好

特点：

- 适用于用户数较小的场景
- 适用于时效性较强（即物品变化频繁），用户个性化兴趣不太明显的领域，如新闻推荐
- 在新用户对很少的物品产生行为后，不能立即对它进行个性化推荐，因为用户相似度表示每隔一段时间离线计算的，故而存在“用户冷启动”问题
- 新物品上线后一段时间，一旦有用户对物品产生行为，就可以将新物品推荐给和对它产生行为的用户兴趣相似的其他用户，故而解决了“项目冷启动”问题
- 解释性较差，因为计算得到的相似用户可能并不是真的相似，并不能真的反映用户的兴趣。；例如，用户都买了卫生纸、水杯等日常用品并不能表示用户之间相似，但如果都买了键盘、屏幕等，则能较可靠的推断用户是相似的，因此在计算用户相似度时要注意这种大家都会关注的“热门物品”，大家都选择热门物品并不能体现用户的真实兴趣（可以依据社交网络来防止相似用户并不相似）

基于物品的协同过滤 Item-based CF（ItemCF），基于用户对物品的偏好，发现物品和物品之间的相似度，然后根据用户的历史偏好信息，将类似的物品推荐给用户。与基于内容的推荐的联系与区别：

- 都是基于相似物品进行推荐
- 不同之处在于如何计算物品相似度：ItemCF 是根据用户历史的偏好（如共现矩阵）推断，而基于内容的推荐是基于物品本身的属性特征

特点：

- 适用于物品数明显小于用户数的场合，如果物品很多，计算物品的相似度矩阵的代价就会很大
- 适合于长尾物品丰富，用户个性化需求强烈的领域，如电商网站
- 用户有新行为，一定会导致推荐结果的实时变化
- 新用户只要对一个物品产生行为，就可以给它推荐和该物品相关的其它物品，故而解决了“用户冷启动”问题
- 不能在不离线更新物品相似度的情况下将新的物品推荐给用户，故而存在“项目冷启动”问题
- 有较强的解释性，因为是依据用户历史偏好的物品来推荐

基于模型的协同过滤 Model-based CF（ModelCF）。基本思想：用户具有一定的特征，决定着他的偏好选择；物品具有一定的特征，影响着用户需是否选择它；用户之所以选择某一个商品，是因为用户特征与物品特征相互匹配。ModelCF 基于样本的用户偏好信息，训练一个推荐模型，然后根据实时的用户喜好的信息进行预测，计算推荐。

参考：基于协同过滤的推荐算法、推荐系统——经典算法（基于内容、协同过滤、混合等）。

5.1.8 推荐中要注意的点

- 长尾效应：位于长尾位置的曝光率低的项目产生的利润不低于只销售曝光率高的项目的利润。注意对长尾物品的推荐效果，热门物品的推荐较长尾物品更容易，在评价算法效果时要注意对长尾物品的推荐效果
- 用户经常购买的物品并不一定能体现用户的偏好，如很多人都会经常买卫生纸，但这并不能体现用户很喜欢卫生纸，卫生纸作为热门物品反而并不能体现用户的偏好。不同的物品对用户的偏好的贡献是不一样的 — **user-aware**
- 使用用户行为数据来预测某个用户对某个物品的评分时，要注意历史行为中的物品起的作用可能是不同的 [10]。如预测我会不会买 iphone 时，可能与我过去买的是什么手机有关 — **target item-aware**
- 如何对待用户没有交互过的物品?

5.1.9 常用指标

此处介绍的指标不涉及具体的计算公式，在不同的场景下有不同的具体定义，计算自然也相应而变。

覆盖率 覆盖率用来描述一个推荐系统对长尾内容或商品的发掘能力。关于覆盖率的定义，最简单的理解是推荐系统能够推荐出来的物品，占平台中全部物品的比例。

多样性 用户的兴趣是非常广泛的，在一个视频应用中，用户可能既喜欢看烧脑电影，也喜欢看动作大片。那么，为了满足用户广泛的兴趣，推荐列表需要能够覆盖用户不同的兴趣领域，即推荐结果需要具有多样性。想提升推荐系统的多样性，就需要在较大的时间跨度上去识别和理解用户的兴趣。

新颖性 新颖，指给用户推荐那些他们以前没有听说过的内容或商品，例如在视频应用中应该尽可能多地向用户推荐他们没有看过的电影。而考虑到很多用户在某个应用中的使用粘性可能并不高，例如一个用户可能同时是多个视频应用的用户，所以仅仅依靠用户在自己系统中的行为记录来保证推荐的新颖性是不够的。

除此之外比较简单方法是基于内容或商品的平均流行度去进行推荐，因为越不热门的东西越可能让用户觉得新颖。

不过，向用户推荐不流行的内容或商品，其实是牺牲了一定的推荐精度的，所以我们需要权衡该指标与其它指标之间的平衡——这不仅在于技术层面的考量，可能也在于商业层面的考量。

惊喜度 如果推荐结果和用户的历史兴趣不相似，但却能够让用户觉得满意，那么就可以说推荐结果的惊喜度很高。想要兼顾推荐系统的惊喜度并不是一件容易的事情，因为这意味着需要降低推荐结果和用户历史兴趣的相似度，所以可能会对预测准确度带来一定的挑战。

但毫无疑问，用户需要惊喜，这会极大提升用户的满意度和使用体验，所以推荐系统对惊喜度的追求只会不断提高，且还需要在不影响预测准确度的前提下来实现。过于关注准确率会导致推荐结果的惊喜度不高。

5.2 经典算法

5.2.1 FM

Factorization Machine，因子分解机。

5.2.2 FFM

Field-aware Factorization Machine，

6 Learning to Rank

6.1 排序学习算法简介

排序学习，Learning to Rank，简言之：给定一个查询 q，和待查询的数据集合 D，返回与 q 最相关的 top k（有序）的结果或者给出 D 中所有元素与 q 的相关性排序。

6.1.1 Pointwise

将排序问题转化为分类、回归问题。以一个 q 和一个待查询对象 q（只有一个待查询对象，这也是 Point 的原因）之间的相关性作为预测目标，即 $Pointwise : q \times d \rightarrow label, q \in Q, d \in D$，以 (q, d) 作为训练样本。

Pointwise 缺点：

- 只考虑了某个查询下，单个待查询对象的相关性，没有考虑该查询下所有待查询对象的排序关系，即带查询对象间的关系
- 查询时，往往排在前面的对象比后面的对象更重要，而 Pointwise 平等地对待所有带查询对象，损失函数会被占多数的非 top K 对象所影响

6.1.2 Pairwise

针对一个查询 q，通过某种方法确定一个大致的待查询对象集合 D_q，Pairwise 将排序问题转化为 D_q 中任意两个对象之间相对关系的分类/回归问题，训练样本为 (d_i, d_j), $d_i, d_j \in D_q$，预测目标为 d_i, d_j 与 q 的相关性的相对大小，例如 1 表示 q_i 比 q_j 更相关。

Pairwise 缺点：

- 是在两个带查询对象之间的相关性的基础上学习的，学习的它们的相对顺序，并不是文档在最终的结果列表中的顺序
- 不同的查询可能会有不同大小的带查询对象集合，这种不均衡问题会影响算法的评估与学习

6.1.3 Listwise

Listwise 将一个 query 对应的所有相关待查询对象看作一个整体，作为单个训练样本，直接优化 MAP、NDCG（Normalized Discounted Cumulative Gain）这样的指标，从而学习到最佳排序结果。

7 Concepts about Math

7.1 chap01

7.1.1 采样

采样是生成一堆数据的过程，如果最终生成的数据服从某个分布，则可以称这堆数据采样自这个分布。

alias sampling 一种高效的针对**离散概率分布**采样方法，但是要先经过预处理，预处理的时间为 $O(n)$，但是与处理完成后采样的时间时 $O(1)$。预处理的大致思路如下：
对于一个给定的离散概率分布：$p(X = x_i), X = x_1, x_2, ..., x_n$。按照序号构造 n 个盒子，每个盒子按照顺序一一与 $x_1, ..., x_n$ 对应，每个盒子的高度为 $n \cdot p(X = x_i)$。接下来通过取长补短，把高度高于 1 的盒子切一部分分到其他高度低于 1 的盒子上，而且每个序号对应处不能有超过两个盒子。所以需要两个数组（Alias，Accept）来记录预处理的结果，一个用来记录每个序号除了原来的盒子还放了哪个盒子，一个用来记录每个序号的外来盒子有多高。
进行采样时，先决定使用哪个序号对应的盒子，再决定使用该序号内的原来的盒子还是外来的盒子。一个简单的例子 Fig.7.1.1

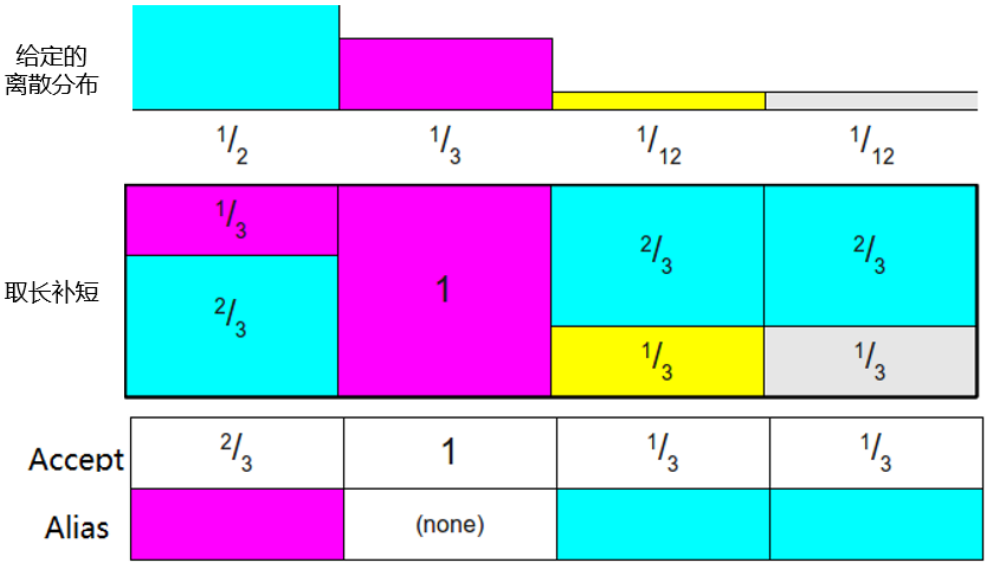


图 19: Alias Sample 例子

参考资料：

1. Darts, Dice, and Coins: Sampling from a Discrete Distribution
2. 【图嵌入】DeepWalk 和 Node2Vec

Importance Sampling 重要性采样，一种近似的抽样方法，通过一些数学上的变化，使得可以对一些不好抽样的分布进行抽样和估计。这个会在强化学习中的 off-policy 的方法中用到，从一个策略进行抽样，更新另外一个策略。求函数 $f(x)$ 的积分可以写成求期望的形式：

$$E_{x\sim p(x)}[f(x)] = \int p(x)f(x)dx \approx \frac{1}{n}\sum_i f(x_i)$$

然而通常数据分布会比较复杂且积分也是一个复杂的过程，因此会用采样来代替之。上式中的第三项就是用采样来代替积分，其中 $\frac{1}{n}$ 表示 $p(x) = \frac{1}{n}$，即数据的分布。但是有时候 $p(x)$ 是个很复杂的分布，从其重采样是很困难的，这个时候该怎么办呢?

找一个已易于采样的分布 $q(x)$，如正态分布，从 $q(x)$ 中采样得到的很多样本作为从 $p(x)$ 中采样的样本集，那么问题就来了，这俩又不是同一个分布，$q(x)$ 中采样的样本集分布符合 $p(x)$ 吗？先看一个公式：

$$E_{x\sim p(x)}[f(x)] = \int q(x)\frac{p(x)}{q(x)}f(x)dx \approx \frac{1}{n}\sum_i \frac{p(x)}{q(x)}f(x_i)$$

这个就是当我们从 $q(x)$ 中采样代替 $p(x)$ 后求 $f(x)$ 积分/期望的公式。可以看出对于从 $q(x)$ 中采样的样本赋予了不同的权重，因为样本集来自 $p(x)$ 的概率是不一样的，其中重要性就是 $\frac{p(x)}{q(x)}$。如 Fig.7.1.1所示。注意：图中 $p(z)$ 与 $f(z)$ 的含义，$p(z)$ 是一种分布，是相对于 z 轴的采样点而言的，比如在红色的两个驼峰处，z 的取点比较多，在其他地方 z 的取点就比较少，这叫样本分布服从 $p(z)$。对于 $f(z)$ 是一种映射关系，将 z 值映射到其他维度。

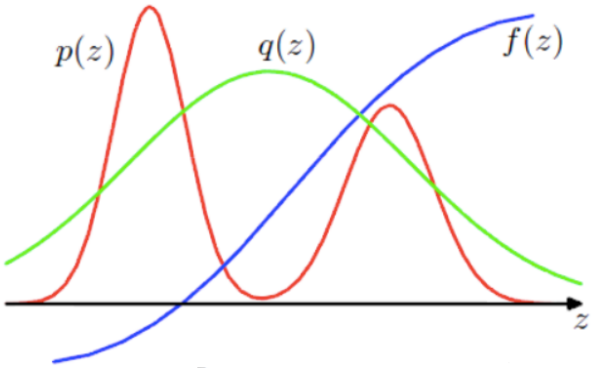


图 20: 重要性采样

参考：随机模拟-Monte Carlo 积分及采样（详述直接采样、接受-拒绝采样、重要性采样）。

接受-拒绝采样 同样的问题：对于一个难以采样的分布 $p(x)$，该怎么采样呢？选择一个易于采样的分布 $q(x)$，从中采样，以一定的概率接受或拒绝采样到的样本，使得经过筛选后的样本集是服从 $p(x)$ 的。具体该怎么操作呢？如 Fig.7.1.1所示，选择 $q(x)$，乘以 k 得到 $kq(x)$ 使之刚好能够保住 $p(x)$。对于 $q(x)$ 中采样到的样本 z_0，从 $[0,1]$ 的均匀分布中取一个数 u_0，如果 $u_0 \leq \frac{p(z_0)}{q(z_0)}$ 则接受 z_0。

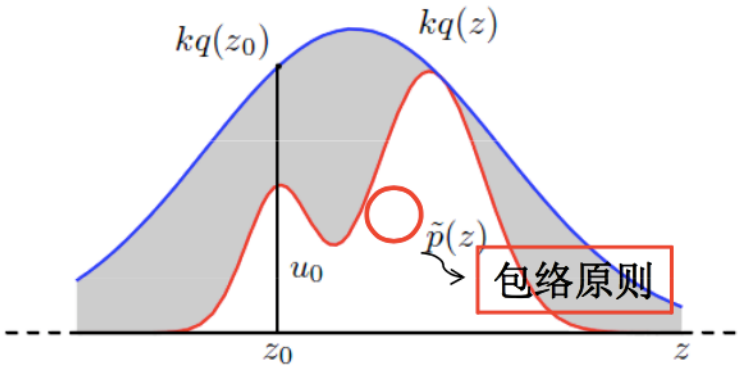


图 21: 接受-拒绝采样

7.1.2 eigen-value & eigen—vector

特征值，特征向量。这两者到底有什么意义呢?

7.1.3 Density estimation

7.1.4 MMD

Maximum mean discrepancy。用来衡量两个分布的差异。具体的衡量过程是：假设有两个分布 p, q，那么利用这两个分布分别生成两个样本集 ps, qs，再假设有一个函数 f，对于 $pm = \sum_{i \in ps} f(i), qm = \sum_{j \in qs} f(j)$, 则分布 p, q 在 f 上的 MMD 为 pm 与 qm 的差或某种基于 pm, qm 的计算值。

7.1.5 P, NP, NP-hard

P 问题：确定性计算机能够在指数级时间解决的问题；

NP 问题：非确定性计算机能够在指数级时间解决的问题；

NPC 问题：存在这样一个 NP 问题，所有 NP 问题都能约化成它，即只要解决了这个问题则所有 NP 问题都能解决。NPC 需要满足两个条件：

- 它是一个 NP 问题
- 所有的 NP 问题都能规约到它

NP-hard 问题：满足 NPC 问题的第二个条件，但不一定满足第一条。NP-hard 问题同样难以找到多项式时间的解法，但不一定是 NP 问题。这几者之间的包含关系如22所示。

傅里叶变换和小波分析 傅里叶变换：知道一段时间内，信号的各个频率分量分别有多少。小波变换：知道一段时间内，信号的各个频率分量分别有多少，以及它们都是什么时候出现的。

参考资料：《The Wavelet Tutorial》、如何通俗地讲解傅立叶分析和小波分析间的关系？- 咚懂咚懂咚的回答。

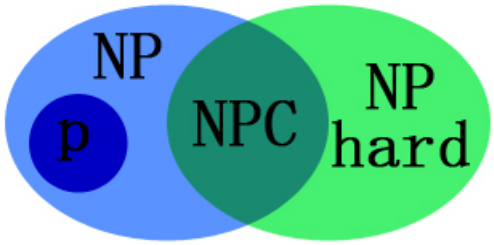


图 22: P_NP_NPC_NP-hard

7.1.6 l_1, l_2 范数对最优化问题的影响

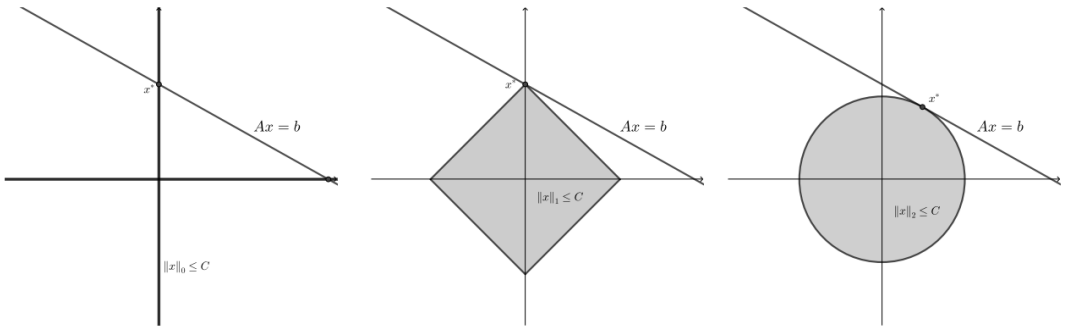
考虑以下优化问题：

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} \|x\|_p, \\ &\text{s.t. } Ax = b \end{aligned} \tag{1}$$

公式.1中 p 表示 0，1，2，$\|x\|_p$ 表示 x 的 l_p 范数。在深度学习中，通常希望得到稀疏的解，即在满足约束的情况下，x 中的非零值的数量尽可能多。咋一看，可能 l_0 范数是最符合情况的，但是 $\|x\|_0$ 是不连续的，当 $p = 0$ 时，公式.1就成了 NP 问题。当 A, b 满足一定条件时，$p = 1$ 的时公式.1的解也是 $p = 0$ 时的解。l_1 范数优化问题更易求解。那么有没有更容易求解的范数呢，l_2 可以吗?

对公式.1进行转化：由于范数本身也是函数，该优化问题就可以视为在 $Ax = b$ 的约束下，$\|x\|_p$ 的最小值，从函数图像角度来看这个优化问题，就是**目标函数与约束函数的交集 — 相交时的最小值**。

当 $x \in \mathbb{R}^2$ 时，如 Fig.23所示。对 l_0 范数，$\{x|\|x\|_0 \le 2\}$ 是全平面，它自然与 $Ax = b$ 相交；$\{x|\|x\|_0 \le 1\}$



(a) ℓ_0 范数　　(b) ℓ_1 范数　　(c) ℓ_2 范数

图 23: l_0, l_1, l_2 范数优化问题求解示意图

退化成两条直线即坐标轴，此时问题的解就是 $Ax = b$ 与坐标轴的交点。

- 对 l_1 范数，根据 C 不同，$\{x|\|x\|_1 \le C\}$ 为一系列正方形，这些正方形的顶点落在坐标轴上，$Ax = b$ 与这些正方形的交点一般是在正方形的顶点即相交于坐标轴，因此 l_1 范数的解有稀疏性
- 对 l_2 范数，根据 C 不同，$\{x|\|x\|_1 \le C\}$ 为一系列圆，且圆有光滑的边界，圆和 $Ax = b$ 的交点可以是圆上任何一点，所以 l_2 范数优化问题一般不能保证解的稀疏性

- 对 l_2 范数，根据 C 不同，$\{x|\|x\|_1 \leq C\}$ 为一系列圆，且圆有光滑的边界，圆和 $Ax = b$ 的交点可以是圆上任何一点，所以 l_2 范数优化问题一般不能保证解的稀疏性

注意：**这里目标函数与约束函数相交一般是指相切，$\{x|\|x\|_p \leq C\}$ 可以看成一个广义的球体，如果该球体与 $Ax = b$ 相交而不是相切，那么一定存在一个更小的 C' 使 $\|x\|_p$ 更小且与 $Ax = b$ 相切，则 C' 成了比 C 更优的解，故一般考虑相切。**

参考资料：

- 《最优化：建模、算法与理论》，刘浩洋、户将、李勇锋、文再文编著，第一章，1.2
- 机器学习中 L1 和 L2 正则化的直观解释

7.1.7 Reparametrization

重参数化技术。参考：漫谈重参数：从正态分布到 Gumbel Softmax

7.1.8 常用统计量

方差（Variance） 一组数据的方差，描述的是数据与它们的均值的离散程度，衡量了这组数据的集中程度。方差可以分为样本方差和总体方差：

- 样本方差：$S^2 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}$，其中 $\bar{x}$ 是**样本均值**
- 总体方差：$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}$，其中 μ 是**总体均值**

为什么不用要平方？如果使用平均差（$\frac{\sum_{i=1}^{N}|x_i - \bar{x}|}{N}$）来衡量一组数据的离散程度，可能不能很好的体现数据的分散程度（参考：为什么要求差的平方？），加上平方后可以放大偏离均值太远的数据的影响。也许这有点像注意力机制，在平均差中，$|x_i - \bar{x}|$ 的注意力值是 $\frac{1}{N}$，在方差中，$|x_i - \bar{x}|$ 的注意力值是 $\frac{|x_i - \bar{x}|}{N}$，这可以体现离均值越远的点对离散程度的贡献越大。

标准差（Standard Deviation） 标准差的平方就是方差，同理，标准差也可以分为样本 std. 和总总体 std.：

- 样本标准差：$S = \sqrt{S^2}$
- 总体标准差：$\sigma = \sqrt{\sigma^2}$

T-statistic

p-value

t test、χ^2 检验 χ^2 检验通常用于检验两个事件的独立性，例如可以用于分析自变量与因变量之间的独立性。如果 χ^2 的值越大，则说明二者之间的关联性越大。

7.1.9 数据的度量

定类变量 即类别变量，其值域是某个离散的类别。能够对对象进行分类，能够判断对象之间是否同类或异类，如性别。**不同类别之间没有大小关系** 。

【可以分类（= 和 ≠），但不能排序】

定序变量 定序变量的值不仅能够代表事物的分类，还能代表事物按某种特性的排序，但定序变量的值之间没有确切的间隔距离，**只能排列出它们的顺序，而不能反映出不同值之间的距离** ，即不能反映一个值比另一个值大多少或小多少。如文化水平，其取值可以是文盲、小学、中学、大学等，值之间有顺序关系（如大学 > 小学），但不能反映不同文化程度之间的距离。

【可以分类（= 和 ≠），可以排序（> 和 <），但不能（+ 和 − ）】

定距变量 定距变量的值之间**可以比较大小，两个值的差有实际意义** 。能确切测量值之间的高低、大小次序之间的距离，因而具有加与减的数学特质。但是，**定距变量没有一个绝对的零点** ，不能乘除或倍数的形式来说明它们之间的关系。例如华氏温度：10、20、30，30 比 20 高 10，但华氏度 30 不是 10 的三倍热（**0 不是没有温度** ）。

【可以分类（= 和 ≠ ），可以排序（> 和 <），可以（+ 和 − ），但不能（× 和 ÷ ）】

定比变量 定比变量除了具有定距变量的特性外，还具有一个真正的零点，因而它具有乘与除（Œ、œ）的数学特质。如 A 的体重是 60kg，而 B 的体重是 30kg，可以算出前者是后者的两倍重，因为其零点是绝对的。

【可以分类（= 和 ≠ ），可以排序（> 和 <），可以（+ 和 − ），可以（× 和 ÷ ）】

以上四种变量类型的性质是逐渐继承的。

参考：【统计学】区分定类、定序、定距、定比变量!! 。

8 Algorithms

8.1 Basic

8.1.1 大整数

Motivation 在 C、C++ 等一些语言中，支持的整数类型所能保存的范围是有效的，相对 Java、Python 等一些语言中支持的范围是要小的。

HOW? 考虑几个可能的涉及大整数的例子：大整数的加法、大整数的减法大整数乘某个数大整数除以某个数。主要涉及到的问题有：

- 大整数的存储
- 大整数之间的加减法
- 大整数与一个常规整数的乘除法

8.1.2 前缀和与差分

8.1.3 并查集

Motivation 将 n 个不同的元素分成一组**不相交**的集合。能够快速地完成两种操作：查找某个元素属于那个集合、合并两个集合。注意："不相交"可以有不同的定义，如 graph 中的节点是否相邻、在某种度量下的不相交等。

HOW?

8.2 Search

8.2.1 贪心算法

贪心搜索是一种搜索解空间的策略，在搜索时，基于当前状态，选择代价最低或估计收益最大的方向作为下一步搜索的方向。使用贪心搜索并不能保证全局最优解，目的是为了找到一个可行解。在使用贪心算法前，先要证明以下两个东西：

- 贪心选择性：指所求问题的整体最优解可以通过一系列局部最优的选择（通过最优度量标准来判断）来到达
- 最优子结构：一个问题的最优解中包含了子问题（比原问题规模更小的问题）的最优解，即每一步贪心选完后会留下子问题，子问题的最优解和贪心选出来的解可以凑成原问题的最优解

和动态规划相比，二者都需要描述清除问题的优化子结构；但是贪心算法重点是证明贪心选择的合理性，DP 重点是找到子问题和合理的递推关系式。

典型的贪心问题：Huffman 编码、活动选择问题。

8.2.2 Beam Search

也叫集束搜索。对贪心搜索的一种改进。在进行搜索时，并不是只沿一个方向搜索，会同时保留 n 个最优的方向，分别以这 n 个方向进行扩展，扩展后在每个方向都可以得到 n 个备选项，那么则有 n^2 个备选项，在这 n^2 个被选项里选择最优的 n 个备选项作为下次扩展的方向，以此类推，即在搜索树的每一层都只保留 n 个方向。n 就是 beam width。

9 LaTeX

9.1 在 pdf 中嵌入代码

在 LaTeX 中嵌入代码有多种方法。可以使用的包有：listings、minted、tcolorbox 等方式。

9.2 参考文献相关

引入参考文件 引入参考文献时，可以使用.bib 文件的方式引入参考文件。写好.bib 文件后，需要按序执行以下四步完成引用：

1. 先编译.tex 文件。个人认为是编译.tex 文件，为参考文献产生位置
2. 编译.bib 文件（一般使用 bibtex），生成编译后的参考文献
3. 再编译.tex 文件，这时会将参考文献引入到.tex 编译后的文件中（好像是.aux 文件）。但这时在正文中可能不会正常显示
4. 再次编译.tex 文件即可得到引用正确、显示正确的 pdf

9.3 数学符号

数学字体 其中某些单元格的内容可能出错了，只需要把多余的字符去掉即可。

A	A	$\mathbb{A}$	$\mathcal{A}$	$\mathfrak{A}$	A	$\mathbf{A}$
A	A	$\mathbb{A}$	$\mathcal{A}$	$\mathfrak{A}$	A	$\mathbf{A}$
B	B	$\mathbb{B}$	$\mathcal{B}$	$\mathfrak{B}$	B	$\mathbf{B}$
C	C	$\mathbb{C}$	$\mathcal{C}$	$\mathfrak{C}$	C	$\mathbf{C}$
D	D	$\mathbb{D}$	$\mathcal{D}$	$\mathfrak{D}$	D	$\mathbf{D}$
E	E	$\mathbb{E}$	$\mathcal{E}$	$\mathfrak{E}$	E	$\mathbf{E}$
F	F	$\mathbb{F}$	$\mathcal{F}$	$\mathfrak{F}$	F	$\mathbf{F}$
G	G	$\mathbb{G}$	$\mathcal{G}$	$\mathfrak{G}$	G	$\mathbf{G}$
H	H	$\mathbb{H}$	$\mathcal{H}$	$\mathfrak{H}$	H	$\mathbf{H}$
I	I	$\mathbb{I}$	$\mathcal{I}$	$\mathfrak{I}$	I	$\mathbf{I}$
J	J	$\mathbb{J}$	$\mathcal{J}$	$\mathfrak{J}$	J	$\mathbf{J}$
K	K	$\mathbb{K}$	$\mathcal{K}$	$\mathfrak{K}$	K	$\mathbf{K}$
L	L	$\mathbb{L}$	$\mathcal{L}$	$\mathfrak{L}$	L	$\mathbf{L}$
M	M	$\mathbb{M}$	$\mathcal{M}$	$\mathfrak{M}$	M	$\mathbf{M}$
N	N	$\mathbb{N}$	$\mathcal{N}$	$\mathfrak{N}$	N	$\mathbf{N}$
O	O	$\mathbb{O}$	$\mathcal{O}$	$\mathfrak{O}$	O	$\mathbf{O}$
P	P	$\mathbb{P}$	$\mathcal{P}$	$\mathfrak{P}$	P	$\mathbf{P}$
Q	Q	$\mathbb{Q}$	$\mathcal{Q}$	$\mathfrak{Q}$	Q	$\mathbf{Q}$
R	R	$\mathbb{R}$	$\mathcal{R}$	$\mathfrak{R}$	R	$\mathbf{R}$
S	S	$\mathbb{S}$	$\mathcal{S}$	$\mathfrak{S}$	S	$\mathbf{S}$
T	T	$\mathbb{T}$	$\mathcal{T}$	$\mathfrak{T}$	T	$\mathbf{T}$
U	U	$\mathbb{U}$	$\mathcal{U}$	$\mathfrak{U}$	U	$\mathbf{U}$
V	V	$\mathbb{V}$	$\mathcal{V}$	$\mathfrak{V}$	V	$\mathbf{V}$
W	W	$\mathbb{W}$	$\mathcal{W}$	$\mathfrak{W}$	W	$\mathbf{W}$
X	X	$\mathbb{X}$	$\mathcal{X}$	$\mathfrak{X}$	X	$\mathbf{X}$
Y	Y	$\mathbb{Y}$	$\mathcal{Y}$	$\mathfrak{Y}$	Y	$\mathbf{Y}$
Z	Z	$\mathbb{Z}$	$\mathcal{Z}$	$\mathfrak{Z}$	Z	$\mathbf{Z}$

9.4 LaTeX 中的颜色

参考latexcolor。

9.5 注脚

两种常用形式：

- `\footnote{注脚的内容}`
- 这种方式很方便，可以在多处共享一个注脚，只需使用同一个注脚编号即可

```
\footnotemark[注脚编号]
\footnotetext[编号]{注脚内容}
```

9.6 章节编号

默认是从 1 开始编号的，如果要从 0 开始，可以加一条命令：

```
\setcounter{section}{-1}
```

10 Coding

10.1 C++

参考文档：cplusplus.com

10.1.1 C++ 数据类型转换

数字 <=> 字符串

- itoa: 整数转为字符串，可以设置基底
- stoi: 字符串转为整数，可以设置基底
- atoi、atol、atoll: 将字符串转为 int/long/long long，可以设置基底
- to_string: 可以讲数字转为字符串，包括整数、小数等

10.1.2 C++ 集合

通过 $\#include < set >$ 来引入 set。关于集合的一些运算，包含在 algorithm 库中。在 algorithm 库中有一个 includes 函数，可以用来判断两个容器之间是否存在包含关系，但是，要注意**两个容器都要先从小到大进行排序!!!**

```
1 // 省略头文件和库的包含代码
2 int container[] = {5,10,15,20,25,30,35,45,50};
3 int continent[] = {40,30,20,10};
```

```
4
5   sort (container,container+10);
6   sort (continent,continent+4);
7
8   // using default comparison:
9   if ( std::includes(container,container+10,continent,continent+3) )
10    cout << "container includes continent!\n";
11  else  cout<<"NO\n";
12
```

10.1.3 字符串

C++ 中字符串用双引号括起来，字符用单引号括起来。

字符串比较 compare

常用方法

- 初始化：`string s(n, 'c');`
- 插入字符串：`s.insert(pos, len, "insert");`
- 删除子串：`s.erase(pos, len, "erase");`
-

10.1.4 数组

数组初始化

- `type [][depth][depth]...[depth]`声明多维数组时，只能有一维的大小不指定
- `int arr[5] = {1, 2, 3, 4, 5}`声明数组时可以给数组赋值（初始化列表），如果指定的数组大小则 {} 中的值的数量应该小于等于数组大小，如果数量小于数组大小，则数组中后续位置将会被填充默认值（对于基础类型，则为 0）。因此，当 {} 未空时，数组将会被填充为默认值
- `int *p = new int[5]`将在堆中分配内存，此时也可以对数组进行初始化：`int *p = new int[]{1, 2, 3, 4, 5}`、`int *p = new int[]()`。对于基本类型，如果不进行初始化，局部定义的数组的值是未定义的；对于类类型，会默认调用其默认构造函数。**堆中分配的数组要用 delete 删除**

10.1.5 遍历 vector 的方式

```
1   vector<int> nums = {1, 2, 3, 4, 5};
2   // 方法1：下标
```

```
  for(int i = 0; i < nums.size(); i++){
    // do something
  }

  // 方法2: 迭代器
  for(vector<int>::iterator it = nums.begin(); it != nums.end(); it++){
    // do something
  }

  // 方法3: auto, 以引用的形式遍历, 可以修改值
  for(auto &it: nums){
    // do something
  }

  // 方法4: auto, 以值的形式遍历, 不可以修改值
  for(auto it: nums){
    // do something
  }

  // 方法5: for_each
  foreach(nums.begin(), nums.end(),
      [](const auto &val) -> void {
        // do something
      })

```

10.1.6 for_each

for_each 源码如下:

```
template<class InputIterator, class Function>
Function for_each(InputIterator first, InputIterator last, Function fn)
{
  while (first!=last) {
    fn (*first);
    ++first;
  }
  return fn;       // or, since C++11: return move(fn);
}
```

其中参数 fn 是一个一元函数（即只能有一个参数），它的返回值会被忽略。

10.1.7 deque

`#include <deque>`。double ended queue，双端队列。deque 是一种动态容器（通常以动态数组的形式，如链表，来实现），可以在两端增加或者删除元素，随机访问 deque 中的元素。

10.1.8 priority_queue

`#include <queue>`。

```
template <class T, class Container = vector<T>,
class Compare = less<typename Container::value_type> > class priority_queue;
```

priority_queue，优先队列。priority_queue 是一种容器适配器（container adaptor），它使用某种容器（可以是 queue、vector 等，默认是 vector）作为其数据的容器，并提供了一些接口来访问容器内的元素，重点就在这些接口上（push，pop，top）！priority_queue 的特点：

- 第一个元素是优先级最大/小的元素（大/小根堆）
- top： 返回顶部的元素，即优先级最大/最小的元素
- pop： 删除顶部的元素

例子：

```cpp
#includde <queue>

int main(){
        // 大根堆
        priority_queue<int> a; // <=> priority_queue<int, vector<int>, less<int> >a

        // 小根堆
        priority_queue<int, vector<int>, greater<int> > b;
}
```

10.2 Python

defaultdict python 中的 defaultdict，也是一种 dict。与传统的 dict 不同，当传统的 dict 的 key 不存在时会 Error，但 defaultdict 不会，而是返回一个默认值。该默认值由创建 defaultdict 时传入的参数有关：$dd = defaultdict(default_factory)$。

当 default_factory 是 list 时，默认值是 [],defaultdict 其他常见取值还有：str, set, int, dict。但不可以是 defaultdict。

glob python 中的一个内置模块，用于查找符合特定规则的文件路劲。如：

```python
import glob
pattern = "./myfloder/prefix-*.txt"
li = glob.glob(pattern) # 此时li中就包含了所有符合pattern这个模式的文件名
```

np.argsort 对数组的元素进行排序（默认是从小到大），生成一个新的数组，数组元素是排序后的数组所对应的下标。

```
import numpy as np
x = np.array([2,1,3,5,4])
y = np.argsort(x) # y = [1, 0, 2, 4, 3]
# 从大到小排序
y = np.argsort(-x) # y = [3, 4, 2, 0, 1]
```

该方法还有更复杂的使用方法，可以根据参数进行调节，$argsort(a, axis = -1, kind = None, order = None)$。

numpy 设置随机数种子 $np.random.seed(seed)$。

python format 用法 对齐"{:}" 通常将对齐符号放在：后。^|<|> 分别是居中、左对齐、右对齐，在填充符号后面可带宽度，在：后可带填充字符，默认为空格。

数字输出格式主要包括小数位数（如.2f）、百分号输出（如.2%）、指数形式输出（如.2e）、带正负符号的输出（如 +.2f）、按不同进制的输出（b、d、o、x 分别是二进制、十进制、八进制、十六进制，在 b|o|x 前加 # 可以输出进制符号，x 的大小写会影响进制符号的大小写。

```
"{:^8}".format("居中")  # 居中显示，宽度为8，默认用空格填充
"{:*<8}".format("左对齐") # 左对齐显示，宽度为8，用*填充
"{:*>8}".format("右对齐") # 右对齐显示，宽度为8，默认用空格填充
"{:.2f}".format(123)
"{:.2%}".format(123)
"{:^8.2f}".format(123)    # 八位的显示宽度，保留2位小数
"{:b}".format(11)
"{:d}".format(11)
"{:o}".format(11)
"{:x}".format(11)
"{:#x}".format(11)
"{:#X}".format(11)
```

numpy nan 的处理 numpy 中可以使用 np.isnan() 来判断数组中的元素是否为 NAN，返回的结果与数组形状相同，元素为 True/False，该位置的元素为 NAN 是则为 True，否则为 False。可以使用 np.nan_to_num(x, copy=True, nan=0, posinf= 1.7976931348623157e+308, neginf=-1.7976931348623157e+308) 来进行转换。该函数可以将 NAN（包括无穷小、无穷大）转换为数字，可以指定转换后的数字。参数：x 是待转换的数据，可以是数组或单个数字；copy 表示是否进行原地转换，相当于 pandas 中的 in_place 参数，但取值与 in_place 相反；nan 表示取代 NAN 的数字；posinf、neginf 表示用什么取代正/负无穷。

找到数组中 nan 指的位置 ：np.argwhere(np.isnan(a))。解释：通过 np.isnan 标记数组中 nan 值的位置为 True，np.argwhere 会返回那些 True 的下标。

matplotlib 中的 marker 如图24所示，更多内容可参见：Matplotlib

matplotlib Animation Animation 可用于制作动画。初始化函数：

```
def __init__(self, fig, func, frames=None, init_func=None, fargs=None,
save_count=None, *, cache_frame_data=True, **kwargs)
```

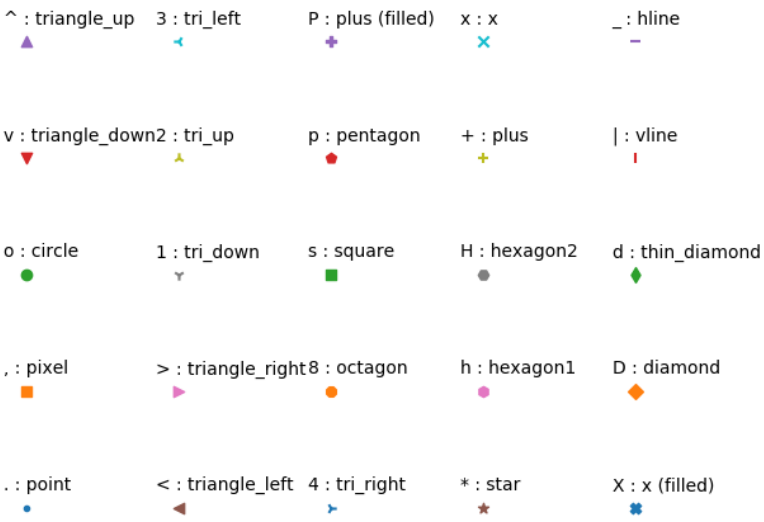


图 24: Matplotlib 中的 markers

```
    """
    fig: 绘制动画的地方
    func: 完成动画动作的回调函数。定义应为: def func(frame, *fargs) -> iterable_of_artists
    frames: 可以认为是每一个动画帧的索引，通常为iterable对象，frames中的每个值会逐个作为func的第一
                                                个参数传入
    init_func: 在绘制第一帧之前调用的函数，定义应为: def init_func() -> iterable_of_artists
    fargs: 用于func的额外的参数
    repeat_delay: 帧之间的延迟，单位为毫秒
    repeat: 是否重复播放动画
    blit: 布尔值故，  默认为False。用于优化绘图，注意这个参数有点坑，当其为True时，会根据artists的
                                                zorder来绘图，当对象没有zorder（如bar）时则会报
                                                错

    """
```

在使用 matplotlib 制作动画的过程中，很重要的是要知道要让什么对象发生变化、发生什么变化 — 知道了这二者之后就可以在每一帧的更新函数中完成对应的动作了。

注意 ：blit 参数的使用，当启用了 blit 时，且要对 bar 改变时，则会报错：*AttributeError: 'BarContainer' object has no attribute 'get_zorder'*，把 blit 设为 False 即可。

matplotlib 调整子图间距

- plt.tight_layout()
- plt.subplot_adjust(left=None, bottom=None, right=None, top=None, wspace=None, hspace=None)

matplotlib 显示中文

```
plt.rcParams['font.sans-serif'] = ['SimHei'] # 步骤一（替换sans-serif字体）
```

```
plt.rcParams['axes.unicode_minus'] = False  # 步骤二（解决坐标轴负数的负号显示问题）
```

10.3 Tensorflow/PyTorch

tf.variable_scope & tf.name_scope

tf.sparse_tensor tf 中的稀疏张量。使用三个稠密的张量来表示。

- indices: 表示原张量中非零值的位置
- values: 表示 indices 中元素所指位置上的值
- dense_shape: 表示原张量的 shape

tf.train.Saver tf 中用于保存、恢复模型参数的接口。主要由两个接口：1）tf.train.Saver().save() 用于保存模型；2）tf.train.Saver().restore() 用于回复模型。

```
import tensorflow as tf
saver = tf.train.Saver(max\_to\_keep=3) # max\_to\_keep表示保存的checkpoint最大次数
...
# 保存模型
# sess: 会话的名字
# save\_path: 模型的保存路径
# global\_step: 保存模型时的后缀
# 使用以下方法保存模型后会产生四个文件，分别是：
# checkpoint文件：会记录最新的模型是哪个
# .ckpt.meta文件：包含元图，保存了计算图的结构，没有变量的值
# .ckpt.data 文件：保存权重等参数
# .ckpt.index 文件：为数据文件提供索引，{还不太确定}
saver.save(sess=sess, save_path=model_save_path, global_step=step)
...
# 恢复模型
# save_path可以不用加模型的后缀
saver.restore(sess=sess, save_path=model_save_path)
```

tf.Session 运行 tf 中操作的类，Session 类封装了运行操作所需要的环境。可以使用手动开启和关闭、上下文的方式使用 Session。在使用 Session 时还可以进行配置，将配置作为 Session 初始化的参数传入。

tf 中的 RNN RNN 的训练数据与其他神经网络的训练数据有一点不一样。通常的 NN 中的训练数据，每个样本就是一个向量。但是在 RNN 中，每个训练样本就是一个矩阵，每一行表示一步的输入，正因为 RNN 所处理的问题不同，RNN 的输出数据也变得更加复杂。
针对不同的 RNN 有不同形式的输出，但可以进行简单的归纳：对于 RNN 中的每个样本，其形式为 $\mathbb{R}^{max\_time \times embed\_size}$，其中 max_time 表示所有样本中序列的最大长度，$embed\_size$ 表示每一步的输入的长度。当输入了一个样本后，会将样本的每一行 — 即每一步的数据输入到 RNN 中，由于 RNN 本身的特点，每一步都可以产生输出，通常来说包含每一步的输出和状态（具体到不同的 RNN 时会有不同）。每一步的输出维度会收到 RNN 模型隐层宽度 — 即隐层单元数量的影响。并且针对不同的任

务，最终需要的输出是不一样，例如 m 对 n、m 对 1、m 对 m 的任务等等。
在 tf 中，有多种关于 RNN 单元的类，如 BasicLSTMCell、GRUCell、MultiRNNCell。可以在这些类的基础上搭建自己的 RNN 模型。

ImageDataGenerator 这是 Keras 中的一个图像数据生成器，同时也可以在 batch 中对数据进行增强，扩充数据集大小，增强模型的泛化能力。比如进行旋转，变形，归一化等。

tf.where tf.where(condition, x=None, y=None, name=None)

condition 是一个布尔数组，condition、x、y 的维度必须相同。

其中 x、y 可以为 None，此时返回的是一个二维数组，该二维数组的行数表示 condition 中为 True 的元素的数量，每一行代表对应为 True 元素的坐标。

当 x、y 不为 None 是（x，y 必须同时不为 None），则会创建一个与 condition 同样 shape 的张量，该张量某位置的元素的取值来自 x 或 y，当该位置在 condition 中为 True 是则来自 x，否则来自 y。

pytorch tensor.view view()相当于数据库中的 view — 对数据进行查看，是查看数据的一种方式。使用 view() 不会产生数据的复制，与原 tensor 共享同一块内存，修改 view 会使原 tensor 发生变化。

TFRecord 下列表来自简书：

- Record 顾名思义主要是为了记录数据的。
- 为了更加方便的建图，原来使用 placeholder 的话，还要每次 feed_dict 一下，使用 TFRecord+Dataset 的时候直接就把数据读入操作当成一个图中的节点，就不用每次都 feed 了。
- 可以方便的和 Estimator 进行对接。
- TFRecord 以字典的方式进行数据的创建。

tf.Graph 不同 Graph 之间的变量是不可以共享的！

tf.pad 这是 tf 种对 tensor 进行填充的函数。*tf.pad(tensor, paddings, mode='CONSTANT', constant_values=0, name=None)*。其中 tensor 为要被填充的张量；paddings 表示填充时要填充多少值进去，控制了填充后 tensor 的大小；mode 表示填充的模式，什么样的值被填充进去；constant_value 表示 mode 为 CONSTANT 时的常量值。其中最重要的参数是 *paddings*。*paddings* 是一个形状为 [n,2] 的 tensor，n 为 input 的 rank（其实可以简单理解为 len(input.shape)）。*paddings* 中的第 i 个元素可以看做是一个长度为 2 的数组，第一个元素表示了在 input 的第 i 维的前面增加的大小，第二个元素表示在第 i 维的后面增加的大小，那么对于 input 的第 i 来说，它的大小较原来增大了 $paddings[i][0]+paddings[i][1]$。

tf.tile 平铺张量。*tf.tile(input, multiples, name=None)*。其中 input 是一个张量；multiples 是一个 1-D 数组，数组的长度是 input 的维度数量，每个元素的值表示 input 被复制的次数，即 *input.dims[i] = input.dims[i] * multiples[i]*。

tf.squeeze　压缩张量，将张量中维度大小为 1 维压缩掉。

tf.squeeze(input, axis=None, name=None)。

input 为输入的张量；axis 为可选参数，可以为整数或者整数 1-D 数组，为 None 时表示移除所有大小为 1 的维，否则移除指定的维。

反之则为 *unsqueeze*。

tf.expand$_d$*ims*　给数据增加维度。*tf.expand_dims(input, axis, name)*，表示在 *input* 的第 *axis* 维插入一个维度，维度大小为 1。

10.4　ML/DL 错误集锦

batch_size 对训练的影响

某个 iteration 成为训练效果的短板

损失函数值为 NaN

类别的正负样本不均衡问题

数据的格式问题

11　Others

11.1　李沐 — 用随机梯度下降来优化人生

要有目标。你需要有目标。短的也好，长的也好。认真定下来的也好，别人那里捡来的也好。就跟随机梯度下降需要有个目标函数一样。

目标要大。不管是人生目标还是目标函数，你最好不要知道最后可以走到哪里。如果你知道，那么你的目标就太简单了，可能是个凸函数。你可以在一开始的时候给自己一些小目标，例如期末考个 80 分，训练一个线性模型。但接下来得有个更大的目标，财富自由也好，100 亿参数的变形金刚也好，得足够一颗赛艇。

坚持走。不管你的目标多复杂，随机梯度下降都是最简单的。每一次你找一个大概还行的方向（梯度），然后迈一步（下降）。两个核心要素是方向和步子的长短。但最重要的是你得一直走下去，能多走几步就多走几步。

痛苦的卷。每一步里你都在试图改变你自己或者你的模型参数。改变带来的痛苦。但没有改变就没有进步。你过得很痛苦不代表在朝着目标走，因为你可能走反了。但是过得很舒服那一定在原地踏步。需要时刻跟自己作对。

可以躺平。你用你内心的激情来迈步子。步子太小走不动，步子太长容易过早消耗了激情。周期性的调大调小步长效果挺好。所以你可以时不时休息休息。

四处看看。每一步走的方向是你对世界的认识。如果你探索的世界不怎么变化，那么要么你的目标太简单，要么你困在你的舒适区了。随机梯度下降的第一个词是随机，就是你需要四处走走，看过很多地方，做些错误的决定，这样你可以在前期迈过一些不是很好的舒适区。

快也是慢。你没有必要特意去追求找到最好的方向和最合适的步子。你身边当然会有幸运之子，他们每一步都在别人前面。但经验告诉我们，随机梯度下降前期进度太快，后期可能乏力。就是说你过早的找到一个舒适区，忘了世界有多大。所以你不要太急，前面徘徊一段时间不是坏事。成名无需太早。

赢在起点。起点当然很重要。如果你在终点附近起步，可以少走很多路。而且终点附近的路都比较平，走着舒服。当你发现别人不如你的时候，看看自己站在哪里。可能你就是运气很好，赢在了起跑线。如果你跟别人在同一起跑线，不见得你能做更好。

很远也能到达。如果你是在随机起点，那么做好准备面前的路会非常不平坦。越远离终点，越人迹罕至。四处都是悬崖。但随机梯度下降告诉我们，不管起点在哪里，最后得到的解都差不多。当然这个前提是你得一直按照梯度方向走下去。如果中间梯度炸掉了，那么你随机一个起点，调整步子节奏，重新来。

独一无二。也许大家有着差不多的目标，在差不多的时间毕业买房结婚生娃。但每一步里，每个人的内心中看到的世界都不一样，导致走的路不一样。你如果跑多次随机梯度下降，在各个时间点的目标函数值可能都差不多，但每次的参数千差万别。不会有人关心你每次训练出来的模型里面参数具体是什么值，除了你自己。

简单最好。当然有比随机梯度下降更复杂的算法。他们想每一步看向更远更准，想步子迈最大。但如果你的目标很复杂，简单的随机梯度下降反而效果最好。深度学习里大家都用它。关注前面，每次抬头瞄一眼世界，快速做个决定，然后迈一小步。小步快跑。只要你有目标，不要停，就能到达。

参考文献

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *international conference on learning representations*, 2016.

[2] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. *ICML*, pages 941–949, 2018.

[3] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. *ICLR*, 2018.

[4] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide deep learning for recommender systems. *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, 2016.

[5] Kyunghyun Cho, van Bart Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP*, pages 1724–1734, 2014.

[6] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. *RecSys*, pages 191–198, 2016.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *north american chapter of the association for computational linguistics*, 2019.

[8] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. JMLR Workshop and Conference Proceedings.

[9] L. William Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 30 (NIPS 2017)*, pages 1024–1034, 2017.

[10] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. Nais: Neural attentive item similarity model for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, pages 2354–2366, 2018.

[11] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Quiñonero Joaquin Candela. Practical lessons from predicting clicks on ads at facebook. *ADKDD@KDD*, pages 1–9, 2014.

[12] N. Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *international conference on learning representations*, 2017.

[13] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, pages 76–80, 2003.

[14] Thang Luong, Hieu Pham, and D. Christopher Manning. Effective approaches to attention-based neural machine translation. *Conference on Empirical Methods in Natural Language Processing*, 2015.

[15] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[16] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011.

[17] Y. Patrice Simard, Dave Steinkraus, and John Platt C. Best practices for convolutional neural networks applied to visual document analysis. *ICDAR-1*, pages 958–963, 2003.

[18] Ilya Sutskever, Oriol Vinyals, and V. Quoc Le. Sequence to sequence learning with neural networks. *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 27 (NIPS 2014)*, pages 3104–3112, 2014.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, N. Aidan Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 30 (NIPS 2017)*, pages 5998–6008, 2017.