机器学习

大纲

目录

机器学习基础

第一章 基础知识

1.1 NFL 定理

归纳偏好用于描述当特征相同时，哪些特征更为重要

假设样本空间 $\mathcal{X}$ 和假设空间 $\mathcal{H}$ 为离散。令 $P(h|X,\xi_a)$ 代表算法 ξ_a 基于训练数据 X 产生假设 h 的概率；令 f 代表希望学习的目标函数。因此，算法在训练集外产生的误差为:

$$E_{ote}(\xi_a|X,f)=\sum_h\sum_{\boldsymbol{x}\in\mathcal{X}-X}P(X)\mathbb{I}(h(\boldsymbol{x})\neq f(\boldsymbol{x}))P(h|X,\xi_a) \tag{1.1}$$

其中 $\mathbb{I}(\cdot)$ 为指示函数，当 $\cdot$ 为真时返回 1，否则返回 0。

若学习目标为二分类，则 $\mathcal{X}\mapsto\{0,1\}$ 且函数空间为 $\{0,1\}^{|\mathcal{X}|}$，其中 $|\cdot|$ 用于计算集合长度。

算法用于解决多个任务，则拥有多个学习的目标函数；假设这些目标函数均匀分布，则这些目标函数的误差总和为:

$$\begin{aligned}\sum_f E_{ote}(\xi_a|X,f)&=\sum_f\sum_h\sum_{\boldsymbol{x}\in\mathcal{X}-X}P(X)\mathbb{I}(h(\boldsymbol{x})\neq f(\boldsymbol{x}))P(h|X,\xi_a)\\&=\sum_h\sum_{\boldsymbol{x}\in\mathcal{X}-X}P(X)P(h|X,\xi_a)\sum_f\mathbb{I}(h(\boldsymbol{x})\neq f(\boldsymbol{x}))\\&\text{根据假设，总有一半是正确的，因此}\\&=\sum_h\sum_{\boldsymbol{x}\in\mathcal{X}-X}P(X)P(h|X,\xi_a)\frac{1}{2}2^{|\mathcal{X}|}\\&=2^{|\mathcal{X}|-1}\sum_{x\in\mathcal{X}-X}P(X)\end{aligned} \tag{1.2}$$

因此可知，在多目标目标函数均匀分布的情况下，不同算法所得的误差总和相同。实际情况中，某一算法通常只用于解决单一问题，且其目标函数的分布不均匀（即目标函数重要性不同），因此不同算法所得的误差总和不同。

这告诉我们，在某一任务上表现好的算法在另一任务上表现不一定好。

1.2 Monte-Carlo estimation

Monte-Carlo estimation 可以用于估计复杂积分，假设 $f:\mathbb{R}^d\to\mathbb{R}$，以及一个定义在 $\mathcal{D}\in\mathbb{R}^d$ 上的 pdf $p:\mathbb{R}^d\to\mathbb{R}$，期望计算积分:

$$I=\int_D f(\boldsymbol{x})d\boldsymbol{x} \tag{1.1}$$

对于上式，假设 $\boldsymbol{x}\sim p(\boldsymbol{x})$，则可以变形为:

$$I=\int_D p(\boldsymbol{x})\frac{f(\boldsymbol{x})}{p(\boldsymbol{x})}d\boldsymbol{x}=\mathbb{E}_p\left[\frac{f(\boldsymbol{x})}{p(\boldsymbol{x})}\right] \tag{1.2}$$

因此可以设计 Monte-Carlo estimator 为:

$$\hat{I}_N=\frac{1}{N}\sum_{i=1}^N\frac{f(\boldsymbol{x_i})}{p(\boldsymbol{x_i})} \tag{1.3}$$

且具有无偏性

- 无偏性:

$$\mathbb{E}\left[\hat{I}_N\right]=I \tag{1.4}$$

- 方差:

$$\mathrm{Var}\left(\hat{I}\right)=\frac{1}{N}\left(\mathbb{E}\left[\left(\frac{f(\boldsymbol{x})}{p(\boldsymbol{x})}\right)^2\right]-I^2\right) \tag{1.5}$$

特别的，当从均匀分布中采样时，$p(\boldsymbol{x})=\frac{1}{V}$，其中 V 为 $\mathcal{D}$ 的体积，则:

$$\hat{I}_N=\frac{V}{N}\sum_{i=1}^N f(\boldsymbol{x_i}) \tag{1.6}$$

当积分代表期望时，可以使用 Monte-Carlo estimation:

$$\begin{aligned}I&=\int_D p(\boldsymbol{x})f(\boldsymbol{x})d\boldsymbol{x}\\&=\frac{1}{N}\sum_{i=1}^N f(\boldsymbol{x_i}),\quad \boldsymbol{x_i}\sim p(\boldsymbol{x})\end{aligned} \tag{1.7}$$

1.3 CNN

1.3.1 Convolution

卷积通常是指:

$$Y_{i,j} = (X * W)_{i,j} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X_{i+m,j+n} W_{m,n} \tag{1.1}$$

卷积操作包括 Conv1d 和 Conv2d

Conv1d: 通常对于一维特征图，给定输入 $X \in \mathbb{R}^{N \times L \times C_{\text{in}}}$，filter $W \in \mathbb{R}^{K \times C_{\text{in}} \times C_{\text{out}}}$，卷积操作为:

$$Y_{n,C_{\text{out}},l} = \sum_{c_{\text{in}}=0}^{C_{\text{in}}-1} \sum_{k=0}^{K-1} X_{n,l+k,c_{\text{in}}} W_{k,c_{\text{in}},c_{\text{out}}} \tag{1.2}$$

Conv2d: 通常对于图像或二维特征图，给定输入 $X \in \mathbb{R}^{N \times H \times W \times C_{\text{in}}}$，filter $W \in \mathbb{R}^{K_H \times K_W \times C_{\text{in}} \times C_{\text{out}}}$，其中 C_{in} 为输入通道数，C_{out} 为输出通道数，K_H, K_W 为卷积核的高度和宽度，卷积操作为:

$$Y_{n,h,w,o} = \sum_{c_{\text{in}}=0}^{C_{\text{in}}-1} \sum_{m=0}^{K_H-1} \sum_{n=0}^{K_W-1} X_{n,h+m,w+n,c_{\text{in}}} W_{m,n,c_{\text{in}},c_{\text{out}}} \tag{1.3}$$

其中，$Y \in \mathbb{R}^{N \times H_{\text{out}} \times W_{\text{out}} \times C_{\text{out}}}$，$h, w$ 为空间位置索引，o 为输出通道索引

池化层类型包括：最大池化、平均池化。

$$\begin{aligned} A_{i,j} &= \max_{m,n}(a, Y_i, j) \\ A_{i,j} &= \frac{1}{M * N} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} Y_{i+m,j+n} \end{aligned} \tag{1.4}$$

```python
import torch.nn as nn
conv = nn.Conv1d(in_channels, out_channels, kernel_size)
conv = nn.Conv2d(in_channels, out_channels, kernel_size)
pool = nn.MaxPool2d(kernel_size, stride)
pool = nn.AvgPool2d(kernel_size, stride)
```

1.3.2 Depthwise Separable Convolution

Depthwise Separable Convolution [1] 是一种轻量级卷积操作，其包括两个步骤: Depthwise Convolution 和 Pointwise Convolution.

Depthwise Convolution 对每一个通道单独使用卷积核 filter.对于输入 $X \in \mathbb{R}^{N \times H \times W \times C_{\text{in}}}$，filter $W \in \mathbb{R}^{K_H \times K_W \times C_{\text{in}}}$，卷积操作为:

$$Y_{n,h,w,c} = \sum_{m=0}^{K_H-1} \sum_{n=0}^{K_W-1} X_{n,h+m,w+n,c} W_{m,n,c} \tag{1.5}$$

Pointwise Convolution 使用 1×1 的卷积核，对每一个通道进行卷积操作。对于输入 $X \in \mathbb{R}^{N \times H \times W \times C_{\text{in}}}$，filter $W \in \mathbb{R}^{1 \times 1 \times C_{\text{in}} \times C_{\text{out}}}$，卷积操作为:

$$Y_{n,h,w,o} = \sum_{c=0}^{C_{\text{in}}-1} X_{n,h,w,c} W_{0,0,c,o} \tag{1.6}$$

Depthwise Separable Convolution 结合以上两种卷积方式，首先使用 Depthwise Convolution ， 然后使用 Pointwise Convolution.

```py
import torch
import torch.nn as nn

depthwise = nn.Conv2d(in_channels, in_channels, kernel_size,
                      groups=in_channels)
pointwise = nn.Conv2d(in_channels, out_channels, kernel_size=1)
deepwise_separable = pointwise(depthwise(input))
```

1.4 State-Space model

State-Space model (SSM) 是用于描述时间序列数据的模型。对于任意时间序列输入 $u(t)$，SSM 首先将其映射到 hidden space $x(t)$，然后进一步映射为输出空间 $y(t)$:

$$u(t) \mapsto x(t) \mapsto y(t) \tag{1.1}$$

SSM 以以下形式表示:

$$\begin{aligned} x'(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{1.2}$$

解为:

$$y(t) = \sum_{n=0}^{t} (CA^{t-n}B + D\delta(t-n))u(n) \tag{1.3}$$

其中，$\delta(t-n)$ 为 Kronecker delta 函数。

1.5 Fourier Transform

1.5.1 思想

傅里叶变换的基本思想是：利用无穷个不同频率周期函数的线性组合来表示一个非周期函数。即:

$$f(t) = \sum_i a_i f_i(t) \tag{1.1}$$

最简单的周期函数为圆周运动，根据欧拉公式，我们可以得到:

$$e^{i\omega t} = \cos(\omega t) + i\sin(\omega t) \tag{1.2}$$

其中 ω 表示旋转速度，正数时为逆时针旋转，负数时为顺时针旋转。圆周运动的频率为 $T = (2\pi)/\omega$. 同时注意到：旋转整数倍周期后回到原点，即

$$\int_0^{nT} e^{i\omega t} dt = 0 \tag{1.3}$$

为了计算方便，我们可以令所有的周期都是 $2\pi/\omega$ 的整数倍，即:

$$f(t)=\sum_{-\infty}^{+\infty}c_k e^{ik\omega_0 t} \tag{1.4}$$

这样一来，我们设定的是正交基:

$$[...\ \ e^{-2i\omega_0 t}\ \ e^{-i\omega_0 t}\ \ 1\ \ e^{i\omega_0 t}\ \ e^{2i\omega_0 t}\ \ ...] \tag{1.5}$$

两边同乘 $e^{i-n\omega_0 t}$ 并积分:

$$\begin{aligned}\int_0^T f(t)e^{-in\omega_0 t}dt&=\sum_{-\infty}^{+\infty}c_k\int_0^T e^{i(k-n)\omega_0 t}dt\\&=Tc_n\end{aligned} \tag{1.6}$$

因此:

$$c_n=\frac{1}{T}\int_0^T f(t)e^{-in\omega_0 t}dt \tag{1.7}$$

这里的 c_n 为不同角频率的圆周运动的系数

1.5.2 傅里叶变换

对于一个连续信号 $f:\mathbb{R}^d\to\mathbb{C}$，其连续傅里叶变换 (CFT) 为 $\mathcal{F}:\mathbb{R}^d\to\mathbb{C}$:

$$\mathcal{F}(f)(\boldsymbol{K})=\int_{\mathbb{R}^d}f(\boldsymbol{x})e^{-2\pi i\boldsymbol{K}\cdot\boldsymbol{X}}d\boldsymbol{X} \tag{1.8}$$

同时，可以进行逆变换:

$$\mathcal{F}^{-1}(f)(\boldsymbol{X})=\int_{\mathbb{R}^d}f(\boldsymbol{K})e^{2\pi i\boldsymbol{K}\cdot\boldsymbol{X}}d\boldsymbol{K} \tag{1.9}$$

对于不连续点序列 $\{x[n]\}_{0\le n\le N}$，其离散傅里叶变换 (DFT) 为:

$$\mathcal{F}x[n]=\sum_{n=0}^{N-1}x[n]e^{-2\pi ink/N},\quad k=0,1,...,N-1 \tag{1.10}$$

同理，可以进行逆变换:

$$\mathcal{F}^{-1}x[k]=\frac{1}{N}\sum_{k=0}^{N-1}x[k]e^{2\pi ink/N},\quad n=0,1,...,N-1 \tag{1.11}$$

同时，可以通过矩阵乘法表示:

$$\mathcal{F}x=W\cdot x \tag{1.12}$$

其中 W 为 DFT 矩阵，$\left\{W_{i,j}=e^{-2\pi ink/N}\right\}_{n,j=0,...,N-1}$:

$$W=\frac{1}{\sqrt{N}}\begin{bmatrix}1&1&1&...&1\\1&e^{-2\pi i1/N}&e^{-2\pi i2/N}&...&e^{-2\pi i(N-1)/N}\\1&e^{-2\pi i2/N}&e^{-2\pi i4/N}&...&e^{-2\pi i2(N-1)/N}\\...&...&...&...&...\\1&e^{-2\pi i(N-1)/N}&e^{-2\pi i2(N-1)/N}&...&e^{-2\pi i(N-1)^2/N}\end{bmatrix} \tag{1.13}$$

其中 $1/\sqrt{N}$ 为归一化系数，使得 W 具有以下性质:

$$W^{-1} \cdot W = I \tag{1.14}$$

1.5.3 卷积和卷积定理

时域上的卷积等价于频域上的乘积，若 $\mathcal{F}[f(t)] = f(\omega), \mathcal{F}[g(t)] = g(\omega)$，则:

$$\begin{aligned} \mathcal{F}[f(t) * g(t)] &= F(\omega)G(\omega) \\ \mathcal{F}[f(t)g(t)] &= \frac{1}{2\pi}F(\omega) * G(\omega) \end{aligned} \tag{1.15}$$

注意: 这里的乘法不是矩阵的点乘，而是 element-wise 的乘法。即对于函数来说，每一点值的乘积作为新的函数值。

第二章 贝叶斯

贝叶斯涉及以下组件:

似然函数（likelihoo）: 表示观测数据在参数 θ 给定情况下的概率，通常记作 $p(D|\theta)$，其中 D 为观测数据

先验分布（prior distribution）: 表示在没有观测数据时对参数 θ 的信念，记作 $p(\theta)$.

后验分布（posterior distribution）: 表示在观测数据更新后参数分布，记作 $p(\theta|D)$，通常由贝叶斯定理进行计算

2.1 基础

2.1.1 贝叶斯优点

贝叶斯的基础形式为:

$$p(y|x) = \frac{p(x,y)}{p(x)} = \frac{p(x|y)p(y)}{p(x)} = \frac{p(x|y)p(y)}{\int p(x|y)p(y)dy} \tag{2.1}$$

即:

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}} \tag{2.2}$$

考虑一系列观测数据 $X = (x_1, x_2, ..., x_n)$，i.i.d.来自某个分布 $p(x|\theta)$，其中 θ 为参数。我们希期通过观测数据来估计参数 θ，即获得 $p(\theta|X)$.通常情况下我们可以利用 MLE 进行处理:

$$\theta_{\text{MLE}} = \arg\max_{\theta} p(X|\theta) = \arg\max_{\theta} \sum_i \log p(x_i|\theta) \tag{2.3}$$

如果利用贝叶斯方法，我们可以得到:

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)d\theta} \overset{iid}{=} \frac{\prod_i p(x_i|\theta)p(\theta)}{\int \prod_i p(x_i|\theta)p(\theta)d\theta} \tag{2.4}$$

这里的性质在于，使用贝叶斯方法得到的后验概率分布 $p(\theta|X)$ 包括了观测数据的信息，这样当我们有新的观测数据时，可以直接利用后验概率分布来估计参数，例如:

$$p(\theta|X, x_{n+1}) = \frac{p(x_{n+1}|\theta)p(\theta|X)}{p(x_{n+1}|X)} \overset{iid}{=} \frac{p(x_{n+1}|\theta)p(\theta|X)}{p(x_{n+1})} \tag{2.5}$$

贝叶斯的优点在于: 无论数据大小，都可以得到后验概率分布，这样可以避免过拟合问题。

2.1.2 Probabilistic ML model

判别式概率模型，Discriminative probabilistic ML model，用于分类和回归等任务。其特点是根据条件概率 $p(y|x, \theta)$ 进行建模，而不是通过联合概率分布 $p(x, y)$. 即，根据 x 预测 y。通常假设 θ 的先验分布与 x 无关，因此有:

$$p(y, \theta|x) = p(y|x, \theta)p(\theta) \tag{2.6}$$

在这里，$p(y|x, \theta)$ 是对与模型的选择，即函数 $y = f(x, \theta)$.

生成式概率模型，Generative probabilistic ML model，则是可以根据联合概率分布 $p(x, y, \theta)$ 进行建模，最终要获得的是 $p(x, y|\theta)$，即

$$p(x, y, \theta) = p(x, y|\theta)p(\theta) \tag{2.7}$$

贝叶斯模型，假设训练数据 (X_{tr}, Y_{tr}) 和一个判别式模型 $p(y, \theta|x)$，我们可以通过贝叶斯方法来估计参数 θ，在训练阶段，我们的 θ 是由训练数据 (X_{tr}, Y_{tr}) 估计得到的，即 $p(\theta|X_{tr}, Y_{tr})$. 根据贝叶斯定理:

$$\begin{aligned} p(\theta|X_{tr}, Y_{tr}) &= \frac{p(X_{tr}, Y_{tr}, \theta)}{p(X_{tr}, Y_{tr})} \\ &= \frac{p(Y_{tr}|X_{tr}, \theta)p(X_{tr}|\theta)p(\theta)}{\int p(Y_{tr}|X_{tr}, \theta)p(X_{tr}|\theta)p(\theta)d\theta} \\ \text{given: } \ p(X_{tr}|\theta) = P(X_{tr}) &= \frac{p(Y_{tr}|X_{tr}, \theta)p(\theta)}{\int p(Y_{tr}|X_{tr}, \theta)p(\theta)d\theta} \end{aligned} \tag{2.8}$$

通过训练，我们获得了后验分布 $p(\theta|X_{tr}, Y_{tr})$. 在测试阶段，加入新数据点 x，此时我们可以通过后验分布 $p(\theta|X_{tr}, Y_{tr})$ 来估计 y 的概率分布:

$$p(y|x, X_{tr}, Y_{tr}) = \int p(y|x, \theta)p(\theta|X_{tr}, Y_{tr})d\theta \tag{2.9}$$

这是对所有的模型 θ 进行平均，其中 $p(y|x, \theta)$ 代表每个模型（由 θ 表示）的预测，而 $p(\theta|X_{tr}, Y_{tr})$ 代表这些模型的不确定性，衡量我们对不同参数的信心。

2.1.3 Conjugate distribution

在贝叶斯模型中， Eq. (2.8) 和 Eq. (2.9) 都存在积分计算，在大部分情况下是难以直接获得数值解的。但共轭分布（Conjugate distribution）可以简化这种计算。

共轭分布是指: 对于先验分布 $p(\theta)$、似然函数 $p(X|\theta)$和后验分布 $p(\theta|X)$, 若先验分布和后验分布属于同一分布族（distribution family），则称 $p(\theta)$ 和 $p(X|\theta)$ 为共轭分布。即:

$$p(\theta) \in \mathcal{A}(\alpha), p(X|\theta) \in \mathcal{B}(\beta) \Rightarrow p(\theta|X) \in \mathcal{A}(\alpha') \tag{2.10}$$

这样的好处在于，我们可以直接获得后验分布 $p(\theta|X)$ 的形式，从而可以忽略积分的过程，例如:

$$p(\theta|X) = \frac{p(\theta)p(X|\theta)}{\int p(\theta)p(X|\theta)d\theta} \tag{2.11}$$

我们知道 $p(\theta|X)$ 的函数形式是与 $p(\theta)$ 相同的，即确保了 $\int p(\theta|X)d\theta = 1$. 因此我们可以忽略积分，得到:

$$p(\theta|X) \propto p(\theta)p(X|\theta) \tag{2.12}$$

接着只需要计算参数即可。

常见的共轭分布:

Likelihood $p(x\|\theta)$	θ	Conjugate prior $p(y)$
Gaussian	μ	Gaussian
Gaussian	σ^{-2}	Gamma
Gaussian	(μ, σ^{-2})	Gaussian-Gamma
Multivariate Gaussian	Σ^{-1}	Wishart
Bernoulli	p	Beta
Multinomial	$(p_1, ..., p_m)$	Dirichlet
Poisson	λ	Gamma
Uniform	θ	Pareto

共轭分布通常只适用于简单概率模型。

2.1.4 Maximum posterior estimation

当共轭分布不可用时，一种简单的方法是使用最大后验估计（maximum a posteriori probability estimate, MAP）.其思想是将分布估计转变为点估计，将参数取为后验分布的最大值，即:

$$\begin{aligned}\theta_{MAP} &= \arg\max_{\theta} p(\theta|X_{tr}, Y_{tr}) \\ &= \arg\max_{\theta} \frac{p(Y_{tr}|X_{tr}, \theta)p(\theta)}{\int p(Y_{tr}|X_{tr}, \theta)p(\theta)d\theta} \\ &\int p(Y_{tr}|X_{tr}, \theta)p(\theta)d\theta \text{ does not depend on } \theta \\ &= \arg\max_{\theta} p(Y_{tr}|X_{tr}, \theta)p(\theta)\end{aligned} \tag{2.13}$$

鉴于 θ_{MAP} 为点估计值，此时测试阶段则转变为:

$$p(y|x, X_{tr}, Y_{tr}) = p(y|x, \theta_{MAP}) \tag{2.14}$$

2.2 Variational inference

2.2.1 KL-divergence

变分推断（Variational inference）是一种近似推断方法，旨在处理复杂的后验分布。其基本思想是，将后验分布 $p(\theta|X)$ 近似为一个简单的分布 $q(\theta) \in \mathcal{Q}$，使得 $q(\theta)$ 尽可能接近 $p(\theta|X)$. 通常情况下，我们可以通过最小化两个分布的 KL-divergence 来获得 $q(\theta)$，即:

$$q(\theta) = \arg\min_{q \in \mathcal{Q}} F(q) := KL(q(\theta) \| p(\theta|X)) \tag{2.1}$$

其中，KL-divergence 为:

$$KL(q(\theta) \| p(\theta|X)) = \int q(\theta) \log\left(\frac{q(\theta)}{p(\theta|X)}\right) d\theta \tag{2.2}$$

注意:

1. KL-divergence 是非负的，当且仅当 $q(\theta) \equiv p(\theta|X)$ 时，KL-divergence 为 0
2. KL-divergence 不对称，即 $KL(q(\theta) \| p(\theta|X)) \neq KL(p(\theta|X) \| q(\theta))$
3. KL-divergence 包含的两个分布必须是相同的支持集，即 $q(\theta)$ 和 $p(\theta|X)$ 必须在相同的空间上定义

> 支撑集(support): 它是集合X的一个子集，要求对给定的X上定义的实值函数f在这个子集上恰好非0. 特别地，在概率论中，一个概率分布是随机变量的所有可能值组成的集合的闭包。

两者必须拥有相同的信息量

存在两个问题:

1. 未知的后验分布 $p(\theta|X)$ 导致 KL-divergence 无法计算
2. KL-divergence 的优化空间为分布空间，通常情况下是无法直接优化的

2.2.2 Evidence lower bound

证据下界（Evidece lower bound, ELOB）用于解决问题 1. 我们知道后验分布的贝叶斯形式为:

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)d\theta} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}} \tag{2.3}$$

考虑 $\log p(X)$，我们可以做出以下变形:

$$\begin{aligned}
\log p(X) &= \int q(\theta) \log(p(X)) d\theta = \int q(\theta) \log \frac{p(X,\theta)}{p(\theta|X)} d\theta \\
&= \int q(\theta) \log \frac{p(X,\theta)q(\theta)}{(p(\theta|X))q(\theta)} d\theta \\
&= \int q(\theta) \log \frac{p(X,\theta)}{q(\theta)} d\theta + \int q(\theta) \log \frac{q(\theta)}{p(\theta|X)} d\theta \\
&= \mathcal{L}(q(\theta)) + KL(q(\theta) \| p(\theta|X))
\end{aligned} \tag{2.4}$$

其中，$\mathcal{L}(q(\theta))$ 为证据下界（Evidence lower bound, ELBO），即:

$$\log p(X) \geq \mathcal{L}(q(\theta)) \tag{2.5}$$

对于 Eq. (2.4)，我们注意到：$\log p(x)$ 与 $q(\theta)$ 无关，而 $\mathcal{L}(p(\theta))$ 和 $KL(q(\theta)\|p(\theta|X))$ 与 $q(\theta)$ 有关。因此，我们可以通过最大化 $\mathcal{L}(q(\theta))$ 来最小化 $KL(q(\theta)\|p(\theta|X))$，即:

$$\begin{aligned} q(\theta) &= \arg\min_{q\in\mathcal{Q}} KL(q(\theta)\|p(\theta|X)) \\ &= \arg\max_{q\in\mathcal{Q}} \mathcal{L}(q(\theta)) \\ &= \arg\max_{q\in\mathcal{Q}} \int q(\theta)\log\frac{p(X,\theta)}{q(\theta)}d\theta \end{aligned} \tag{2.6}$$

其中 $\mathcal{L}(q(\theta))$ 中分布都是已知可以计算的，进一步我们可以得到:

$$\begin{aligned} \mathcal{L}(q(\theta)) &= \int q(\theta)\log\frac{p(X,\theta)}{q(\theta)}d\theta = \int q(\theta)\log\frac{p(X|\theta)p(\theta)}{q(\theta)}d\theta \\ &= \int q(\theta)\log p(X|\theta)d\theta + \int q(\theta)\log\frac{p(\theta)}{q(\theta)}d\theta \\ &= \mathbb{E}_{q(\theta)}\log p(X|\theta) - KL(q(\theta)\|p(\theta)) \end{aligned} \tag{2.7}$$

对于第一项 $\mathbb{E}_{q(\theta)}\log p(X|\theta)$，我们需要其最大化。即使取 $\log p(X|\theta)$ 的加权平均值，其收敛点仍然与 MLE 一致：将 $q(\theta)$ 设置在集中于 MLE 点估计的位置，有 $\hat{\theta} = \arg\max_\theta p(X|\theta)$，此时参数对应的似然函数 $p(X|\theta)$ 取最大值，有:

$$\mathbb{E}_{q(\theta)}\log p(X|\theta) \to \log p(X|\theta) \tag{2.8}$$

这告诉我们最大化第一项会导致参数 θ 逐渐收敛到 θ_{MLE}. 同时这一项被叫作 data term，当模型的对数似然性很高时，意味着模型在给定的参数下生成观察数据 X 的概率大于在其他参数下生成数据的概率。因此，可以说模型与观察数据的拟合程度较好

第二项被称为 regularizer，可以防止模型的过拟合。

2.2.3 Mean field approximation

对于 $q(\theta)$ 的选择，我们可以使用均场近似（mean field approximation）来简化问题。均场近似假设 $q(\theta)$ 可以分解为一系列独立的分布，即:

$$q(\theta) = \prod_{i=1}^{m} q_i(\theta_i) \tag{2.9}$$

均场近似的思想是在每一步中，固定参数 $\{q_i(\theta_i)\}_{i\neq j}$，只对单一参数 $q_j(\theta_j)$ 做优化，其中参数 θ_i 仍然可以为向量，因此我们的目标转化为:

$$q_j(\theta_j) = \arg\max_{q_j(\theta_j)} \mathcal{L}(q(\theta)) \tag{2.10}$$

此时将固定参数视为常量，对 $q_j(\theta_j)$ 做解析解，可以得到:

$$
\begin{aligned}
\mathcal{L}(q(\theta)) &= \int q(\theta)\log\frac{p(X,\theta)}{q(\theta)}d\theta = \int q(\theta)\log p(X,\theta)d\theta - \int q(\theta)\log q(\theta)d\theta \qquad (2.11)\\
&= \int\prod_i q_i(\theta_i)\log p(X,\theta)\prod_i d\theta_i - \int\prod_i q_i(\theta_i)\log\prod_i q_i(\theta_i)\prod_i d\theta_i\\
&= \int q_j(\theta_j)\int\prod_{i\neq j} q_i(\theta_i)\log p(X,\theta)\prod_i d\theta_i - \int q_j(\theta_j)\int\prod_{i\neq j} q_i(\theta_i)\log\prod_i q_i(\theta_i)\prod_i d\theta_i\\
&= \mathbb{E}_{q_j(\theta_j)}\Big[\mathbb{E}_{q_{i\neq j}}\log p(X,\theta)\Big] - \int q_j(\theta_j)\int\prod_{i\neq j} q_i(\theta_i)\left(\log q_j(\theta_j) + \log\prod_{i\neq j} q_i(\theta_i)\right)\prod_i d\theta_i\\
&= \mathbb{E}_{q_j(\theta_j)}\Big[\mathbb{E}_{q_{i\neq j}}\log p(X,\theta)\Big] - \int q_j(\theta_j)\log q_j(\theta_j)\underbrace{\int\prod_{i\neq j} q_i(\theta_i)\prod_i d\theta_i}_{=1} - \underbrace{\int q_j(\theta_j)}_{=1}\int\prod_{i\neq j} q_i(\theta_i)\log\prod_{i\neq j} q_i(\theta_i)\prod_i d\theta_i\\
&= \mathbb{E}_{q_j(\theta_j)}\Big[\mathbb{E}_{q_{i\neq j}}\log p(X,\theta)\Big] - \int q_j(\theta_j)\log q_j(\theta_j)d\theta_j - \int\prod_{i\neq j} q_i(\theta_i)\sum_{i\neq j}\log q_i(\theta_i)\prod_{i\neq j} d\theta_i\\
&= \mathbb{E}_{q_j(\theta_j)}\Big[\mathbb{E}_{q_{i\neq j}}\log p(X,\theta) - \log q_j(\theta_j)\Big] - \text{constant}
\end{aligned}
$$

引入 Lagrangian 函数计算其最小值:

$$
\partial_{q_j(\theta_j)}\mathcal{L}(q(\theta)) + \sum_i \lambda_i\left(\int p_i(\theta_i)d\theta_i - 1\right) = 0 \qquad (2.12)
$$

$$
0 = \mathbb{E}_{q_{i\neq j}}\log p(X,\theta) - \partial_{q_j(\theta_j)}\mathbb{E}_{q_j(\theta_j)}\left[\log q_j(\theta_j)\right] + \partial_{q_j(\theta_j)}\lambda_j\int p_j(\theta_j)d\theta_j
$$

利用变分法解决求导:

$$
\begin{aligned}
\partial_{q_j(\theta_j)}\mathbb{E}_{q_j(\theta_j)}\left[\log q_j(\theta_j)\right] &= \partial_{q_j(\theta_j)}\int q_j(\theta_j)\log q_j(\theta_j)d\theta_j\\
&= \partial_{q_j(\theta_j)} q_j(\theta_j)\log q_j(\theta_j)\\
&= \log q_j(\theta_j) + 1\\
\partial_{q_j(\theta_j)}\lambda_j\int p_j(\theta_j)d\theta_j &= \lambda_j
\end{aligned} \qquad (2.13)
$$

因此， Eq. (2.12) 改写为:

$$
\begin{aligned}
0 &= \mathbb{E}_{q_{i\neq j}}\log p(X,\theta) - \log q_j(\theta_j) - 1 + \lambda_j\\
&= \mathbb{E}_{q_{i\neq j}}\log p(X,\theta) - \log q_j(\theta_j) + \text{constant}
\end{aligned} \qquad (2.14)
$$

得到:

$$
q_j(\theta_j) = \frac{1}{Z_j}\exp\Big(\mathbb{E}_{q_{i\neq j}}\log p(X,\theta)\Big) \qquad (2.15)
$$

其中 Z_j 为归一化常数，使得 $q_j(\theta_j)$ 满足概率分布的性质。

因此， Mean field approximation 的算法为:

1. 初始化:

$$q(\theta) = \prod_i q_i(\theta_i) \tag{2.16}$$

2. 重复，直到 ELBO 收敛:
   - 对于每一个 $q_i(\theta_i)$，做如下计算更新:

$$q_j(\theta_j) = \frac{1}{Z_j} \exp\Big(\mathbb{E}_{q_{i\neq j}} \log p(X, \theta)\Big) \tag{2.17}$$

   或

$$\log q_{j(\theta_j)} = \mathbb{E}_{q_{i\neq j}} \log p(X, \theta) + \text{constant} \tag{2.18}$$

   - 重新计算 ELBO:

$$\mathcal{L}(q(\theta)) \tag{2.19}$$

问题在于如何计算 Z_j 与期望 $\mathbb{E}_{q_{i\neq j}} \log p(X, \theta))$ 是否能够被解析解计算出来。如果要确保其能够被计算出来，需要假设 $\theta \to [\theta_1, ..., \theta_m]$ 的过程具有共轭性，即:

$$\begin{aligned} \forall \theta_j, \quad p\Big(\theta_j | \theta_{i\neq j}\Big) \in \mathcal{A}\big(\alpha_j\big), \quad p\Big(x |\ \theta_j, \theta_{i\neq j}\Big) \in \mathcal{B}\big(\beta_j\big) \\ \to p\Big(\theta_j | X, \theta_{i\neq j}\Big) \in \mathcal{A}(\alpha') \end{aligned} \tag{2.20}$$

在实际操作中，可以这样对共轭性进行检验:

对于每一个 θ_j:
- 固定 $\{\theta_i\}_{i\neq j}$ （将其视为常数）
- 检查 $p(X|\theta)$ 与 $p(\theta)$ 是否对于 θ_j 是共轭的

2.3 Dirichlet process mixture model

2.3.1 Dirichlet process

Dirichlet process 是一种用于非参数贝叶斯统计的随机过程，可以创建无限个连续分布 H 的离散副本，即 $H \mapsto G$. 记作 $G \sim DP(\alpha, H)$，其中 α 为浓度参数，决定了聚类的程度。浓度参数越大，生成的簇的数量通常越多。

Dirichlet process 可以由 stick-breaking process 来描述。假设我们有一个长度为 1 的棍子，我们从棍子的一端开始，每次从棍子的长度中折断一部分，折断的长度服从 beta 分布，折断的位置服从 beta 分布。这样我们可以得到一个无限个分布的序列:

1. 生成一个无限长序列 $V_k \sim \mathrm{Beta}(1, \alpha) \in (0, 1)$
2. 生成一个无限长序列权重 π_k:

$$\pi_k = V_k \prod_{j=1}^{k-1}(1 - V_j), \sum_{k=1}^{\infty} \pi_k = 1 \tag{2.1}$$

3. 从连续分布 H 中抽取无限多个样本 θ_k，构成新的分布 G:

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}, \delta_{\theta_k} = \delta(\theta - \theta_k) \tag{2.2}$$

Dirichlet process 有以下性质:

1. 期望不变:

$$\mathbb{E}_{DP(\alpha, H)}[x] = \mathbb{E}_H[x] \tag{2.3}$$

2.

$$\alpha \to \infty \Rightarrow DP(\alpha, H) = H \tag{2.4}$$

3. 序列为无限长，无法完全在计算机中表达

2.3.2 Mixture model

混合模型（Mixture Model）是一种统计模型，它假设数据来自多个不同的分布，每个分布被称为一个“成分”(component)。混合模型的主旨在于通过这些成分的组合来更好地描述数据的总体分布。每个成分可以用不同的概率分布函数来定义，如高斯分布、伯努利分布等。

对于给定的观察值 x，混合模型的概率密度函数可以表示为:

$$p(x) = \sum_{k=1}^{K} \pi_k p(x|\theta_k), \sum_{k=1}^{K} \pi_k = 1 \tag{2.5}$$

混合模型广泛应用于许多领域，包括但不限于:

- 聚类: 通过对数据进行分组，帮助识别数据中的模式和结构。例如，K 均值聚类可以看作是高斯混合模型的一个特例。
- 密度估计: 能够捕捉到复杂的分布形状，比单一的概率分布（如正态分布）更具灵活性。

- 信号处理: 在音频和图像处理等领域中，用于建模混叠信号。
- 生物信息学: 在基因表达数据分析中识别不同的生物状态。

考虑利用核混合模型（Kernel Mixture Model）估计 pdf，我们可以初步写为:

$$f(y|P) = \int \mathcal{K}(y|\theta)dP(\theta) \tag{2.6}$$

其中，$\mathcal{K}(\cdot\,|\theta)$ 为核函数，以 θ 作为其参数，可能包括每个核的中心位置、宽度等；P 是混合测度。

> 测度（measure）: 测度是一个函数，它将集合映射到实数。常见的测度包括: 长度、面积、概率测度等。假设样本空间 Ω，一个概率测度 P 满足以下条件:
> - $P(A) \geq 0, \forall A \in \Omega$
> - $P(\Omega) = 1$
> - 对于不相交事件 $A_1, A_2, ...$，有 $P\left(\bigcup_i A_i\right) = \sum_i P(A_i)$
>
> 混合测度（mixture measure）: 混合测度是指在混合模型中，表示由多种不同的分布组合而成的分布特征。混合测度可以写作:
>
> $$P = \int P_\theta dH(\theta) \tag{2.7}$$
>
> 其中: P_θ 是给定参数的成分分布，例如可以是正态分布、指数分布等; H 是先验测度，描述参数 θ 的分布。

因此，Eq. (2.6) 计算了在参数空间中，对于每个 θ 的核函数的加权求和。

2.3.3 DP for mixutre model

DP 适合用作为未知混合分布的先验。例如在 Eq. (2.6) 中，考虑其为无限核混合模型，混合测度 P 视为未知。此时，我们可以令 $P \sim \pi_{\mathcal{P}}$，其中 $\mathcal{P}$ 代表在样本空间中所有可能的概率测度，$\pi_{\mathcal{P}}$ 则代表样本空间中的先验分布。考虑将 $\pi_{\mathcal{P}}$ 选做 DP 的先验，这样我们可以获得一个离散的 DP 混合模型:

$$f(y) = \sum_{k=1}^{\infty} \pi_k \mathcal{K}(y|\theta_k) \tag{2.8}$$

其中 π_i 来自 DP 的权重（参数为 α），且:

$$y_i \sim \mathcal{K}(\theta_i), \quad \theta_i \sim P, \quad P \sim DP(\alpha, P_0) \tag{2.9}$$

采用DP 作为 P 的先验，会导致后验计算变得复杂。根据 Eq. (2.7) 我们可知，以 DP 作为先验的混合模型，其混合测度拥有无限个参数，因此在拥有样本 $y^n = (y_1, y_2, ..., y_n)$ 的条件下无法直接获得 P 的后验分布。解决方法是通过边缘化 P 来获得参数 $\theta^n = (\theta_1, \theta_2, ..., \theta_n)$ 的先验分布。具体的，我们可以用 Polya urn 预测规则来描述这种情形。即:

$$p(\theta_i|\theta_1, ..., \theta_{i-1}) \sim \left(\frac{\alpha}{\alpha + i - 1}\right) P_0(\theta_i) + \sum_{j=1}^{i-1} \delta_{\theta_j} \tag{2.10}$$

从 Eq. (2.10) 开始进行聚类，假设有 n 个样本，k 个簇，n_k 代表第 k 个簇的样本数量，θ_k 代表第 k 个簇的参数，我们有:

$$p(\theta_i|\theta_{-i}) = \underbrace{\left(\frac{\alpha}{\alpha+n-1}\right)P_0(\theta_i)}_{\text{新建聚类}} + \underbrace{\sum_{h=1}^{k^{(-i)}} \overbrace{\left(\frac{n_h^{(-i)}}{\alpha+n-1}\right)}^{\text{DP 中聚类 h 的权重/占比}} \delta_{\theta_h^{*(-i)}}}_{\text{选择已有聚类}} \tag{2.11}$$

其中 $\theta_h^*, h = 1, ..., k^{(-i)}$ 是 θ_{-i} 中的唯一值，代表了在移除第 i 个样本后剩下的唯一聚类参数；$n_h^{(-i)} = \sum_{j \neq i} 1_{\theta_j = \theta_h^*}$ 代表除去第 i 个样本后，第 h 个簇的样本数量。

2.3.4 DPMM with Gibbs sampling

Gibbs sampling 允许我们从多维分布中抽样，通过迭代更新每个维度的样本。对于 DPMM，我们可以通过 Gibbs sampling 来优化 Eq. (2.11):

令 $\theta^* = (\theta_1^*, ..., \theta_k^*)$ 为参数 θ 的唯一值，且令 S_i 为第 i 个样本的聚类分配，即若 $\theta_i = \theta_c^*$ 则 $S_i = c$. Gibbs sampler 的步骤如下:

1. 通过从多项式条件后验中抽样来更新分配 S:

$$P(S_i = c|S_{i-1}, \theta^*, \alpha, P_0) \propto \begin{cases} n_c^{(-i)}\mathcal{K}(y_i|\theta_c^*), & c = 1, ..., k^{(-i)} \\ \alpha \int \mathcal{K}(y_i|\theta)dP_0(\theta), & c = k^{(-i)} + 1 \end{cases} \tag{2.12}$$

2. 通过从条件后验中抽样来更新参数 θ^*:

$$p(\theta_c^*| -) \propto P_0(\theta_c^*) \prod_{i:S_i=c} \mathcal{K}(y_i|\theta_c^*) \tag{2.13}$$

其中 $\prod_{i:S_i=c} \mathcal{K}(y_i|\theta_c^*)$ 是聚类 c 中所有样本在参数 θ_c^* 下的似然函数的乘积，反应了在当前聚类分配下，样本数据对于参数的支持程度。

聚类行为的控制因素:

1. 浓度参数 α: α 的大小直接影响聚类的数量。当 α 接近于 0 时，获取的聚类会趋向于集中，从而展现出一种共同参数 $y_i \sim \mathcal{K}(\theta)$ 的行为。相反，增大 α 则有更高的可能性去接受新聚类。
2. 先验 P_0 的方差: 高方差的先验 P_0 表示对聚类位置的不确定性，阻碍新聚类的形成。

2.4 Bayesian optimization

有些时候，我们不仅需要根据数据预计结果，还期望获得预计结果的不确定性。

在回归中，我们的目标是基于来自未知函数的观察数据点对函数进行建模。传统的非线性回归方法通常给出一个被认为最适合数据集的单一函数。然而，可能存在多个函数同样适合观察到的数据点。我们观察到，当多元正态分布的维度为无限时，我们可以使用这些无限数量的函数在任何点进行预测。当我们开始有观察数据时，我们不再保留无限数量的函数，而只保留适合观察数据的函数，形成后验分布。当我们有新的观察数据时，我们将当前的后验作为先验，并使用新的观察数据点来获得一个新的后验。[2]

2.4.1 Gaussian process

Gaussian distribution 具有良好的性质，对于多元高斯分布，其联合分布、条件分布、边缘分布都是高斯分布。例如，假设多元高斯分布:

$$p(f_1, f_2) \sim \mathcal{N}(f_1, f_2|\mu, \Sigma) \tag{2.1}$$

则:

$$\begin{gathered}
\begin{pmatrix} f_1 \\ f_2 \end{pmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\right) \\
p(f_1) \sim \mathcal{N}(f_1|\mu_1, \Sigma_{11}) \\
p(f_1|f_2) \sim \mathcal{N}(f_1|\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(f_2 - \mu_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21})
\end{gathered} \tag{2.2}$$

Gaussian process (GP) 假设我们观测的样本来自一个连续随机过程。对于每一个观测的样本与其对应的输出，我们假设输入 $X = \{x\}_{i=1}^m, x_i \in \mathbb{R}^d$，对应观测值 $f = \{f\}_{i=1}^m$，且满足:

$$p(f|X) = \mathcal{N}(\mu, K) \tag{2.3}$$

其中 f_i 为随机函数，其值为 $f(x_i)$，是我们观察到的、来自随机过程的值。μ 为均值函数，K 为协方差函数，因此还有:

$$\mu = \{\mu_i\} = \{\mu(x_i)\} \tag{2.4}$$

对于协方差函数 K，其本质是量化两个点之间距离的函数，例如核函数中的高斯核。在 GP 过程中有多种选择，例如:

$$\begin{aligned}
K &= \left\{K_{i,j}\right\} = \left\{K\left(x_i, x_j\right)\right\} \\
&= \frac{2}{\pi}\sin^{-1}\left(2\frac{x_i^\top \Sigma x_j}{\sqrt{\left(1 + 2x_i^\top \Sigma x_i\right)\left(1 + 2x_j^\top \Sigma x_j\right)}}\right)
\end{aligned} \tag{2.5}$$

初次以外，协方差函数 K 还可以组合使用:

- 相加: $K(x, x') = K_1(x, x') + K_2(x, x')$
- 相乘: $K(x, x') = K_1(x, x') \cdot K_2(x, x')$
- 卷积: $K(x, x') = \int K_1(x, z)K_2(z, x')dz$

此时，记作:

$$f(x) \sim \mathcal{GP}(\cdot\,|\mu(x), K(x, x')) \tag{2.6}$$

2.4.2 GP Regression

假设观测到数据点 $S_m = \{X, y\} = \{(x_i, y_i)\}_{i=1}^m$，$y$ 为噪音观测值:

$$y_i = f(x_i) + \varepsilon_i \tag{2.7}$$

其中 $f(\cdot) \sim \mathcal{GP}(\cdot\,|0, K)$ 为我们希望拟合的随机过程函数，$\varepsilon \sim \mathcal{N}(0, \sigma^2)$ 为高斯噪声。因此，先验为:

$$p(f) \sim \mathcal{N}(0, K) \tag{2.8}$$

根据我们的假设，y 服从多元正态分布:

$$p(y) \sim \mathcal{N}(0, K + \sigma^2 I) \tag{2.9}$$

因此，似然函数为:

$$p(y|f) = \mathcal{N}(y|f, \sigma^2 I) \tag{2.10}$$

假设测试输入值 x_*，我们希望预测其输出值 y_*，则:

$$y_* = f_* + \varepsilon_*, \quad f_* = f(x_*) \tag{2.11}$$

我们需要根据观测的 y 获得估计的 f_*，因此考虑此时联合分布，根据 Eq. (2.2):

$$p(y, f_*) \sim \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}K + \sigma^2 I & K_* \\ K_*^\top & K_{**}\end{bmatrix}\right) \tag{2.12}$$

其中 $K_* = \{K(x_*, x)\}_{i=1}^m, K_{**} = K(x_*, x_*)$. 同理，根据 Eq. (2.2)，我们可以得到 f_* 的条件分布，这就是我们的 GP Regression:

$$\begin{gathered} p(f_*|y) = \mathcal{N}(f_*|\mu_*, \Sigma_*) \\ \mu_* = K_*^\top (K + \sigma^2 I)^{-1} y,\ \Sigma_* = K_{**} - K_*^\top (K + \sigma^2 I)^{-1} K_* \end{gathered} \tag{2.13}$$

其中对于 μ_*，有 $K_* \in \mathbb{R}^m, (K + \sigma^2 I)^{-1} \in \mathbb{R}^{m\times m}, y \in \mathbb{R}^m$，因此 $\mu_* \in \mathbb{R}$. 同理，$\Sigma_* \in \mathbb{R}$. 因此，μ_* 可改写为:

$$\mu_* = \sum_{i=1}^m \alpha_i K(x_*, x_i), \quad \alpha = (K + \sigma^2 I)^{-1} y \tag{2.14}$$

GP Regression 的超参数在于协方差函数 K 的选择，以及噪声方差 σ^2 的选择。通常我们可以通过最大化似然函数来估计这些参数。考虑:

$$p(y) = \mathcal{N}(0, K + \sigma^2 I_m) \tag{2.15}$$

对数似然函数为:

$$\begin{aligned} \mathcal{L} = \log p(y) &= -\frac{1}{2} y^\top (K + \sigma^2 I_m)^{-1} y - \frac{1}{2}\log\det(K + \sigma^2 I_m) - \frac{m}{2}\log 2\pi \\ &= p(y|\theta) = -\frac{1}{2} y^\top C^{-1}(\theta) y - \frac{1}{2}\log\det(C(\theta)) - \frac{m}{2}\log 2\pi \end{aligned} \tag{2.16}$$

其中 $C(\theta) = K + \sigma^2 I_m$

第三章 Latent variable model

3.1 Laten variable model

潜变量模型 (Latent variable model) 是一种统计模型，其中包含了一些未观测的变量，这些变量通常被称为潜变量 (latent variable)。潜变量模型通常用于描述数据背后的潜在结构，以及数据生成的机制。潜变量模型可以用于多种任务，如聚类、降维、异常检测等。

对于观测数据 X 与其模型参数 θ，要估计模型参数 θ，通常采用 MLE 方法，即

$$\theta_{\text{MLE}} = \arg\max_{\theta} \log p(X|\theta) \tag{3.1}$$

我们假设存在某种潜在变量 Z，其与观测数据 X 之间存在关系，此时我们可以对 $\log p(X|\theta)$ 进行分解，类似 Eq. (2.4):

$$\begin{aligned} \log p(X|\theta) &= \int q(Z) \frac{p(X,Z|\theta)}{q(Z)} dZ + \int q(Z) \log \frac{q(Z)}{p(Z|X,\theta)} dZ \\ &= \mathcal{L}(q,\theta) + KL(q\|p) \geq \mathcal{L}(q,\theta) \end{aligned} \tag{3.2}$$

因此，我们可以通过最大化 ELBO $\mathcal{L}(q,\theta)$ 来使 $\log p(X|\theta)$ 尽可能大。对于 ELOB，给出其广泛定义 variational lower bound:

函数 $g(\xi, x)$ 是另一函数 $f(x)$ 的 variational lower bound，当且仅当:

- $\forall \xi, f(x) \geq g(\xi, x)$
- $\left[\forall x_0, \exists \xi(x_0) \Rightarrow f(x_0) = g(\xi(x_0), x_0)\right]$

例如过二次函数最低点的切线

这样，对于:

$$x = \arg\max_{x} f(x) \tag{3.3}$$

我们可以通过对 $g(\xi, x)$ 区块坐标更新 (Block-coordinate updates) 来获得 x 的近似解，即:

$$\begin{aligned} x_n &= \arg\max_{x} g(\xi_{n-1}, x) \\ \xi_n &= \xi(x_n) = \arg\max_{\xi} g(\xi, x_n) \end{aligned} \tag{3.4}$$

3.2 EM Algorithm

3.2.1 EM Algorithm

在最大化 $\log p(X|\theta)$ 时，我们不仅要最大化参数 θ，还要最大化潜变量 Z 的分布，即

$$\mathcal{L}(q,\theta) = \int q(Z) \frac{p(X,Z|\theta)}{q(Z)} dZ \to \max_{q,\theta} \tag{3.1}$$

E-step，设置一个初始点 θ_0，类似 Eq. (2.4):

$$
\begin{aligned}
q(Z) &= \arg\max_q \mathcal{L}(q, \theta_0) = \arg\min_q KL(q\|p) \\
&= p(Z|X, \theta_0) = \frac{p(X, Z|\theta_0)}{p(X|\theta_0)} = \frac{p(X, Z|\theta_0)}{\int_i p(X, z_i|\theta_0) dz_i}
\end{aligned} \tag{3.2}
$$

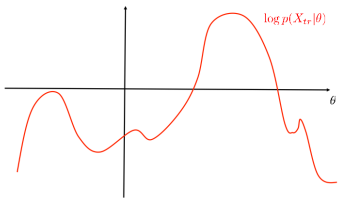
当 $p(Z|X, \theta_0)$ 无法获得解析解时，可以采取 variational inference 的方法。

M-step，考虑到 Z 的具体值不明确但知道其分布 $q(Z)$，我们采用其期望:

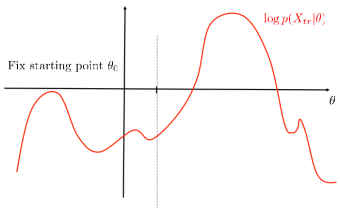
$$
\theta_* = \arg\max_\theta \mathcal{L}(q, \theta) = \arg\max_\theta \mathbb{E}_Z \log p(X, Z|\theta) \tag{3.3}
$$

将新的到的 θ_* 传入 E-step，重复直到收敛。在更新过程中 variational lower bound 是单调递增的，因此可以保证收敛

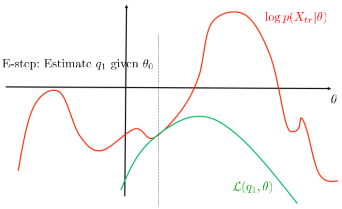
EM-algorithm



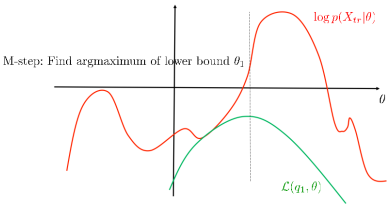
EM-algorithm



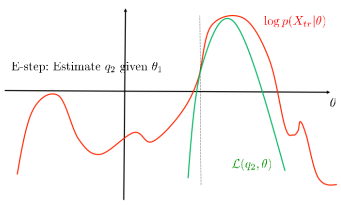
EM-algorithm



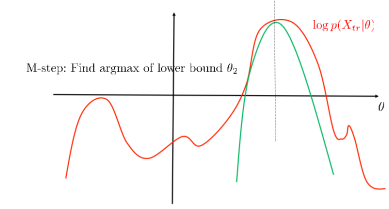
EM-algorithm



EM-algorithm



EM-algorithm



3.2.2 Categorical latent variables

对于离散型潜变量，假设 $z_i \in \{1, ..., K\}$，则

$$p(x_i|\theta) = \sum_k p(x_i, z_i = k|\theta) = \sum_k p(x_i|z_i = k, \theta)p(z_i = k|\theta) \tag{3.4}$$

进行 EM，E-step:

$$\begin{aligned} q(z_i = k) &= p(z_i = k|x_i, \theta) \\ &= \frac{p(x_i|z_i = k, \theta)p(z_i = k|\theta)}{p(x_i|\theta)} \\ &= \frac{p(x_i|z_i = k, \theta)p(z_i = k|\theta)}{\sum_l p(x_i|z_i = l, \theta)p(z_i = l|\theta)} \end{aligned} \tag{3.5}$$

M-step:

$$\begin{aligned} \theta_* &= \arg\max_\theta \mathbb{E}_Z \log p(X, Z|\theta) \\ &= \arg\max_\theta \sum_i \sum_k q(z_i = k) \log p(x_i, z_i = k|\theta) \end{aligned} \tag{3.6}$$

而对于连续型潜变量:

$$p(x_i|\theta) = \int p(x_i, z_i|\theta)dz_i = \int p(x_i|z_i, \theta)p(z_i|\theta)dz_i \tag{3.7}$$

E-step:

$$\begin{aligned} q(z_i) = p(z_i|x_i, \theta) &= \frac{p(z_i|x_i, \theta)}{p(x_i|\theta)} \\ &= \frac{p(x_i|z_i, \theta)p(z_i|\theta)}{\int p(x_i|z_i, \theta)p(z_i|\theta)dz_i} \end{aligned} \tag{3.8}$$

根据 Eq. (2.11)，只有当 $p(X|Z, \theta)$ 与 $p(Z|\theta)$ 为共轭分布时，E-step 才能获得解析解；否则，需要使用 stochastic variational inference 的方法

对于连续型潜变量，其重要应用之一在于 representation learning:

1. 表示学习的目标：Representation learning 的核心目标是生成有效的数据表示，而连续潜变量提供了一个强大的工具来建模数据的内在结构。
2. 潜变量在表示学习中的作用：通过引入连续潜变量，模型能够更灵活地捕捉数据的连续变化和模式，形成有效的表示。这些潜变量通常在隐藏层中起作用，影响最终输出的生成。
3. 生成模型中的应用：许多现代的生成模型，如 GAN（生成对抗网络）和 VAE，都利用连续潜变量来生成新样本，通过学习数据的潜在结构来提高生成能力。
4. 优化和推断：在 representation learning 的上下文中，涉及到从观测数据中推断潜变量的分布，并优化这些潜变量以获得更好的数据表示。连续潜变量可以利用梯度下降等优化方法进行推断。

3.3 VAE

3.3.1 Mixture PCA

在线性代数下视角的 PCA 一般涉及特征值分解与主成分投影，对于 n 个具有 p 维特征的数据 $\boldsymbol{X} \in \mathbb{R}^{n\times p}$，将其中心化后计算协方差矩阵:

$$\boldsymbol{\Sigma} = \frac{1}{n}\boldsymbol{X}^\top \boldsymbol{X} \tag{3.1}$$

然后对协方差矩阵进行特征值分解:

$$\boldsymbol{\Sigma}\boldsymbol{v}_j = \lambda_j \boldsymbol{v}_j, \quad j = 1, 2, ..., p \tag{3.2}$$

其中 $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_p \geq 0$ 为特征值，$\boldsymbol{v}_j$ 是对应的特征向量。选择前 k 个特征，将中心化后的数据投影到前 k 个特征向量上，得到降维的表示:

$$\begin{aligned} \boldsymbol{Z} &= \boldsymbol{X}\boldsymbol{V}_k \in \mathbb{R}^{n\times k} \\ \boldsymbol{V}_k &= [\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_k] \in \mathbb{R}^{p\times k} \end{aligned} \tag{3.3}$$

在 latent variable model 视角下，PCA 可以被视为一个潜变量模型，其中潜变量为降维后的数据。假设 $x \in \mathbb{R}^D, z \in \mathbb{R}^d, D \gg d$，则:

$$\begin{aligned} p(X, Z|\theta) &= \prod_i p(x_i|z_i, \theta)p(z_i|\theta) \\ &= \prod_i \mathcal{N}(x_i|\boldsymbol{V}z_i + \mu, \sigma^2 I)\mathcal{N}(z_i|0, I) \end{aligned} \tag{3.4}$$

θ作为参数，包含 $\boldsymbol{V} \in \mathbb{R}^{D\times d}, \mu, \sigma \in \mathbb{R}^D$ 我们可以利用 EM 算法求解 latent variable model 视角下的 PCA。以 Eq. (3.4) 中假设为 gaussian 分布为例，考虑到 gaussian 和 gaussian 互为共轭，因此可以获得解析解。使用 EM 而不是直接求解 PAC 的好处在于:

- EM 算法每一个迭代的的复杂度为 $O(nDd)$，而直接求解 PCA 的复杂度为 $O(nD^2)$；因此当 $D \gg d$ 时 EM 算法更加高效
- 可以处理缺失的 x_i 或者多观察的 z_i
- 可以通过确定 $p(\theta)$ 自动确定 d 的值，而不需要想 PCA 一样提前确定
- 可扩展至混合 PCA

现在考虑混合 PCA(Mixture PCA)，即降维后的数据存在于多个子空间中，假设 $x \in \mathbb{R}^D, t \in \{1, ..., K\}, z \in \mathbb{R}^d$，其中 t 是每个子空间的索引，则有:

$$\begin{aligned} p(X, Z, T|\theta) &= \prod_i p(x_i|z_i, t_i, \theta)p(z_i|\theta)p(t_i|\theta) \\ &= \prod_i \mathcal{N}\left(x_i|\boldsymbol{V}_{t_i}z_i + \mu_{t_i}, \sigma^2 I\right)\mathcal{N}(z_i|0, I)\pi_{t_i} \end{aligned} \tag{3.5}$$

其中参数 θ 包含 $\boldsymbol{V}_k \in \mathbb{R}^{D\times d}, \mu_k, \sigma_k \in \mathbb{R}^D, \pi_k \in \mathbb{R}^K$，并有 $p(t_i = k) = \pi_k$

E-step:

$$
\begin{aligned}
q(Z,T) = p(Z,T|X,\theta) &= \prod_i p(z_i,t_i|x_i,\theta) \\
&= \prod_i \frac{p(x_i|z_i,t_i,\theta)p(z_i|\theta)p(t_i|\theta)}{\sum_i \int p(x_i|z_i,t_i,\theta)p(z_i|\theta)p(t_i|\theta)dz_i}
\end{aligned} \tag{3.6}
$$

M-step:

$$
\begin{aligned}
\theta_* &= \arg\max_\theta \mathbb{E}_{Z,T} \log p(X,Z,T|\theta) \\
&= \arg\max_\theta \sum_i \mathbb{E}_{Z,T}[\log p(x_i|z_i,t_i,\theta) + \log p(z_i|\theta) + \log p(t_i|\theta)]
\end{aligned} \tag{3.7}
$$

通常来说，PCA 仅构造线性子空间，即只能捕捉数据中的线性关系。然而，在许多实际应用中，数据常常分布在非线性流形上。例如，在图像处理、自然语言处理等领域，数据往往具有复杂的模式和结构，这些模式和结构不能用平面或超平面来描述。

为了更好地处理非线性数据分布，可以使用一些其他的降维技术，例如:

- t-SNE: 一种有效的降维方法，特别适合于处理高维数据的非线性结构，能够保留局部邻域的相似性。
- UMAP: 类似于 t-SNE，但更注重全局结构的同时保持局部结构。
- 自编码器 : 基于神经网络的方法，可以学习复杂的非线性映射，例如 VAE。

3.3.2 VAE

假设 $X \in \mathbb{R}^{n\times D}, Z \in \mathbb{R}^{n\times d}$，则 latent variable model 告诉我们:

$$
\begin{aligned}
p(X,Z|\theta) &= \prod_i p(x_i|z_i,\theta)p(z_i|\theta) \\
&= \prod_{i=1}^n \left(\prod_{j=1}^D \mathcal{N}\left(x_{ij}|\mu_j(z_i), \sigma_j^2(z_i)\right) \right) \mathcal{N}(z_i|0,I)
\end{aligned} \tag{3.8}
$$

在这里，我们不需要局限 $\mathcal{N}(\cdot)$ 的 μ 为 z_j 的线性函数，而是可以使用神经网络来表示非线性的 $\mu_j(z_i)$ 和 $\sigma_j^2(z_i)$，这样我们可以学习到更复杂的非线性关系

但使用非线性的 $\mu_j(z_i)$ 和 $\sigma_j^2(z_i)$ 会导致 $p(x_i|z_i,\theta)$ 与 $p(x_i|\theta)$ 不再共轭，因此无法获得解析解。同样，在 EM 算法的 E-step 中，我们也无法获得后验的解析解。即，无法计算:

$$
q(Z) = p(Z|X,\theta) = \frac{p(X,Z|\theta)}{p(X|\theta)} \tag{3.9}
$$

从 Eq. (3.2) 可知，我们从 $KL(q\|p)$ 的定义直接推导出可以通过求解 $p(Z|X,\theta)$ 来替代求解 $q(Z)$. 而在不能求解 $p(Z|X,\theta)$ 的情况下，我们则利用 variational inference 的方法求解 $q(Z)$. 例如我们可以利用 mean field approximation 的方法，即:

$$
q(Z) = \prod_i q_i(z_i) \tag{3.10}
$$

但是在神经网络的视角下，我们完全可以另外 [训练一个神经网络用于拟合 $q(Z)$]，用参数 φ 表示:

回归神经网络的本质: 拟合

$$
\begin{aligned}
&q(z_i|x_i,\varphi) \approx p(z_i|x_i,\theta) \\
&q(z_i|x_i,\varphi) = \prod_{j=1}^{d} \mathcal{N}\big(z_{ij}|\mu_j(x_i),\sigma_j^2(x_i)\big)
\end{aligned} \tag{3.11}
$$

因此:

$$
\begin{aligned}
&\text{encoder: } \varphi : x \mapsto q(z|x,\theta), \mathbb{R}^D \to \mathbb{R}^{2d} \\
&\text{decoder: } \theta : z \mapsto p(x|z,\theta), \mathbb{R}^d \to \mathbb{R}^{2D}
\end{aligned} \tag{3.12}
$$

其中 $2d$ 与 $2D$ 都是包括了 μ 和 σ^2 的两个参数

优化神经网络 φ 等价于:

$$
q(Z|X,\varphi) = \arg\min_{\varphi} KL(q(Z|X,\varphi)\|p(Z|X,\theta)) \tag{3.13}
$$

根据 Eq. (2.4)，Eq. (3.13) 等价于最大化 ELBO:

$$
\begin{aligned}
q(Z|X,\varphi) &= \arg\max_{\varphi,\theta} \mathcal{L}(\varphi,\theta) \\
&= \arg\max_{\varphi,\theta} \int q(Z|X,\varphi) \log \frac{p(X,Z|\theta)}{q(Z|X,\varphi)} dZ
\end{aligned} \tag{3.14}
$$

鉴于在非共轭情况下我们无法使用 EM 对 $q(Z|X,\varphi)$ 进行求解，我们因此使用 stochastic gradient 的方法

3.3.3 Stochastic gradient

Stochastic gradient 通常使用 mini-batch 与 Monte-Carlo estimation 来优化 ELBO，对于

$$
\begin{aligned}
\mathcal{L}(\varphi,\theta) &= \int q(Z|X,\varphi) \log \frac{p(X,Z|\theta)}{q(Z|X,\varphi)} dZ \\
&= \int q(Z|X,\varphi) \log \frac{p(X|Z,\theta)P(Z)}{q(Z|X,\varphi)} dZ
\end{aligned} \tag{3.15}
$$

对 θ 求导有:

$$
\begin{aligned}
\nabla_\theta \mathcal{L}(\varphi,\theta) &= \nabla_\theta \int q(Z|X,\varphi) \log \frac{p(X|Z,\theta)P(Z)}{q(Z|X,\varphi)} dZ \\
&= \sum_{i=1}^{n} \int q(z_i|x_i,\varphi) \nabla_\theta \log \frac{p(x_i|z_i,\theta)P(z_i)}{q(z_i|x_i,\varphi)} dz_i \\
&= \sum_{i=1}^{n} \int q(z_i|x_i,\varphi) \nabla_\theta \log p(x_i|z_i,\theta) dz_i \\
&\approx n \int q(z_i|x_i,\varphi) \nabla_\theta \log p(x_i|z_i,\theta) dz_i, i \sim \mathcal{U}\{1,...,n\} \quad \text{(mini-batch)} \\
&\approx \frac{n}{|\mathcal{U}|} \sum_{i\in\mathcal{U}} \nabla_\theta \log p(x_i|z_i^*,\theta), z_i^* \sim q(z_i|x_i,\varphi) \quad \text{(Monte-Carlo estimation)} \\
&= n \nabla_\theta \log p(x_i|z_i^*,\theta), \quad i \sim \mathcal{U}\{1,...,n\}, |\mathcal{U}| = 1, z_i^* \sim q(z_i|x_i,\varphi)
\end{aligned} \tag{3.16}
$$

其中 $\mathcal{U}$ 为数据集的子集，其大小为 $|\mathcal{U}|$，包含随机选择的$(\{x\}_i, \{z\}_i)^{|\mathcal{U}|}$；$z_i^*$ 为从 mini-batch 中的 $q(z_i|x_i,\varphi)$ 采样的样本。通常来说，先对数据集进行 mini-

batch 选择，然后在选定的 mini-batch 中进行 Monte-Carlo 采样，这样可以提高计算效率

对 φ 求导有:

$$\begin{aligned}\nabla_\varphi \mathcal{L}(\varphi,\theta) = \nabla_\varphi \Big[&\int q(Z|X,\varphi)\log p(X|Z,\theta)dZ \\ &- \int q(Z|X,\varphi)\frac{\log q(Z|X,\varphi)}{P(Z)}dZ\Big]\end{aligned} \tag{3.17}$$

第一项:

$$\begin{aligned}\nabla_\varphi \int q(Z|X,\varphi)\log p(X,Z|\theta) &= \int \log p(X|Z,\theta)\nabla_\varphi q(Z|X,\varphi)dZ \\ &= \int q(Z|X,\varphi)\log p(X|Z,\theta)\nabla_\varphi \log q(Z|X,\varphi)dZ \quad \text{(log-derivative trick)} \\ &= \sum_{i=1}^{n}\int q(z_i|x_i,\varphi)\log p(x_i|z_i,\theta)\nabla_\varphi \log q(z_i|x_i,\varphi)dz_i \\ &= n\int q(z_i|x_i,\varphi)\log p(x_i|z_i,\theta)\nabla_\varphi \log q(z_i|x_i,\varphi)dz_i, i\sim\mathcal{U}\{1,...,n\} \quad \text{(mini-batch)} \\ &= \frac{n}{|\mathcal{U}|}\sum_{i\in\mathcal{U}}\log p(x_i|z_i^*,\theta)\nabla_\varphi \log q(z_i|x_i,\varphi), z_i^*\sim q(z_i^*|x_i,\varphi) \quad \text{(Monte-Carlo estimation)} \\ &= n\log p(x_i|z_i^*,\theta)\nabla_\varphi \log q(z_i^*|x_i,\varphi), \quad i\sim\mathcal{U}\{1,...,n\}, |\mathcal{U}|=1, z_i^*\sim q(z_i|x_i,\varphi)\end{aligned} \tag{3.18}$$

注意到 $\nabla_\varphi \log q(z_i^*|x_i,\varphi)$ 为分数函数 (score function)，具有以下性质:

- $$\begin{aligned}\mathbb{E}[\nabla_\theta \log p(x|\theta)] &= \int p(x|\theta)\nabla_\theta \log p(x|\theta)dx \\ &= \int p(x|\theta)\frac{1}{p(x|\theta)}\nabla_\theta p(x|\theta)dx \\ &= \nabla_\theta \int p(x|\theta)dx = 0\end{aligned} \tag{3.19}$$

  考虑到 z_i^* 为我们抽样与 $q(z_i|x_i,\varphi)$ 的样本，因此可以认为 $\nabla_\varphi q(z_i^*|x_i,\varphi)$ 在 0 附近震荡，且其质量严格与抽样情况相关。除非 n 具有较大值，否则几乎为 0，因此可以认为梯度下降时速度较慢

- $$\begin{aligned}\mathrm{Var}(\nabla_\theta \log p(x|\theta)) &= \mathbb{E}[\nabla_\theta \log p(x|\theta)^2] \\ &= I(\theta)\end{aligned} \tag{3.20}$$

  可知 score function 的方差为 fisher information. Fisher information 反应了估计参数 θ 的方差下界，即 Cramer-Rao Lower Bound，它越大，代表估计的 θ 约精确。因此可以认为在梯度下降过程中，随着参数 θ 的逼近精确值，方差逐渐增加，导致收敛速度变慢

综上所述 $\nabla_\varphi \log q(z_i^*|x_i,\varphi)$ 的存在会导致梯度下降效率降低，因此一般不这么做，而使用 reparameterization trick 来避免这个问题。

3.3.4 Reparameterization trick

考虑复杂期望的求导:

$$\frac{\partial}{\partial_x}\int p(y|x)h(x,y)dy \tag{3.21}$$

假设 y 可以被表达为一个随机变量 ε 与 x 的函数，即 $y=g(x,\varepsilon)$，利用 Monte-Carlo estimation，我们可以将上式改写为:

$$\begin{aligned}\int p(y|x)h(x,y)dy &= \int r(\varepsilon)h(x,g(x,\varepsilon))d\varepsilon \\ &\approx \frac{d}{dx}h(x,g(x,\varepsilon^*)),\quad \varepsilon^*\sim r(\varepsilon) \\ &= \frac{\partial}{\partial x}h(x,g(x,\varepsilon^*))+\frac{\partial}{\partial y}h(x,g(x,\varepsilon^*))\frac{\partial}{\partial x}g(x,\varepsilon^*)\end{aligned} \tag{3.22}$$

常见的 reparameterization trick 有:

$p(y\|x)$	$r(\varepsilon)$	$g(\varepsilon,x)$
$\mathcal{N}(y\|\mu,\sigma^2)$	$\mathcal{N}(\varepsilon\|0,1)$	$x=\mu+\sigma\varepsilon$
$\mathcal{G}(y\|1,\beta)$	$\mathcal{G}(\varepsilon\|1,1)$	$x=\beta\varepsilon$
$\varepsilon(y\|\lambda)$	$\mathcal{U}(\varepsilon\|0,1)$	$x=-\frac{\log(\varepsilon)}{\lambda}$
$\mathcal{N}(y\|\mu,\Sigma)$	$\mathcal{N}(\varepsilon\|0,I)$	$x=A\varepsilon+\mu$, where $AA^\top=\Sigma$

reparameterization trick 并不适用于所有连续分布，且不适用于离散分布。对于离散分布，我们可以使用 Gumbel-Softmax trick 来进行 reparameterization

对于 ELBO Eq. (3.17) 第一项，我们首先进行 mini-batch，然后对其使用 reparameterization trick，最后使用 Monte-Carlo estimation:

$$\begin{aligned}\nabla_\varphi\int q(Z|X,\varphi)\log p(X|Z,\theta) &\approx n\nabla_\varphi\int q(z_i|x_i,\varphi)\log p(x_i|z_i,\theta)dz_i \\ &= n\nabla_\varphi\int r(\varepsilon)\log p(x_i|g(\varepsilon,x_i,\varphi)z_i,\theta)d\varepsilon \\ &\approx n\nabla_\varphi\log p(x_i|g(\varepsilon^*,x_i,\varphi)z_i,\theta) \\ &i\sim\mathcal{U}\{1,...,n\},|\mathcal{U}|=1,z_i=g(\varepsilon,x_i,\varphi),\varepsilon^*\sim r(\varepsilon)\end{aligned} \tag{3.23}$$

3.3.5 VAE Algorithm

我们的目标函数则为:

$$\mathcal{L}(\varphi,\theta)=\mathbb{E}_{q(Z|X,\varphi)}\log p(X|Z,\theta)-KL(q(Z|X,\varphi)\|p(Z)) \tag{3.24}$$

其中，$q(Z|X,\varphi)$ 为 encoder 网络，$p(X|Z,\theta)$ 为 decoder

接着更新 φ,θ，迭代直至收敛。因为两个都是无偏估计且存在 ELBO 的下界保证，因此可以保证收敛

对于 $X\in\mathbb{R}^{n\times d}$，随机选取 mini-batch $\mathcal{U}=(\{x\}_i,\{z\}_i)^{|\mathcal{U}|}$，计算:

$$\text{stoch.grad}_\theta \mathcal{L}(\varphi, \theta) = \frac{n}{|\mathcal{U}|} \sum_{i \in \mathcal{U}} \nabla_\theta \log p(x_i | z_i^*, \theta) \tag{3.25}$$

其中 $z_i^* \sim q(z_i | x_i, \varphi)$

$$\begin{aligned} \text{stoch.grad}_\varphi \mathcal{L}(\varphi, \theta) = \frac{n}{|\mathcal{U}|} \sum_{i \in \mathcal{U}} & \nabla_\varphi \log p(x_i | g(\varepsilon^*, x_i, \varphi), \theta) \\ & - \nabla_\varphi KL(q(z_i | x_i, \varphi) \| p(z_i)) \end{aligned} \tag{3.26}$$

其中 $z_i = g(\varepsilon, x_i, \varphi), \varepsilon^* \sim r(\varepsilon)$

3.3.6 VAE Code

对于目标函数:

$$\mathcal{L}(\varphi, \theta) = \mathbb{E}_{q(Z|X,\varphi)} \log p(X|Z, \theta) - KL(q(Z|X, \varphi) \| p(Z)) \tag{3.27}$$

对于第一项期望, 可以使用 Monte-Carlo estimation, 这样只剩下 $\log p(X|Z, \theta)$, 根据数据类型一般考虑 $p(X|Z, \theta)$ 为 Bernoulli 或者 Gaussian 分布, 在 Bernoulli 分布下, 我们可以使用交叉熵损失函数:

$$\text{loss} = -\sum_i \sum_j \left[x_{ij} \log y_{ij} + \left(1 - x_{ij}\right) \log\left(1 - y_{ij}\right) \right] \tag{3.28}$$

考虑到 x_{ij} 与 y_{ij} 范围取值为 $[0, 1]$, 因此通常来说都可以使用交叉熵损失函数

对于 KL-divergence, 考虑到 $q(Z|X, \varphi)$ 为高斯分布, $p(Z)$ 为标准高斯分布, 因此 KL-divergence 可以直接计算:

$$KL(q(Z|X, \varphi) \| p(Z)) = \frac{1}{2} \sum_{i=1}^{d} [1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2] \tag{3.29}$$

proof:

对于两高斯分布的 KL-divergence 的:

$$\begin{aligned} &\text{KL}\ (\mathcal{N}(\mu, \Sigma) \| \mathcal{N}(\mu', \Sigma')) = \\ &\qquad \frac{1}{2} \left[\text{tr}(\Sigma'^{-1}\Sigma) + (\mu' - \mu)^\top \Sigma'^{-1} (\mu' - \mu) - k + \log\left(\det(\Sigma'^{-1}\Sigma)\right) \right] \end{aligned} \tag{3.30}$$

其中 Σ 为协方差矩阵, k 为维度

特殊的, 当一个高斯分布为标准高斯分布时, 其 KL-divergence 为:

$$\text{KL}\ (\mathcal{N}(\mu, \Sigma) \| \mathcal{N}(0, I)) = \frac{1}{2} \left[\text{tr}(\Sigma) + \mu^\top \mu - k - \log(\det(\Sigma)) \right] \tag{3.31}$$

更特殊的, 对于对角协方差矩阵 $\Sigma = \text{diag}(\sigma_1^2, \sigma_2^2, ..., \sigma_k^2)$ 有:

$$\text{tr}(\Sigma) = \sum_{i=1}^{k} \sigma_i^2, \quad \det(\Sigma) = \prod_{i=1}^{k} \sigma_i^2 \tag{3.32}$$

因此:

$$\mathrm{KL}\left(\mathcal{N}(\mu,\Sigma)\|\mathcal{N}(0,I)\right)=\frac{1}{2}\left[\sum_{i=1}^{k}(1+\log(\sigma_i^2)-\mu_i^2-\sigma_i^2)\right] \quad (3.33)$$

因此，目标函数可以改写为:

$$\mathcal{L}(\varphi,\theta)\simeq\frac{1}{2}\sum_{i=1}^{d}[1+\log(\sigma_i^2)-\mu_i^2-\sigma_i^2]+\frac{1}{L}\sum_{l=1}^{L}\log p(x|z_l,\theta) \quad (3.34)$$

其中 $z_l=\mu+\sigma\odot\varepsilon,\varepsilon\sim\mathcal{N}(0,I)$

```python
def loss_function(recon_x, x, mu, logvar):
    """
    计算 VAE 的损失函数，包括重构损失和 KL 散度
    """
    # 重构损失
    BCE = F.binary_cross_entropy(recon_x, x, reduction='sum')
    # KL 散度计算
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + KLD
```

对于 encoder $P(Z|X,\varphi)$，要计算 μ 和 $\log\sigma^2$，这样我们可以根据 μ 和 $\log\sigma^2$ 采样 z，这样我们可以保证 z 为高斯分布

```python
def encode(self, x):
    """
    编码器前向传播，输出潜在变量的均值和对数方差
    """
    h1 = F.relu(self.fc1(x))
    mu = self.fc_mu(h1)
    logvar = self.fc_logvar(h1)
    return mu, logvar
def reparameterize(self, mu, logvar):
    """
    重参数化技巧，从 Q(Z|X) 中采样潜在变量 Z
    """
    std = torch.exp(0.5 * logvar)  # 标准差 σ
    eps = torch.randn_like(std)    # 从标准正态分布中采样 ε
    return mu + eps * std          # 潜在变量 Z
```

对于 decoder $P(X|Z,\theta)$，我们可以直接计算重构的 x:

```python
def decode(self, z):
    """
    解码器前向传播，重构输入
    """
    h3 = F.relu(self.fc3(z))
    return torch.sigmoid(self.fc4(h3))  # 输出范围 [0,1]
```

3.4 Discrete Latent Variables

3.4.1 Reinforce estimator

在连续潜变量模型中，我们利用 reparameterization trick 对 Eq. (3.17) 的第一项进行了求导。而对于离散潜变量模型，我们无法使用 reparameterization trick，因此我们需要使用 Reinforce estimator. 考虑:

$$\mathcal{L}(\varphi) = \sum_Z q(Z;\varphi) f(Z) = \mathbb{E}_{q(Z;\varphi)} f(Z) \tag{3.1}$$

对其求导:

$$\begin{aligned} \nabla_\varphi \mathcal{L}(\varphi) &= \sum_Z \nabla_\varphi q(Z;\varphi) f(Z) \\ &= \sum_Z q(Z;\varphi) f(Z) \nabla_\varphi \log q(Z;\varphi), \quad \text{(log-derivative trick)} \\ &= \frac{1}{M} \sum_{m=1}^M f(z_m) \nabla_\varphi \log q(z_m;\varphi), \quad \text{(Monte-Carlo estimation)} \end{aligned} \tag{3.2}$$

因此 reinforce estimator 为:

$$g(z_{1:M}, \varphi) = \frac{1}{M} \sum_{m=1}^M f(z_m) \nabla_\varphi \log q(z_m;\varphi), \quad z_m \sim q(z_m;\varphi) \tag{3.3}$$

注意到，reinforce estimator 不仅允许离散潜变量，甚至允许不可微函数 $f(\cdot)$ 的存在。但同时存在以下缺点:

1. 方差较大，通过增大 M 只会使 std 以 $1/\sqrt{M}$ 的速率减小
2. 其梯度方向由向量 $\nabla_\varphi \log q(z_m;\varphi)$ 决定，步长由标量 $f(z_m)$ 决定，因此可以认为梯度方向指向 Z 概率增加的方向。
3. 不像 reparameterization trick, reinforce estimator 缺少 $\nabla_\varphi f(Z)$ 的信息（例如 VAE 中 decoder 的梯度信息），而只使用了值；因此还对函数 $f(Z)$ 的移动 (e.g. $f(x)+c$) 敏感

3.4.2 Gumbel-SoftMax trick

考虑到 Eq. (3.3) 这么多的缺点，一个合理的想法是将离散潜变量转化为连续潜变量，然后使用 reparameterization trick，即:

$$\mathbb{E}_{q(Z;\varphi)} f(Z) = \mathbb{E}_{q(\tilde{Z}|\varphi)} f\big(\tilde{Z}\big) = \mathbb{E}_{p(\gamma)} f\big(\tilde{Z}(\gamma,\varphi)\big) \tag{3.4}$$

其中 γ 为噪声，$\tilde{Z}(\gamma,\varphi)$ 为 Z 的连续估计，此时要确保 $f(\cdot)$ 连续可微

一种方式是使用 Gumbel-Max trick，假设 z 为 K 类离散变量，各自概率为 $\{\pi_i\}_{i=1}^K, \sum_i \pi_i = 1$，则

$$\begin{aligned} z &= \arg\min_i \frac{\zeta_i}{\pi_i}, \quad \zeta_i \sim \mathrm{Exp}(1) \\ &= \arg\max_i [\log \pi_i - \log \zeta_i] \\ &= \arg\max_i [\log \pi_i + \gamma_i], \quad \gamma_i \sim \mathrm{Gumbel}(0,1) \end{aligned} \tag{3.5}$$

唯一的问题在于 $\arg\max(\cdot)$ 不可导

proof:

对于:

$$Y_i = \frac{\zeta_i}{\pi_i} = g(\zeta_i), \quad \zeta_i \sim \text{Exp}(1) \tag{3.6}$$

计算其 pdf:

$$\begin{aligned} f_{Y_i}(y) &= f_{\zeta_i}(g^{-1}(\zeta_i)) \left| \frac{d}{dy} g^{-1}(\zeta_i) \right| \\ &= f_{\zeta_i}(\pi_i y)\pi_i \\ &= \pi_i e^{-\pi_i y} \end{aligned} \tag{3.7}$$

因此:

$$Y_i \sim \text{Exp}(\pi_i) \tag{3.8}$$

且:

$$\begin{aligned} P\Big(Y_i = \min\left\{Y_j\right\}_{j=1}^{K}\Big) &= \int_0^{+\infty} P(Y_i = y) \prod_{i \neq j} P(Y_j \geq y) dy \\ &= \int_0^{+\infty} \pi_i e^{-\pi_i y} \prod_{i \neq j} e^{-\pi_j y} dy \\ &= \int_0^{+\infty} \pi_i \exp\left(-\pi_i y + \sum_{i \neq j} -\pi_j y \right) dy \\ &= \int_0^{+\infty} \pi_i \exp(-y) dy \\ &= \pi_i \end{aligned} \tag{3.9}$$

因此使用 Gumbel-SoftMax trick，**即使用带温度控制的** SoftMax **替代** argmax:

$$\text{softmax}(x; \tau)_j = \frac{\exp\big(x_j/\tau\big)}{\sum_i \exp(x_i/\tau)} \tag{3.10}$$

其中温度 τ **控制与** argmax **的相似性:**

- **当** $\tau = 0$ **时，** $\text{softmax} = \text{argmax}$
- **当** $\tau = \infty$ **时，** $\text{softmax} = \text{Uniform}$

因此:

$$\tilde{z}(\gamma, \pi) = \text{softmax}(\log \pi_i + \gamma_i; \tau), \quad i = 1, ..., K \tag{3.11}$$

其中 $\gamma_i \sim \text{Gumbel}(0, 1)$，**等价于**

$$\gamma_i = -\log(\log u_i), \quad u_i \sim \text{Uniform}(0, 1) \tag{3.12}$$

此时， Eq. (3.1) **可改写为:**

$$\mathcal{L}(\varphi) = \mathbb{E}_{p(\gamma)} f\big(\tilde{Z}(\gamma, \varphi)\big) \tag{3.13}$$

此时

$$\begin{aligned}\nabla_\varphi \mathcal{L}(\varphi) &= \nabla_\varphi \mathbb{E}_{p(\gamma)} f\big(\tilde{Z}(\gamma,\varphi)\big) \\ &= \nabla_\varphi f\big(\tilde{Z}(\gamma^*,\varphi)\big), \quad \gamma^* \sim \text{Gumbel}(0,1)\end{aligned} \tag{3.14}$$

引入噪声 γ 的好处有:

- 提升泛化能力
- 正确的噪声类型（例如 $\text{Gumbel}(0,1)$）可以使 $\tilde{z}$ 类似于 one-hot vector，提升训练集与测试集的相似度

对于温度 τ，通常使 $\tau \leq 1/(K-1)$，并且使用 grid search 搜索。小 τ 会导致高方差，但更能表示离散值；大 τ 反之。

3.4.3 Control Variates

对于 Eq. (3.3)，另一个合理的想法是控制 reinforce estimator 的方差，利用 variant control，通过减去已知量的估计误差来降低未知量的估计误差. 假设对于未知随机变量 $V(X)$，以及已知随机变量 $W(X)$ 与 其计算出的期望 $\mu = \mathbb{E}[W(X)]$，我们可以构造一个新的随机变量，与 $V(X)$ 具有相同期望，但方差更小:

$$Z = V(X) - \alpha(W(X) - \mu) \tag{3.15}$$

此时方差为:

$$\text{Var}(Z) = \text{Var}(V(X)) - 2\alpha\ \text{Cov}(V(X), W(X)) + \alpha^2\ \text{Var}(W(X)) \tag{3.16}$$

注意到上式为关于 α 的二次方程，最小值为:

$$\begin{aligned}\text{Var}_{\min}(Z) &= \text{Var}(V(X)) - \frac{\text{Cov}^2(V(X), W(X))}{\text{Var}^2(W(X))} \\ \alpha^* &= \frac{\text{Cov}(V(X), W(X))}{\text{Var}(W(X))}\end{aligned} \tag{3.17}$$

除此以外，我们还可以利用多个已知变量 $W_i(X)$ 来降低方差（查看 [3]）

因此，假设 $\mu = \mathbb{E}_{q(Z;\varphi)}[b(Z)]$，则 Eq. (3.3) 可以改写为:

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_{q(Z;\varphi)}[f(Z) - b(Z) + \mu] \\ &= \mathbb{E}_{q(Z;\varphi)}[f(Z) - b(Z)] + \mu(\varphi)\end{aligned} \tag{3.18}$$

利用 Monte-Carlo estimation，我们可以得到:

$$\mathcal{L} = \mathbb{E}_{q(z_{1:M}|\varphi)}\left[\frac{1}{M}\sum_{m=1}^{M} f(z_i) - b(z_i)\right] + \mu(\varphi) \tag{3.19}$$

求导得到 reinforce estimator:

$$g(z_{1:M}, \varphi) = \frac{1}{M}\sum_{m=1}^{M}[f(z_m) - b(z_m)]\nabla_\varphi \log q(z_m;\varphi) + \nabla_\varphi \mu(\varphi) \tag{3.20}$$

其中 $z_i \sim q(Z;\varphi)$，$b(Z)$ 被称为 baseline，$b(Z)\nabla_\varphi \log q(Z;\varphi)$ 被称为 control variate. 接下来讨论 baseline 的选择:

1. 选择 baseline 为常数 $b(Z)=c$，有 $\nabla_\varphi \mu(\varphi)=0$，因此:

$$\begin{aligned}\mathrm{Var}(g) &= \mathrm{Var}\left(\frac{1}{M}\sum_{m=1}^{M}[f(z_m)-b(z_m)]\nabla_\varphi \log q(z_m;\varphi)\right)\\ &= \frac{1}{M^2}\sum_{m=1}^{M}\mathrm{Var}\Big([f(z_m)-c]\nabla_\varphi \log q(z_m;\varphi)\Big), \quad \text{i.i.d. } z_m \qquad (3.21)\\ &= \frac{1}{M^2}\sum_{m=1}^{M}\Big[\mathrm{Var}\Big(f\cdot\nabla_\varphi\Big)+c^2\ \mathrm{Var}\Big(\nabla_\varphi\Big)-2c\ \mathrm{Cov}\ \Big(f\cdot\nabla_\varphi,\nabla_\varphi\Big)\Big]\end{aligned}$$

   因此，c 的最佳选择为:

$$c^*=\frac{\mathrm{Cov}\Big(f(Z)\nabla_\varphi \log q(Z;\varphi),\nabla_\varphi \log q(Z;\varphi)\Big)}{\mathrm{Var}\Big(\nabla_\varphi \log q(Z;\varphi)\Big)} \qquad (3.22)$$

   但如果有额外的观察项 (例如 VAE 中的 $\log q(Z|X,\varphi)$)，则最佳 baseline 的选择应该与 x 有关，因此不再适用于上式
2. NVIL: 针对上面的问题，[4] 提出了使用 MSE 来估计 baseline:

$$b(X)=\arg\min_b \mathbb{E}_{p(X)}\mathbb{E}_{q(Z|X,\varphi)}[f(Z)-b(X)]^2 \qquad (3.23)$$

3. MuProp: [5] 提出了将 $f(z)$ 的一阶泰勒展开作为 baseline:

$$b(Z)=f(\mu)+\nabla_Z f(\mu)^\top\cdot(Z-\mu) \qquad (3.24)$$

   对于 μ 取 $\mu=\mu(\varphi)=\mathbb{E}_{q(Z;\varphi)}Z$，有

$$g(Z,\varphi)=(f(Z)-b(Z))\nabla_\varphi \log q(Z;\varphi)+\nabla_\varphi f(\mu(\varphi)) \qquad (3.25)$$

除此以外，还有其他不同的方法，如 [6], [7], etc.

3.5 GAN

3.5.1 GAN

GAN 的思想在于，使神经网络生成的分布 $q(x)$ 与真实分布 $p(x)$ 尽可能接近。接近程度由判别器决定:

$$f(x;\varphi) = p(y=1|x,\varphi) \tag{3.1}$$

或者计算两个分布之间的举例:

$$D_f(p\|q) \tag{3.2}$$

通过判别器的指示，我们让 $q(x)$ 逐渐逼近 $p(x)$. 在这里我们需要对 $q(x)$ 这个分布进行采样，但$q(x)$ 的具体分布不知道，因此我们可以训练一个生成器:

$$\hat{x} = G(z;\theta), \quad z \sim p(z) \tag{3.3}$$

使得:

1. z 容易从 $p(z)$ 中采样，例如 $\mathcal{N}(0,I)$,
2. 采样的结果可以映射到 $q(x)$ 中，即 $\hat{x} \sim q(x)$

如果使用 Eq. (3.1), 考虑到这是一个二分类问题，且判别器需要尽可能分开 $p(x)$ 与 $q(x)$，因此其 loss 为:

$$\mathcal{L}(\varphi,\theta) = \mathbb{E}_{p(x)}[\log f(x;\varphi)] + \mathbb{E}_{p(z)}[\log(1-f(G(z;\theta);\varphi))] \tag{3.4}$$

上式等价于使用 Eq. (3.2), 其 loss 为:

$$\mathcal{L}(\varphi,\theta) = D_f(p(x)\|q(x;\theta)) \tag{3.5}$$

我们只需要

1. 更新判别器:

$$\varphi = \arg\max_{\varphi} \mathcal{L}(\varphi,\theta) \tag{3.6}$$

2. 更新生成器（最小化 p,q 距离）:

$$\theta^{t+1} = \theta^t + \nabla_\theta \mathcal{L}(\varphi,\theta^t) \tag{3.7}$$

3. 重复 1,2 直至收敛

其中，D_f 为 f-divergence，可以是任意的 divergence measure

$$D_f(P\|Q) = \int_{\mathcal{X}} f\left(\frac{p(x)}{q(x)}\right) q(x)dx \tag{3.8}$$

其中 f 指明了 divergence 的形式:

$$f(t) = \begin{cases} t\log t, & \text{KL-divergence} \\ -\log t, & \text{Reverse KL-divergence} \\ \frac{1}{2}|t-1|, & \text{Total variation} \end{cases} \tag{3.9}$$

3.5.2 Optimal transport

根据 Eq. (3.4) 和 Eq. (3.5) 我们知道：最小化 generator 的 loss 等价于最小化 generator 生成的分布与 target 分布的 JS divergence。

但是用 JS divergence 来作为度量有个致命缺陷，就是在两个分布互不相交的情况下，两个分布的 JS divergence 永远都是常数 $\log 2$，并且由于 generator 生成的分布和target分布的支撑集是在嵌在高维空间中的低维流形, 所以他们重叠的部分的测度几乎为0。这样完全无法进行度量两个分布在不相交情况下的距离。计算梯度的时候也会出现 0 梯度的情况。[8]

因此我们需要采用新的度量方式，这就是 optimal transport. Optimal transport 是一个线性规划问题, 它的目标是找到两个分布之间的最小运输成本。假设从 x 运输到 y 具有一定的成本 c，一般来说，定义为:

$$c(x, y) = \|x - y\|_k^k \tag{3.10}$$

原始定义为:

$$L = \arg\min_{\Gamma} \sum_{i,j}^{M,N} \Gamma_{i,j} c\big(x_i, y_j\big) \tag{3.11}$$

optimal transport 具有概率版本，其 optimal transport divergence 为

$$T(P, Q) = \inf_{\Gamma \in P(x \sim P, y \sim Q)} \mathbb{E}_{(x,y)\sim\Gamma} c(x, y) \tag{3.12}$$

其中，Γ 为 P 到 Q 的一个联合分布，表示从 P 到 Q 的运输方案，满足联合分布的性质。转换为对偶问题 (dual problem) 为:

$$T(P, Q) = \sup_{\varphi, \psi \in L_1} \Big(\mathbb{E}_{p(x)} \varphi(x) + \mathbb{E}_{q(y)} \psi(y) \Big) \tag{3.13}$$

其中 $L_1 : \{\varphi(x) + \psi(x) \le c(x, y)\}$

因此，需要保证神经网络的函数是光滑的 Lipschitz 函数，即:

$$\|f(x) - f(y)\|_K \le L \, \|x - y\|_K \tag{3.14}$$

对于 FFN，由仿射变换和逐点非线性组成的函数，这些非线性是光滑的 Lipschitz 函数（例如 sigmoid, tanh, elu, softplus 等）[9]. 对于线性矩阵计算函数，通过以下方式判断:

$$\begin{aligned} \|Ax_1 - Ax_2\|_2 &\le L \|x_1 - x_2\|_2 \\ \sigma_{\max} = \sup_x \frac{\|Ax\|_2}{\|x\|_2} &\le L \end{aligned} \tag{3.15}$$

其中 $\sup_x \frac{\|Ax\|_2}{\|x\|_2}$ 刚好等价于矩阵的 spectral norm, 即矩阵的最大奇异值。因此，为了使矩阵 A 为满足 Lipschitz 函数，只需要使

$$A := \frac{A}{\sigma_{\max}} \tag{3.16}$$

可以使用 power iteration 来计算 $\sigma_{\max}$ (power iteration wiki)

3.5.3 Gan Algorithm

$$\begin{aligned}&\min_{\theta}\min_{\varphi}\mathcal{L}(\theta,\varphi)=\\&\min_{\theta}\min_{\varphi}\mathbb{E}_{p(x)}\log D(x;\varphi)+\mathbb{E}_{p(z)}\log(1-D(G(z;\theta);\varphi))\end{aligned}\tag{3.17}$$

1. $$\varphi^{t+1}=\varphi^{t}+\alpha\nabla_{\varphi}\mathcal{L}(\theta^{t},\varphi^{t})\tag{3.18}$$
2. $$\theta^{t+1}=\theta^{t}-\alpha\nabla_{\theta}\mathcal{L}(\theta^{t},\varphi^{t+1})\tag{3.19}$$
3. 重复 1,2 直至收敛

3.6 Normalizing Flows

来源与论文[10]，与 VAE 类似，normalizing flows 假设潜变量 z，并可以根据数据 x 得到潜变量，即:

$$z = f_\theta(x) \tag{3.1}$$

与 VAE 不同的是，normalizing flows 不使用 decoder 从 z 获得 x，而是期望找到 $f_\theta^{-1}(\cdot)$ 使得:

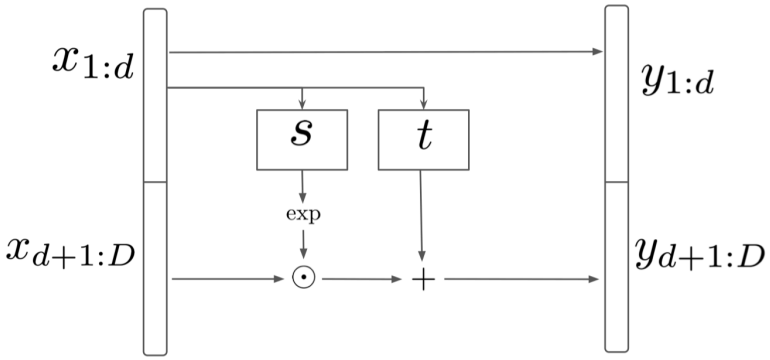
$$x = f_\theta^{-1}(z) \tag{3.2}$$

根据变量转换公式，我们有:

$$\begin{aligned} p(x) &= p(z)\left|\det \frac{df_\theta^{-1}(z)}{dz}\right| \\ &= p(f(x))\left|\det \frac{df_\theta(x)}{dx}\right| \end{aligned} \tag{3.3}$$

此时需要保证 z 和 x 的维度相同。同时，还需要保证 $f_\theta(\cdot)$ 是可逆的，因此设计如下灵活且可解决的双射函数作为 coupling layer. 假设 $x \in \mathbb{R}^D$ 且 $d < D$，有:

$$\begin{aligned} y_{1:d} &= x_{1:d} \\ y_{d+1:D} &= x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d}) \end{aligned} \tag{3.4}$$



可以很容易得到其逆变换:

$$\begin{aligned} x_{1:d} &= y_{1:d} \\ x_{d+1:D} &= (y_{d+1:D} - t(y_{1:d})) \odot \exp(-s(y_{1:d})) \end{aligned} \tag{3.5}$$

其中 $s, t : \mathbb{R}^d \mapsto \mathbb{R}^{D-d}$. 这样，Jacobian 矩阵为:

$$\frac{\partial y}{\partial x^\top} = \begin{bmatrix} I_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x_{1:d}^\top} & \operatorname{diag}(\exp[s(x_{1:d})]) \end{bmatrix} \tag{3.6}$$

代入 Eq. (3.3)，我们有:

$$\left|\det \frac{df_\theta(x)}{dx}\right| = \exp\left(\sum_j s(x_{1:d})_j\right) \tag{3.7}$$

注意到，在 Eq. (3.4) 中，$y_{1:d}=x_{1:d}$ 并没有经过变换，我们可以结合多个不同的 coupling layer 来解决这个问题，对于在一个 coupling layer 上未经变换的部分，我们让其在下一个 coupling layer 进行变换。即:

$$f_\theta(x)=f^N\circ\cdots\circ f^1(x) \tag{3.8}$$

因此，根据 MLE:

$$\begin{aligned}\log p_\theta(x)&=\log p(f_\theta(x))+\log\left|\det\frac{\partial f_\theta(x)}{\partial x^\top}\right|\\&=\log p(f_\theta(x))+\sum_i^N\log\left|\det\frac{\partial f^i}{\partial f^{i-1}}\right|\end{aligned} \tag{3.9}$$

第四章 优化算法

4.1 Conjugate gradient algorithm

link: Conjugate gradient method-Wikipedia

from: Rasmussen, C. (2006). Conjugate gradient algorithm, version 2006-09-08. available online.

共轭梯度算法（Conjugate Gradient Algorithm）是一种用于求解大规模线性系统 $\boldsymbol{Ax} = \boldsymbol{b}$ 的迭代方法，其中 $\boldsymbol{A}$ 是一个对称正定矩阵。这个算法尤其适用于稀疏矩阵，因为它可以避免直接求解矩阵的逆，降低了计算复杂度。

共轭梯度算法的基本思想是通过迭代的方法逐步逼近线性方程的解，利用前一步的解信息来加速收敛。其步骤通常如下:

1. 初始化：选择初始点 x_0，计算残差 $\boldsymbol{r}_0 = b - \boldsymbol{A}\boldsymbol{x}_0$, 设置初始搜索方向 $\boldsymbol{p}_0 = \boldsymbol{r}_0$
2. 迭代:
   - 计算步长

$$\alpha_k = \frac{\boldsymbol{r}_k^\top \boldsymbol{r}_k}{\boldsymbol{p}_k^\top \boldsymbol{A}\boldsymbol{p}_k} \tag{4.1}$$

   - 更新解

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \tag{4.2}$$

   - 更新残差

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{p}_k \tag{4.3}$$

   - 检查收敛条件，若满足则停止迭代

$$\boldsymbol{r}_{k+1} = \boldsymbol{c} \tag{4.4}$$

   - 否则，计算新的搜索方向

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k, \beta_k = \frac{\boldsymbol{r}_{k+1}^\top \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^\top \boldsymbol{r}_k} \tag{4.5}$$

3. 重复步骤 2，直到满足收敛条件

4.2 Natural Gradient

机器学习论文

第一章 聚类

1.1 Prototypical Contrastive Learning

https://arxiv.org/abs/2005.04966

https://github.com/salesforce/PCL

无监督特征学习方法，将对比学习与聚类结合。

不仅学习用于实例区分的低级特征，更重要的是将聚类所发现的语义结构编码到学习到的嵌入空间中

第二章 Latent Variable Model

2.1 Variational RNN

2.1.1 Motivation

对于通常 RNN 来说，其训练为:

$$h_t = f_\theta(h_{t-1}, x_t) \tag{6.1}$$

而推理过程则使用训练好的神经网络 θ，这导致 RNN 网络的变异性较低，即: 输出的任何变化或波动都仅仅取决于 RNN 经过训练所学习到的 θ。

这意味着，RNN 生成的输出受学习到的模式和规律影响，而不是受隐藏状态之间的直接转移过程的影响。换句话说，内部状态改变 (即隐藏状态的变化) 并不会直接引入新的变异性，所有变化都通过输出层的概率机制来实现。

因此，作者提出: 在每一个时间步加入一个 VAE 机制，隐变量 z_t 从 $\mathcal{N}(\theta_t)$ 采样，但 θ_t 又由 h_{t-1} 决定。这样，z_t 的变化将直接影响到输出的变异性。

2.1.2 数学公式

在生成过程中:

对于每个时间步 t 的 VAE，其隐变量 z_t 采样于:

$$z_t \sim \mathcal{N}\big(\mu_{0,t}, \mathrm{diag}\big(\sigma_{0,t}^2\big)\big) \tag{6.2}$$

其中，$\mathcal{N}(\cdot)$ 的参数由 h_{t-1} 生成:

$$\mu_{0,t}, \sigma_{0,t} = \varphi_\tau^{\mathrm{prior}}(h_{t-1}) \tag{6.3}$$

x_t 的生成则为:

$$x_t | z_t \sim \mathcal{N}\big(\mu_{x,t}, \mathrm{diag}\big(\sigma_{x,t}^2\big)\big) \tag{6.4}$$

其中，$\mathcal{N}(\cdot)$ 的参数由 z_t 和 h_{t-1} 生成:

$$\mu_{x,t}, \sigma_{x,t} = \varphi_\tau^{\mathrm{dec}}(\varphi_\tau^z(z_t), h_{t-1}) \tag{6.5}$$

对于 RNN:

$$\begin{aligned} h_t &= f_\theta(\varphi_\tau^x(x_t), \varphi_\tau^z(z_t), h_{t-1}) \\ p(x_{\le T}, z_{\le T}) &= \prod_{t=1}^{T} p(x_t | z_{\le t}, x_{<t}) p(z_t | x_{<t}, z_{<t}) \end{aligned} \tag{6.6}$$

在推理过程中:

隐变量的后验采样于:

$$z_t | x_t \sim \mathcal{N}\big(\mu_{z,t}, \mathrm{diag}\big(\sigma_{z,t}^2\big)\big) \tag{6.7}$$

其中，$\mathcal{N}(\cdot)$ 的参数由 x_t 和 h_{t-1} 生成:

$$\mu_{z,t}, \sigma_{z,t} = \varphi_\tau^{\mathrm{enc}}(\varphi_\tau^x(x_t), h_{t-1}) \tag{6.8}$$

进而:

$$q(z_{\leq T}|x_{\leq T}) = \prod_{t=1}^{T} q(z_t|x_{\leq t}, z_{<t}) \tag{6.9}$$

我们的目标函数为:

$$\begin{aligned}\mathbb{E}_{q(z\leq T|x\leq T)}\Bigg[\sum_{t=1}^{T}(-\mathrm{KL}(q(z_t|x_{\leq t}, z_{<t})\|p(z_t|x_{<t}, z_{<t}))\\ +\log p(x_t|z_{\leq t}, x_{<t}))\Bigg]\end{aligned} \tag{6.10}$$

第三章 长序列算法

3.1 记忆力

一种记忆力的评估方法是评估模型，在第 n 步时可以利用多远的信息计算输出[11]. 对于输入序列 $u(t)$，输出序列 $y(t)$，统计以下输出中不为零的数量:

$$\frac{\partial y(t)}{\partial u(t-n)}, \quad n=0,1,...,t \tag{7.1}$$

以 SSM 为例，有:

$$\frac{\partial y(t)}{\partial u(t-n)} = CA^nB \tag{7.2}$$

除此以外，[12] 提出了以下方式:

1. 生成序列: 序列第一个位置为`[bos]`，后续的每一个位置随机来自于字典中的 token
2. 计算 pre-softmax logits: 在计算注意力时，会计算 query 和所有 key 的相似性(点积)，因此这里相当于计算了位置 i 和所有位置的相似性

$$\text{logits}_{i,j} = q_i^\top k_j \tag{7.3}$$

3. 对 pre-softmax logit 归一化: 在所有注意力头上平均

3.2 Hyena Hierarchy

论文: [11]

3.2.1 结合 SSM 的卷积

以普通的卷积为例，假设 $S \in \mathbb{R}^{L\times L}$ 为 filter，$U \in \mathbb{R}^{L\times C}$ 为输入，y 为输出，则有:

$$y_t = (h*u)_t = \sum_{n=0}^{L-1} h_{t-n}u_n \tag{7.1}$$

这里 $h \in \mathbb{R}^L$，为了便于 SSM 的加入，令 filter S 为 Toeplitz 矩阵，即每一个对角线上元素相同:

$$\begin{gathered} S_{i,j} = S_{i+1,j+1} \\ S = \begin{bmatrix} h_0 & h_{-1} & ... & h_{-L+1} \\ h_1 & h_0 & ... & h_{-L+2} \\ ... & ... & ... & ... \\ h_{L-1} & h_{L-2} & ... & h_0 \end{bmatrix} \end{gathered} \tag{7.2}$$

根据 SSM Eq. (1.3)，我们可以得到 filter:

$$h_t = \begin{cases} 0 & t<0 \\ CA^tB + D\delta_t & t \geq 0 \end{cases} \tag{7.3}$$

3.2.2 FFT Conv

卷积操作可以利用 FFT 优化运行速度。卷积操作的空间复杂度为$O(L^2)$，而利用 FFT 可以将其降低到$O(L\log L)$。具体运用了傅里叶变换的卷积性质 Eq. (1.15). 考虑 filter 为 Toeplitz 矩阵的特殊情况，循环矩阵:

$$S_h = \begin{bmatrix} h_0 & h_1 & \dots & h_{L-1} \\ h_{L-1} & h_0 & \dots & h_{L-2} \\ \dots & \dots & \dots & \dots \\ h_1 & h_2 & \dots & h_0 \end{bmatrix} \tag{7.4}$$

利用 FFT，我们可以将循环矩阵对角化:

$$S_h = W^{-1} D_H W \tag{7.5}$$

其中 D_H 为对角矩阵，W 为 DFT 矩阵。因此，卷积操作可以写为:

$$\begin{aligned} y &= S_h u \\ &= W^{-1} D_H W u \\ &= \text{iFFT}(D_h \ \text{FFT}(u)) \end{aligned} \tag{7.6}$$

其中，对角矩阵 D_H 的对角元素为循环矩阵的特征值，可以通过以下方式计算:

$$p(\lambda) = \det(S_h - \lambda I) = 0 \tag{7.7}$$

3.2.3 Order-N hyena operator

假设 $\left(v, x^1, ..., x^N\right)_t$ 为输入 u 的投影，同时 filters $\left(h^1, ..., h^N\right)_t$为可学习的，hyena operator 执行以下循环操作:

$$\begin{aligned} z_t^1 &= v \\ z_t^{n+1} &= x_t^n \left(h^n * z^n\right)_t \quad n = 1, ..., N, \\ y_t &= z_t^{N+1} \end{aligned} \tag{7.8}$$

这一循环操作的卷积由 FFT 完成，空间复杂度是 $O(NL\log L)$.同时注意到，每一步循环操作包括:

1. 对时域进行卷积 $(h_t^n * z_t^n)$,
2. 对频域进行卷积 (时域 element-wise product, $x_t^n \left(h^n * z^n\right)_t$).

作者认为: 时域上的卷积被认为提高了记忆的长度，而频域上的卷积被认为提高了频率的精细度。

3.2.4 self-attention operatior

通常来说，self-attention 只包括 3 个部分: query, key, value:

$$\begin{aligned} y &= \text{self-attetnion}(u) \\ &= \text{softmax}\left(\frac{1}{\sqrt{D}} u M_q M_k^\top u^\top\right) u M_v \\ &= A(q,k)v \end{aligned} \tag{7.9}$$

卷积本质可以理解为对信号的加权和操作，在时域中反映为对历史信息（过往信号）的累积，而在频域中则反映为对信号的频率成分的加权调整

其中，$M_q, M_k, M_v \in \mathbb{R}^{D\times D}$ 为输入 $u \in \mathbb{R}^{L\times D}$ 可学习的投影。

在 hyena 的 attention 操作中，我们可以将其拓展为更多的部分。

首先，对于注意力矩阵，使用替代的注意力矩阵 $A(q,k)$，其计算方式为:

$$A(q,k) = D_q S_\varepsilon D_k S_\varphi \tag{7.10}$$

其中，$D_q, D_k \in \mathbb{R}^{L\times L}$ 分别为 q,k 的对角矩阵。S_ε, S_φ 为 Toeplitz 矩阵，其参数由 SSM 决定。

因此，3 个部分的 self-attention 操作可以写为:

$$H_3(q,k,v) = A(q,k)v = D_q S_\varepsilon D_k S_\varphi v \tag{7.11}$$

拓展到多个部分，我们令 $D_x^n = \mathrm{diag}(x^n) \in \mathbb{R}^{L\times L}$，$S_h^n$ 为 Toeplitz 矩阵，来源于 filter h^n，则有:

$$y = H(u)v = D_x^N S_h^N D_x^{N-1} S_h^{N-1} ... D_x^1 S_h^1 v \tag{7.12}$$

3.2.5 Hyena filter

Hyena filter 采用 FFN 更新:

$$h_t = \mathrm{Window}(t) \cdot (\mathrm{FFN} \circ \text{Positional Encoding})(t) \tag{7.13}$$

3.2.6 算法

Algorithm 1: Projection

Require $u \in \mathbb{R}^{L\times D}$

在 dim L 上: $z = \mathrm{Linear}(u), \quad \mathrm{Linear}: \mathbb{R}^D \to \mathbb{R}^{(N+1)D}$

在 dim D 上: $z = \mathrm{DepthwiseConv1d}(h, z)$

reshape: 将 z 拆分为 $v, x^1, ..., x^N \in \mathbb{R}^{D\times L}$

Return $(v, x^1, ..., x^N)$

Algorithm 2: Hyena Filter

Require 序列长度 L，Positional Embedding Dim D_e

$t = \mathrm{PositionalEncoding}(L), \quad t \in \mathbb{R}^{L\times D_e}$

在 dim L 上: $h = \mathrm{FFN}(t), \quad \mathrm{FFN}: \mathbb{R}^{D_e} \to \mathbb{R}^{ND_e}$

reshape: $h \in \mathbb{R}^{N\times D\times L}$

$h = h \cdot \mathrm{Window}(t), \quad h \in \mathbb{R}^{N\times D\times L}$

split: $h = (h^1, ..., h^N)$

Return $(h^1, ..., h^N)$

Algorithm 3: Forward

Require $u \in \mathbb{R}^{L \times D}$, order N operator, PED D_e

$x^1, ..., x^N, v = \text{Projection}(u)$

$h^1, ..., h^N = \text{HyenaFilter}(L, D_e)$

for $n = 1, ..., N$ **do**

  在 dim$D$上: $v_t \leftarrow x_t^n \cdot \text{FFTConv}(h^n, v)_t$

end for

Return v

第四章 其他算法

4.1 Noise-contrastive estimation

https://proceedings.mlr.press/v9/gutmann10a

4.1.1 数学推导

假设观察到数据的分布（概率密度函数 pdf）为 $p_d(\cdot)$，考虑 pdf 由参数 $\boldsymbol{\alpha}$ 决定，因此可以认为 pdf 属于参数家族 $\{p_m(\cdot;\boldsymbol{\alpha})\}_{\boldsymbol{\alpha}}$.其中 $\boldsymbol{\alpha}$ 为参数向量，且存在某个 $\boldsymbol{\alpha}^*$ 使得 $p_d(\cdot)=p_m(\cdot;\boldsymbol{\alpha}^*)$。

这里的问题是，如何在观察数据的基础上，通过最大化目标函数去估计参数 $\boldsymbol{\alpha}^*$. 我们知道的是，不管参数估计结果如何，都一定满足:

$$\int p_m(\boldsymbol{u};\hat{\boldsymbol{\alpha}})d\boldsymbol{u}=1 \tag{8.1}$$

为了绕过这个限制，并且保证为一，我们可以考虑使用归一化的方法，即:

$$p_m(\cdot;\boldsymbol{\alpha})=\frac{p_m^0(\cdot;\boldsymbol{\alpha})}{Z(\alpha)}d\boldsymbol{u}, Z(\boldsymbol{\alpha})=\int p_m^0(\boldsymbol{u};\boldsymbol{\alpha})d\boldsymbol{u} \tag{8.2}$$

使用归一化避免积分限制

这里，$p_m^0(\cdot;\boldsymbol{\alpha})$ 不一定要满足 Eq. (8.1) 的限制，可以是一个形式类似 pdf 的函数。但是，归一化系数 $Z(\boldsymbol{\alpha})$ 包括积分，通常是难以计算的。为了避免直接计算 $Z(\boldsymbol{\alpha})$ 的积分，我们可以将其考虑为一个新参数，通过优化的方法估计 $Z(\boldsymbol{\alpha})$. 即:

要尽量避免积分，积分计算困难

$$p_m(\cdot;\boldsymbol{\theta})=\frac{p_m^0(\cdot;\boldsymbol{\alpha})}{C}, \theta=\{\boldsymbol{\alpha},c\}, c=Z(\boldsymbol{\alpha}) \tag{8.3}$$

问题出现在使用最大似然估计 MLE 时:

$$\begin{aligned}\hat{\boldsymbol{\theta}}&=\arg\max_{\boldsymbol{\theta}}\sum_{\boldsymbol{u}}\log p_m(\boldsymbol{u};\boldsymbol{\theta})\\&=\arg\max_{\boldsymbol{\theta}}\sum_{\boldsymbol{u}}\log\frac{p_m^0(\boldsymbol{u};\boldsymbol{\alpha})}{c}\\&\Rightarrow c\to 0\end{aligned} \tag{8.4}$$

会导致 c 趋近于无穷小，这样的结果显然无效。

为了解决这样的问题，作者提出了一种新的估计方法，即 Noise-contrastive estimation。其基本思想是，引入噪声分布 $p_n(\cdot)$，通过比较 $p_m(\cdot;\boldsymbol{\alpha})$ 和 $p_n(\cdot)$ 的相似性来估计参数 $\boldsymbol{\alpha}$.

从本质上来说，区分噪声与观测数据属于二分类问题。而二分类问题我们可以使用 logistic 回归解决，即: Logistic 回归通过使用 sigmoid 函数将线性组合的输入映射到 0 到 1 之间的输出，以估计某个事件发生的概率。其模型形式为:

$$\begin{aligned}p(C=1|\boldsymbol{u})&=\sigma(\boldsymbol{u}^\top\boldsymbol{\theta})=\frac{1}{1+\exp(-\boldsymbol{u}^\top\boldsymbol{\theta})}\\p(C=0|\boldsymbol{u})&=1-p(C=1|\boldsymbol{u})=1-\sigma(\boldsymbol{u}^\top\boldsymbol{\theta})\end{aligned} \tag{8.5}$$

其中，$C=1$ 代表为观测数据，$C=0$ 代表为噪声数据。

当进行回归时，我们的目标函数为:

$$\begin{aligned} L(\boldsymbol{\theta}) &= \sum_{t=1}^{T} \log p(C_t|\boldsymbol{u}_t;\boldsymbol{\theta}) \\ &= \sum_{t=1}^{T} C_t \log p(C=1|\boldsymbol{u}_t) + (1-C_t)\log p(C=0|\boldsymbol{u}_t) \end{aligned} \tag{8.6}$$

对于观测的数据: $X=\{\boldsymbol{x}_1,\boldsymbol{x}_2,...,\boldsymbol{x}_T\}$，我们引入相同大小的噪声数据 $Y=\{\boldsymbol{y}_1,\boldsymbol{y}_2,...,\boldsymbol{y}_T\}$. 根据 logistic 回归，我们令 $U=X\cup Y=\{\boldsymbol{u}_1,\boldsymbol{u}_2,...,\boldsymbol{u}_{2T}\}$.对于每一个 $\boldsymbol{u}_t \mapsto C_t=\{0,1\}$，当 $\boldsymbol{u}_t$ 为观测数据时，$C_t=1$；当 $\boldsymbol{u}_t$ 为噪声数据时，$C_t=0$.

$p_m(\boldsymbol{u};\boldsymbol{\theta})$，即通过优化获得参数 $\boldsymbol{\theta}$.因此，现在需要做的是获得目标函数 $L(\boldsymbol{\theta})$，且根据 logistic 回归的 Eq. (8.6)，我们需要求出 $p(C=1|\boldsymbol{u};\boldsymbol{\theta})$

直接类比 logistic 回归的结果即可

我们知道，观测数据的 pdf 为等价于 $C_t=1$ 时对应的条件分布；噪声数据的 pdf 为等价于 $C_t=0$ 时对应的条件分布:

$$p_m(\boldsymbol{u};\boldsymbol{\theta})=P(\boldsymbol{u}|C=1;\boldsymbol{\theta}), p_n(\boldsymbol{u};\boldsymbol{\theta})=P(\boldsymbol{u}|C=0;\boldsymbol{\theta}) \tag{8.7}$$

我们同时知道，C 的分布为 bernoulli 分布，有 $P(C=1)=P(C=0)=1/2$. 因此，我们可以使用贝叶斯来计算:

$$\begin{aligned} P(C=1|\boldsymbol{u};\boldsymbol{\theta}) &= \frac{P(C=1\cup \boldsymbol{u};\boldsymbol{\theta})}{P(\boldsymbol{u};\boldsymbol{\theta})} \\ &= \frac{P(\boldsymbol{u}|C=1;\boldsymbol{\theta})\cdot P(C=1;\boldsymbol{\theta})}{P(\boldsymbol{u}|C=1;\boldsymbol{\theta})\cdot P(C=1;\boldsymbol{\theta}) + P(\boldsymbol{u}|C=0;\boldsymbol{\theta})\cdot P(C=0;\boldsymbol{\theta})} \\ &= \frac{p_m(\boldsymbol{u};\boldsymbol{\theta})}{p_m(\boldsymbol{u};\boldsymbol{\theta})+p_n(\boldsymbol{u};\boldsymbol{\theta})} \\ &= h(\boldsymbol{u};\boldsymbol{\theta}) \\ P(C=0|\boldsymbol{u};\boldsymbol{\theta}) &= 1-h(\boldsymbol{u};\boldsymbol{\theta}) \end{aligned} \tag{8.8}$$

代入 Eq. (8.6)，我们可以得到

$$L(\boldsymbol{\theta})=\sum_t C_t \log h(\boldsymbol{u}_t;\boldsymbol{\theta}) + (1-C_t)\log(1-h(\boldsymbol{u}_t;\boldsymbol{\theta})) \tag{8.9}$$

考虑到 $\boldsymbol{u}_t \mapsto C_t=\{0,1\}$，我们可以化简上述公式为:

$$L(\boldsymbol{\theta})=\sum_t \log h(\boldsymbol{x}_t;\boldsymbol{\theta}) + \log(1-h(\boldsymbol{y}_t;\boldsymbol{\theta})) \tag{8.10}$$

我们有 $2T$ 个样本，因此对目标函数平均化:

$$L(\boldsymbol{\theta})=\frac{1}{2T}\sum_t \log h(\boldsymbol{x}_t;\boldsymbol{\theta}) + \log(1-h(\boldsymbol{y}_t;\boldsymbol{\theta})) \tag{8.11}$$

解决:
1. 损失值不可比
2. 梯度下降快
3. 数值不稳定

参考文献

[1] A. G. Howard, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[2] J. Wang, "An intuitive tutorial to Gaussian processes regression," *Computing in Science & Engineering*, 2023.

[3] J. Goodman, "Lecture Notes on Monte Carlo Methods." [Online]. Available at: https://math.nyu.edu/~goodman/teaching/MonteCarlo2005/notes/VarianceReduction.pdf

[4] A. Mnih and K. Gregor, "Neural variational inference and learning in belief networks," in *International Conference on Machine Learning*, 2014, pp. 1791–1799.

[5] S. Gu, S. Levine, I. Sutskever, and A. Mnih, "Muprop: Unbiased backpropagation for stochastic neural networks," *arXiv preprint arXiv:1511.05176*, 2015.

[6] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv preprint arXiv:1611.00712*, 2016.

[7] G. Tucker, A. Mnih, C. J. Maddison, J. Lawson, and J. Sohl-Dickstein, "Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[8] ymhuang, "机器学习工具（二）Notes of Optimal Transport," Jul. 2024, [Online]. Available at: https://zhuanlan.zhihu.com/p/82424946

[9] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, 2017, pp. 214–223.

[10] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[11] M. Poli *et al.*, "Hyena hierarchy: Towards larger convolutional language models," in *International Conference on Machine Learning*, 2023, pp. 28043–28078.

[12] Y. Chen, A. Lv, J. Luan, B. Wang, and W. Liu, "HoPE: A Novel Positional Encoding Without Long-Term Decay for Enhanced Context Awareness and Extrapolation," *arXiv preprint arXiv:2410.21216*, 2024.