感兴趣的一些统计学习模型

赵显文

2025 年 6 月 1 日

,

前言

这篇笔记涵盖了一些感兴趣的统计学习的话题

具体地，包括 EM 算法及应用，条件随机场，MCMC，最大熵学习，扩散模模型，SAC，LDA, 变分推断，VAE，Gibbs 采样，再生核 Hibbert 空间与最大均值差异（MMD）等随机话题。本笔记将清晰表述每个模型所有解决的基本问题并给出一些 intuition 解释，力求从理论推导详尽表达其中原理，并整理每个相关程序。

赵显文

2025 年 6 月 1 日

目录

第一章 EM 算法原理及应用 **1**

1.1 EM 算法原理简介 1

1.1.1 无隐变量概率模型的参数估计方式 1

1.1.2 EM 算法解析 2

1.2 EM 算法收敛性证明及广义 EM 4

1.2.1 EM 收敛性证明 4

1.2.2 广义 EM 5

1.3 EM 算法案例 6

1.3.1 Kmean 算法中的 EM 7

1.3.2 GMM 算法中的 EM 7

1.3.3 HMM 算法中的 EM 7

1.3.4 CRF 算法中的 EM 7

第二章 变分推断及其应用 **8**

2.1 基础变分推断—基于平均场 9

2.1.1 VI 的解析形式及迭代求解方法 11

2.1.2 基于（随机）梯度下降的 VI 12

2.2 应用实例—VAE 15

2.2.1 重参数化技巧—神经网络求分布 15

2.2.2 VAE 解析 16

第三章 Monte Carlo Markov Chain Method 20

3.1 Markov Process 20

3.2 采样 20

3.2.1 Metropolis-Hashing 采样 20

3.2.2 Gibbs 采样 20

3.2.3 Langevin 采样 20

第四章 条件随机场模型的原理及应用 21

4.1 概率图模型及相近模型导言 21

4.2 线型链条件随机场模型详解 22

4.3 条件随机场模型学习及推理 24

4.3.1 前向-后向算法求边缘分布 24

4.3.2 条件随机场的预测任务 26

4.4 Bert-Bilstm-CRF 实体命名实战 28

4.4.1 基础模型 Bilstm-CRF 解析 28

4.4.2 Bert-Bilstm-CRF 33

第五章 Diffusion Model 35

5.1 Score Based Model 35

5.2 DDPM 推导 35

5.3 SDE 视角下的 Diffusion Model 35

5.4 ODE 视角下的 Diffusion Model 35

5.5 Flow Matching 35

5.5.1 Flow based Model 35

第六章 Kalman Filter 36

第一章　EM 算法原理及应用

1.1　EM 算法原理简介

EM 算法就是期望最大化算法。**解决的是含有隐变量的概率模型的模型参数估计问题，具体解决的是频率学派的极大似然估计 MLE，和贝叶斯学派的极大后验概率估计 MAP 的迭代求解问题**。

那其实对于没有隐变量的概率模型常用的模型参数估计也就是矩估计、MLE，贝叶斯估计如 MAP。

1.1.1　无隐变量概率模型的参数估计方式

对于样本 $x_1, x_2, \cdots, x_n \sim f(x, \theta)$,$\theta$ 是概率模型的参数，我们想要根据观测到的样本进行 θ 的估计。

概率模型大致有两种看法，频率学派的观点是模型的参数是确定性的，而贝叶斯学派认为模型的参数也是随机的，服从某一个先验分布 $p(\theta)$。沟通两种观点的桥梁是**贝叶斯公式：**

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} = \frac{p(x|\theta)p(\theta)}{\int_\theta p(x|\theta)p(\theta)\,dx}$$

参数估计可以看作求 $p(\theta|x)$ 的一个数字统计量。那就需要一个 “损失函数” $\mathcal{L}(\theta, \widehat{\theta})$, 也就是通过优化这个损失函数获取我要得到的量$\widehat{\theta}$.

常用的损失函数有如下的三种格式：

① $\mathcal{L}(\theta, \widehat{\theta}) = (\theta - \widehat{\theta})^2$

② $\mathcal{L}(\theta,\widehat{\theta})=\left|\theta-\widehat{\theta}\right|$

③ $L(\theta,\widehat{\theta})=\begin{cases}1 & \left|\theta-\widehat{\theta}\right|\leq\delta\\ 0 & \left|\theta-\widehat{\theta}\right|>\delta\end{cases}$

1.1.2　EM 算法解析

EM 算法就是用迭代的方式求解含隐变量的 MLE 问题，具体的迭代公式（最原始的 EM 算法）如下

$$\begin{aligned}\theta^{t+1}&=\arg\max_{\theta}\int_{z}p(z|x,\theta^{t})\log P(x,z|\theta)dz\\&=\arg\max_{\theta}E_{z\sim p(z|x,\theta^{t})}[\log P(x,z|\theta)]\end{aligned}$$

因此，E 步就是求期望，其核心就是求出后验分布 $p(z|x,\theta^t)$,M 步就是求出使期望最大的那个 $\theta\rightarrow\theta^{t+1}$。下面开始推导这个形式的含义, 即为什么 z 要服从那个分布，也就是为什莫 E 步要那样操作，至于 M 步取最大的操作其实就是 EM 算法收敛性的保障

我们的目标函数是 $\log P(x|\theta)$,其实在一些文献中，$P(x|\theta)$ 也常用 $P_\theta(x)$ 表示，有放松关注对象的意思.条件概率公式 $P(z|x)=\frac{P(x,z)}{P(x)}$

$$\begin{aligned}\log P(x|\theta)&=\log P(x,z|\theta)-\log P(z|x,\theta)\leftarrow(\text{给}z\text{加一个先验}q(z))\\&=\log\frac{P(x,z|\theta)}{q(z)}-\log\frac{P(z|x,\theta)}{q(z)}\end{aligned}$$

为什莫要引入 q(z)? 因为对于目标函数 $P(x|\theta)$, 我们根本不知道关于隐变量 z 的有用的信息，我们不知道后验分布 $P_\theta(z|x)$, 和完全数据分布 $P_\theta(x,z)$, 因此我引入一种分布 $q(z)$ 当作一种先验，这个分布是我引入的我完全可以控制他。引入已知信息显然帮助我们进行下去。

等式两边对 $z \sim q(z)$ 求期望，显然有

$$\log P(x|\theta) = \underbrace{\int_z q(z) \log \frac{p(x,z|\theta)}{q(z)} dz}_{ELBO} \underbrace{- \int_z q(z) \log \frac{p(z|x,\theta)}{q(z)} dz}_{KL(q(z)||p(z|x,\theta))}$$

其实这种形式很常见诸如变分推断以及相关的新兴的生成模型 VAE, Diffusion Model。等式右边第一项称为证据下界（就是先验和完全数据分布的相对熵), 第二项（含符号, 就是先验与后验的相对熵)。由于 KL 散度衡大于等于 0 并且当且仅当两个分布相同（测度论上就是几乎处处相等）KL 散度为 0 , 因此 ELBO 就是对数似然函数的下界。最大不了对数似然函数, 那退而求其次最大化证据下界 ELBO 也行，并且为了使 ELBO 更大程度上接近对数似然函数，显然要保证第二项 KL 散度尽量趋近于 0，因此只要让我们加的先验分布 $q(z) \to p_\theta(z|x)$ 就好了。那么我们的目标 $\max \log P(x|\theta)$, 就可以转化为两步, 即先使 $q(z) \to p_\theta(z|x)$ 保证 KL 散度接近于 0, ELBO 接近目标函数（对数似然函数)，这样之后直接 $\max\limits_{KL \to 0} ELBO$ 等价于 $\max_\theta \log P(x|\theta)$.

写成迭代式，很自然的有

$$q(z) = P(z|x, \theta^t)$$

将 $q(z)$ 带入 ELBO 的公式有

$$\begin{aligned}
\theta^{t+1} &= \arg\max_\theta \log(x|\theta) = \arg\max_\theta ELBO \\
&= \arg\max_\theta \int_z P(z|x,\theta^t) \log \frac{p(x,z|\theta)}{P(z|x,\theta^t)} dz \\
&= \arg\max_\theta \{ \int_z P(z|x,\theta^t) \log P(x,z|\theta) dz - \int_z P(z|x,\theta^t) \log P(z|x,\theta^t) \} \\
&= \arg\max_\theta \{ \int_z P(z|x,\theta^t) \log P(x,z|\theta) dz - \underbrace{H(P(z|x,\theta^t))}_{\text{熵，与}\theta\text{无关}} \} \\
&= \arg\max_\theta \int_z P(z|x,\theta^t) \log P(x,z|\theta) dz == \arg\max_\theta E_{z \sim p(z|x,\theta^t)} [\log P(x,z|\theta)]
\end{aligned}$$

上文是从 ELBO + KL 入手解决的对数似然函数最大化问题，下面用一个更直接的方式，本质上两者都是一样的基于 Jesson Inequation。KL 散度的

上述性质本质上就是通过 Jesson 不等式（凸函数的性质）。当然了那个额外的 $q(z)$ 也是不可或缺的。

$$\begin{aligned}\log P(x|\theta) &= \log \int_z P(x,z|\theta)dz = \log \int_z \frac{P(x,z|\theta)}{q(z)} q(z)dz \\ &= \log E_{q(z)}[\frac{P(x,z|\theta)}{q(z)}] \to \log \text{凹函数，期望近似平均} \\ &\geq E_{q(z)}[\log \frac{P(x,z|\theta)}{q(z)}] =\triangleq ELBO\end{aligned}$$

由凹函数（log）的性质，等号成立当且仅当 $\frac{P(x,z|\theta)}{q(z)} = Constant, \forall z$，因而 $q(z) = \frac{P(x,z|\theta)}{C}$, 等式左右对 z 积分, 有 $1 = \frac{1}{C}\int_z P(x,z|\theta)dz = \frac{1}{C}P(x|\theta)$, 因而 $C = P(x|\theta)$, 所以有

$$q(z) = \frac{P(x,z|\theta)}{C} = \frac{P(x,z|\theta)}{P(x|\theta)} = P(z|x,\theta)$$

同样的说明了当给定的 $q(z) \to p_\theta(z|x)$ 时,优化对数似然等价于优化 ELBO, 然后就可以同上述一样的做法推出迭代公式。

1.2　EM 算法收敛性证明及广义 EM

1.2.1　EM 收敛性证明

$$\begin{aligned}\theta^{t+1} &= \arg\max_\theta \int_z p(z|x,\theta^t) \log P(x,z|\theta)dz \\ &= \arg\max_\theta E_{z\sim p(z|x,\theta^t)}[\log P(x,z|\theta)]\end{aligned}$$

显然概率有上界，我们只要证单调性：$\log P(x|\theta^{t+1}) \geq \log P(x|\theta^t)$ 由于条件概率公式有 $\log P(x|\theta) = \log P(x,z|\theta) - \log P(z|x,\theta)$，等式两边都对 $z \sim$

$p(z|x,\theta^t)$ 求期望，显然左边与 z 无关，不变。

$$右边 = \underbrace{\int_z p(z|x,\theta^t)\log P(x,z|\theta)dz}_{\mathcal{Q}(\theta,\theta^t)} - \underbrace{\int_z p(z|x,\theta^t)\log P(z|x,\theta)dz}_{\mathcal{H}(\theta,\theta^t)}$$

因此 $\log P(x|\theta) \triangleq \mathcal{Q}(\theta,\theta^t) - \mathcal{H}(\theta,\theta^t)$, 要证 $\log P(x|\theta^{t+1}) \geq \log P(x|\theta^t)$, 只需证 $\mathcal{Q}(\theta^{t+1},\theta^t) \geq \mathcal{Q}(\theta^t,\theta^t)$ 且，$\mathcal{H}(\theta^{t+1},\theta^t) \leq \mathcal{H}(\theta^t,\theta^t)$。

1. $\theta^{t+1} = \arg\max_\theta \int_z p(z|x,\theta^t)\log P(x,z|\theta)dz = \arg\max_\theta \mathcal{Q}(\theta,\theta^t)$, 因此 $\mathcal{Q}(\theta^{t+1},\theta^t) \geq \mathcal{Q}(\theta^t,\theta^t)$。

2. $\mathcal{H}(\theta^{t+1},\theta^t) - \mathcal{H}(\theta^t,\theta^t) = \int_z p(z|x,\theta^t)\log\frac{P(x,z|\theta^{t+1})}{P(x,z|\theta^t)}dz$, 显然 $\mathcal{H}(\theta^{t+1},\theta^t) - \mathcal{H}(\theta^t,\theta^t) = -KL[p(z|x,\theta^t)||P(x,z|\theta^{t+1})] \leq 0$, 因此 $\mathcal{H}(\theta^{t+1},\theta^t) \leq \mathcal{H}(\theta^t,\theta^t)$

1.2.2　广义 EM

其实从解析上述“狭义”EM，我们就知道的 EM 算法的关键：

1. 引入一个分布 q(z), 使 q(z)→ 后验 $p_\theta(z|x)$

2. 优化对数似然转为约束其证据下界 ELBO

狭义的 EM 直接 $q(z) = P(z|x,\theta^t)$，很显然当问题本身比较复杂时 $P(z|x,\theta^t)$ 可能就是 intractable 的，算不出来，表达不出来。即使表达出来 M 步还有个求期望（对 $z \sim P(z|x,\theta^t)$ ）这个更难了，这将引出下两个话题“变分推断”以及 MCMC 方法（MonteCarlo-Hasting 或者 Gibbs Sampling 技术或者重要性采样）。

显然本节广义 EM 其实就是解决 $P(z|x,\theta^t)$ 表达不出的问题，因为后面确实是个问题不过还是有方法去解决的，也就是可解的，因而这里研究的 EM 的范畴。当然融入变分推断和蒙特卡洛法参与 EM 计算的算法就是 EM 的变种对应 VBEM 和 MCEM。当然了解决这个问题就是转化为优化问题：

$$\widehat{q}(z) = \arg\max_{q(z)} \mathcal{D}_{KL}[q(z)||P(z|x,\theta)]$$

考虑到 $\log P(x|\theta) = ELBO + KL(q||p(z|x,\theta))$, 记 $ELBO \triangleq \mathcal{L}(\theta, q(z))$ 因此，写成迭代式（广义 EM）

1. E 步：固定 θ, 因为这是学习任务所以 x 已知，等式左侧一定，因而可通过下式求出 $\widehat{q}(z)$

$$\widehat{q}(z) = \arg\min_{q(z)} \mathcal{D}_{KL}[q(z)||P(z|x,\theta)] = \arg\max_{q(z)} \mathcal{L}(q,\theta)$$

2. M 步：固定 $\widehat{q}(z)$, 求出新的 θ, 之后的 E 步用此 θ 计算

$$\widehat{\theta} = \arg\max_{\theta} \mathcal{L}(\widehat{q}(z),\theta)$$

因此就像采用坐标轮换法求解 $\max \mathcal{L}(q,\theta)$ 问题

再看一下这个 $\mathcal{L}(q,\theta)$

$$\begin{aligned}\mathcal{L}(q,\theta) = ELBO &= E_{q(z)}[\log \frac{P(x,z|\theta)}{q(z)}] \\ &= E_{q(z)}[\log P(x,z|\theta)] + H[q(z)]\end{aligned}$$

其实在最大化 $\mathcal{L}(q,\theta)$ 时（执行 E 步），我们其实隐含者最大化我们引入的先验分布 $q(z)$ 的熵，这很 make sense! 一个损失函数就应该有理论基础以及一些 make sense 的内容。并且 $q(z)$ 的选择既要熵最大又要接近后验 $P_\theta(z|x)$, 这里面就是一种 trade off, 一种博弈，这也是热衷于介绍 EM 的原因。

1.3　EM 算法案例

EM 算法是解决生成模型的学习问题，生成模型就是建立观测变量 x，和隐变量 z 的联合概率。学习问题体现在用对 MLE 采用 EM 算法的方式求解模型参数的估计。因此凡是类似的生成模型（采用隐变量生成观测变量）的方式的模型如 GMM，HMM，CRF 的学习过程都可以采用 EM 算法取求解。在解决这些具体问题中，显然最核心的就是 z 和 x 间的交互（概率图）。

1.3.1 Kmean 算法中的 EM

1.3.2 GMM 算法中的 EM

1.3.3 HMM 算法中的 EM

1.3.4 CRF 算法中的 EM

第二章　变分推断及其应用

总的来讲, 变分推断是一种用来求解后验分布 $P(\theta|x)$ 一种方法, 是在 Bayes 框架下的一种近似推断（Inference）, 其中 x 是观测变量，θ 是隐变量，包含模型参数。因为 Bayes 角度, 模型的参数也是随机的。而一些隐变量之所以会出现，只是辅助我们进行建模的一种" 假设"。因为引入隐变量最起码可以解析的表达一下，要不然只有一堆数据 x 啥也不知道。

$$P(\theta|x) = \frac{P(\theta) * P(x|\theta)}{P(x)} = \frac{P(\theta) * P(x|\theta)}{\int_\theta P(\theta)P(x|\theta)d\theta}$$

显然通过上述后验分布的公式可以看出，分布要积分，这个很难的。但是你的后验还必须要求，因为它是 Bayes 决策的一个基础。举一个基础的决策的例子，就是给定一组观测变量 x, 之后来了一个新的样本 $\tilde{x}$, 目标是求 $P(\tilde{x}|x)$. 显然要通过隐变量（所以说隐变量是我们研究问题的较量）。

$$P(\tilde{x}|x) = \int_\theta P(\tilde{x}, \theta|x)d\theta = \int_\theta P(\tilde{x}|\theta, x) * P(\theta|x)d\theta = \mathbb{E}_{\theta|x}[P(\tilde{x}|\theta)]$$

上述公式中最后的期望里面把 x 去掉了，这是因为在 θ 固定情况下，$x, \tilde{x} \sim i.i.d.$。当然了在 θ 也是随机的条件下，也就是等号的最左端，$x, \tilde{x}$ 并不独立。

不论如何，后验在贝叶斯里面还是很重要的。我们一定要求出来（解析，数值，精确，确定性近似，随机近似，都行，只要求出）。显然在 Bayes 框架下，我们可以引入先验分布 $q(z)$, 因为这是我们引入的，所以可以辅助我们进行计算后验。这个在 EM 算法中可见一般。并且如果加的好，具体

怎么个好法，这里不得不提出共轭分布：就是在似然函数一定下，先验分布和后验分布有相同的形式。如果组成共轭分布，那样后验的结构就确定下来了，直接求参数就行。但是这样的共轭分布少之又少。最常见的是先验是 Guass 分布，后验也是 Guass 分布，并且后验的参数体现了给定观测数据 x 的方差和先验分布的 trade off. 毕竟方差大，可信度低，这很 make sense! 另一个例子是 Beta 分布的后验还是 Beta 分布，再似然分布为伯努利分布时。

但是这样的例子太少了，可以说是神来之笔，我们遇到的问题不会这么巧。因此我们要探究普适性的方法。本章的变分推断就是一种普适性的方法。他是一种近似推断，并且不像 MCMC 那样，VI 是确定性的近似推断。

在介绍完基础的变分推断的原理后，我们介绍两个深度生成模型。本质都是 VAE，都用了变分推断的技术。编码器就是为了建模后验分布 $P(\theta|x)$，而解码器就是根据后验生成新的样本，就类似与上面的决策过程，建模 $P(\tilde{x}|\theta)$。当然了要给出一个确定量，其实本质上，Bayes 决策根据后验任何一个采样出来的样本 $\hat{x}$ 都是贝叶斯估计量，当然也可以用均值，中位数那些。VAE 就是假设后验是高斯分布，直接预测均值的。

2.1　基础变分推断—基于平均场

和 EM 一样，要引入一个类似于先验的分布 $q(z)$. 我们可以写出如下的等式，用 X, Z 分别表示观测数据（多个样本）和隐变量对应的数据（对应的多个样本)。

Note：其实 $q(z)$ 其实可以看作是 $q(z|X)$，因为这是我们人为引入的，他的作用就是接近后验 $P(Z|X)$，关键是这个 X 确实是观测的，人为引入 $q(z)$ 的结构可能确实要参考一下 X 的信息。并且只是引入好 $q(z)$ 的结构且和原来的 $P(X)$ 没有任何瓜葛也就是独立，也是需要优化具体找出

来所以就直接简记 q(z)。另外其实这里 P 省略了其中的参数 $P_\theta(X)$, 因为这里主要考虑的是隐变量与模型的参数 θ 关系不大，方便起见省略了。这里 $X = \{x^{(i)}\}_{i=1}^n, Z = \{z^{(i)}\}_{i=1}^n$, 为观测样本集和对应的隐变量样本集，且$z^{(i)} \sim P(z|x^{(i)})$。

Note：这里 X，Z 代表多个样本还是一个样本本质上是一样的，因为样本的独立同分布假设，可以把后面的 X，Z 看作一个样本。

$$\begin{aligned}\log P(X) &= \log P(X,Z) - \log P(Z|X) \\ &= \log \frac{P(X,Z)}{q(z)} - \log \frac{P(Z|X)}{q(z)}\end{aligned}$$

两边对 $z \sim q(z)$ 求期望，可得

$$\log P(X) = ELBO + \mathcal{D}_{KL}[q(z)||P(Z|X)] \triangleq \mathcal{L}(q) + \mathcal{D}_{KL}[q(z)||P(Z|X)]$$

我们变分推断的目的就是估计处这个后验 $P(Z|X)$ 或者写成 $P(z|x)$ 形式，因为每个样本都满足独立同分布假设), 显然我们让后一项 $\mathcal{D}_{KL}(q||P(Z|X)) \to 0$, 那么 $q(z)$ 就是我要的近似解。所以我们的目标就是求解下述优化问题。

$$\widehat{P}(z|X) = q^*(z) = \arg\min_q \mathcal{D}_{KL}(q(z)||P(Z|X))$$

从上式可以看出，给定观测样本 X 后，$P(X)$ 是确定的，那么问题就转化为下式，这是 **VI 与 EM 的不同**。

$$\widehat{P}(z|X) = q^*(z) = \arg\max_q \mathcal{L}(q)$$

所以这个变分推断的“变分”就是变分法的变分，因为最优化 $\mathcal{L}(q)$ 而来的其中 $q(z)$ 是个函数。那我们显然得给 $q(z)$ 加点约束，限制一下求解空间，并且我们之所以这样做完全是为了用 $q(z)$ 逼近我们的后验 $P(Z|X)$, 为的就是好算!。那就设 $q(z) = \prod_{i=1}^M q_i(z_i)$. 也就是假设 z 这个隐变量有多个元素，可以分成 M 个团，每个团里面有 z 的几个分量，团 z_i 与团 $z_j, j \neq i$ 之间相

互独立，这个假设在 CRF 里面也用到过。当然了这里 z, z_i 都是多维变量，且 z_i 是 z 中的几个分量。这其实就是基于统计物理学里的平均场理论。

其实这就是变分推断了。下面就是优化问题了。

2.1.1 VI 的解析形式及迭代求解方法

引入了平均场假设其实整个优化的结果可以更进一步解析！

$$\mathcal{L}(q) = \underbrace{\int_{z\sim q(z)} q(z) * \log P(X,Z)dz}_{①} - \underbrace{\int_{z\sim q(z)} q(z) * \log q(z)dz}_{②}$$

那带入 $q(z)$，一项一项看 $q_i(z_i)$ 长啥样。为方便期间简记 q_i

$$\begin{aligned}
① &= \int_z \prod_{i=1}^{M} q_i * \log P(X,Z)dz \\
&= \int_{z_i} q_i * \left(\int_{z_j \neq z_i} \prod_{j\neq i} q_j * \log P(X,Z)dz_j \right) dz_i \\
&= \int_{z_i} q_i(z_i) * \mathbb{E}_{z_j, j\neq i} \left[\log P(X,Z)\right] dz_i \\
&\triangleq \int_{z_i} q_i(z_i) * \log \widetilde{P}(X,Z_i)dz_i + C_1
\end{aligned}$$

最后一步为了和②凑成 KL 散度，把上一步的期望或者上上步的积分记成那个符号（*Just some notation!*）并赋予**概率**的意义，因而会通过 e^{C_1} 来**归一化**，log 后有 C_1 这个常数项，而因为是对整体求期望，因为 Z 中的每一个样本 $z^{(i)}$ 也是 $i.i.d.$ 的，因此求期望后，每一个样本对应的除去 i 所在的

团的分量全部被积分积掉了，因此只剩下 $Z_i = \{z_i^{(k)}\}_{k=1}^n$

$$\begin{aligned}
② &= \int_z \prod_{i=1}^M q_i * \sum_{i=1}^M \log q_i dz \\
&= \sum_{i=1}^M \int_{z_i} q_i(z_i) * \log q_i(z_i) dz_i \\
&= \int_{z_i} q_i(z_i) * \log q_i(z_i) dz_i + \underbrace{\sum_{j \neq i} \int_{z_j} q_j(z_j) * \log q_j(z_j) dz_j}_{Constant, C_2}
\end{aligned}$$

因此①-②= $\int_{z_i} q_i(z_i) * \log \frac{\widetilde{P}(X,Z_i)}{q_i(z_i)} dz_i + C = -\mathcal{D}_{KL}\left[q_i(z_i)||\widetilde{P}(X,Z_i)\right]$. 因为平均场假设把 $q(z)$按照分量的形式 分成了相互独立的 M 个部分 $q_i(z_i)$, 因此要求最优的 $q^*(z)$, 分别求出最优的 $q_i^*(z_i)$:

$$q_i^*(z_i) = \arg\min_{q_i} \mathcal{D}_{KL}\left[q_i(z_i)||\widetilde{P}(X,Z_i)\right]$$

那其实根据解析形式的迭代求解就出来了:

1. 初始化 $q(z)$, 也就是初始化 $q_1, q_2, \cdots, q_M$
2. $\hat{q}_1(z_1) = \int_{z_2} \cdots \int_{z_M} q_2 \cdots q_M \left[\log P_\theta(X,Z)\right] dz_2 \cdots dz_M$
3. $\hat{q}_2(z_2) = \int_{z_1} \int_{z_3} \cdots \int_{z_M} \hat{q}_1 q_3 \cdots q_{M-1} \left[\log P_\theta(X,Z)\right] dz_1 dz_3 \cdots dz_M$
4. ⋮
5. $\hat{q}_M(z_M) = \int_{z_1} \int_{z_2} \cdots \int_{z_{M-1}} \hat{q}_1 \hat{q}_2 \cdots \hat{q}_{M-1} \left[\log P_\theta(X,Z)\right] dz_1 dz_2 \cdots dz_{M-1}$
6. 返回第 2 步，用新的 $\hat{q}_1, \cdots, \hat{q}_M$ 参与计算，直到收敛。

2.1.2　基于（随机）梯度下降的 VI

优化的目标式如下，解决梯度下降的核心是求解梯度，对于变分问题我们可以将待优化函数参数化，将变分问题转为一般优化问题。虽有或多

或少把可行域下降了，但不要忘了我们求的最优的参数 $q(z)$ 本身就是为了最大程度趋近于复杂的后验分布且更方便计算，所以这点损失不用过于关注。

$$\widehat{P}(z|X) = q^*(z) = \arg\max_q \mathcal{L}(q)$$

假设 $q(z)$ 的参数为 ϕ, 那我们的优化目标转为:

$$\phi^* = \arg\max_\phi \mathcal{L}(\phi)$$

. 实际上我们用深度学习去解决问题的时候就是这么做的！我们用一个网络当作函数 q, 输入一个 x, 输出一个采样值 z, 当然了我们可以用重参数化技巧，从另一个分布中采样，而当前的网络只输出一个特定的值如 $q(z)$ 的均值或者方差等等. 不管怎样，我们用网络比如 MLP 当作函数 q，网络的参数 ϕ 就是 $q(z)$ 的参数，通过随机梯度下降优化 ϕ 以获得最优的 $q_\phi(z)$. 一下证明中应用了 $\int$ 和 ∇ 交换的性质

$$\begin{aligned}
\nabla_\phi \mathcal{L}(\phi) &= \nabla_\phi \int_{z\sim q_\phi} q_\phi * [\log P_\theta(X,Z) - \log q_\phi(z)]\, dz \\
&= \int_{z\sim q_\phi} [\nabla_\phi q_\phi] * [\log P_\theta(X,Z) - \log q_\phi(z)]\, dz \\
&\qquad + \int_{z\sim q_\phi} q_\phi * \{\nabla_\phi [\log P_\theta(X,Z)] - \nabla_\phi [\log q_\phi(z)]\}\, dz \\
&= \int_{z\sim q_\phi} [\nabla_\phi q_\phi] * [\log P_\theta(X,Z) - \log q_\phi(z)]\, dz - \int_{z\sim q_\phi} q_\phi * \nabla_\phi [\log q_\phi(z)]\, dz \\
&= \int_{z\sim q_\phi} [\nabla_\phi q_\phi] * [\log P_\theta(X,Z) - \log q_\phi(z)]\, dz - \int_{z\sim q_\phi} \nabla_\phi q_\phi dz \\
&= \int_{z\sim q_\phi} [\nabla_\phi q_\phi] * [\log P_\theta(X,Z) - \log q_\phi(z)]\, dz - \nabla_\phi \underbrace{\int_{z\sim q_\phi} q_\phi(z) dz}_{=1, \nabla C = 0} \\
&= \int_{z\sim q_\phi} [\nabla_\phi q_\phi] * [\log P_\theta(X,Z) - \log q_\phi(z)]\, dz
\end{aligned}$$

下面我们用一个 trick，这个技巧在强化学习中策略梯度推导的模型中使用过.

$$\nabla_\phi \log q_\phi = \frac{\nabla_\phi q_\phi}{q_\phi}$$

因此将 $\nabla_\phi q_\phi$ 带入上述公式，可以看成一个求期望的过程，那么可以用 MC 的方法去采样计算了。

$$\begin{aligned}
\nabla_\phi \mathcal{L}(\phi) &= \int_{z\sim q_\phi} [\nabla_\phi q_\phi] * [\log P_\theta(X,Z) - \log q_\phi(z)]\, dz \\
&= \int_{z\sim q_\phi} q_\phi(z) * [\nabla_\phi \log [q_\phi(z)]]\, [\log P_\theta(X,Z) - \log q_\phi(z)]\, dz \\
&= \mathbb{E}_{z\sim q_\phi} [\nabla_\phi \log [q_\phi(z)]\, (\log P_\theta(X,Z) - \log q_\phi(z))]
\end{aligned}$$

我们可以再用一个 *Reparameter Trick* 来解决 MC 方法中方差大（比如采样 z^1, z^2, 输出的 $q(z^1), q(z^2) < 1$ 比较小，但通过 log 后再微分, 结果差的就很大）的问题。它可以解决采样过程的难微分问题，以及可以减少针对参数求微分后再对参数进行期望运算的高方差问题。本质上就是将微分过程和采样（求期望）过程分离。整体的想法如下，设计合适的分布 $p(\xi)$ 用于采样以及对应的转化函数 $f(\xi)$(由中介分布和目标分布 $p(\xi)$ $q(z)$ 唯一确定)，使得下式等价

$$z \sim q_\phi(z) \Longleftrightarrow f\phi(\xi), \xi \sim p(\xi)$$

举一个常见的例子要从 $z \sim N(\mu, \sigma^2)$ 中采样一个 z, 只需从 $\xi \sim N(0,1)$, 采样一个 ξ, 然后通过一个确定的映射 $f : \xi \mapsto \mu + \xi \cdot \sigma$。

下文公式的第二行本质上是概率统计里面的知识，就是已知 $f : x \longmapsto y$ 和 $x \sim p(x)$, 求 $y \sim q(x)$, 除了要加对应的变换还要加那个雅可比矩阵。

$$\begin{aligned}\nabla_\phi \mathcal{L}(\phi) &= \nabla_\phi \mathbb{E}_{z\sim q_\phi}\left\{[\log P_\theta(X,Z) - \log q_\phi(z)]\right\} \\ &= \nabla_\phi \mathbb{E}_{\xi\sim p(\xi)}\left\{[\log P_\theta(X,Z) - \log q_\phi(z)]\right\} \\ &= \mathbb{E}_{\xi\sim p(\xi)}\left\{\nabla_\phi\left[\log P_\theta(X,Z) - \log q_\phi(z)\right]\right\} \\ &= \mathbb{E}_{\xi\sim p(\xi)}\left\{\nabla_z\left[\log P_\theta(X,Z) - \log q_\phi(z)\right] * \nabla_\phi z\right\} \\ &= \mathbb{E}_{\xi\sim p(\xi)}\left\{\nabla_z\left[\log P_\theta(X,Z) - \log q_\phi(z)\right] * \nabla_\phi f_\phi(\xi)\right\}\end{aligned}$$

因此采用随机梯度下降的公式如下：

$$\phi^{t+1} = \phi^t + \lambda^t \cdot \widetilde{\nabla_\phi \mathcal{L}}(\phi)$$

其中 N 为采样个数，且

$$\widetilde{\nabla_\phi \mathcal{L}}(\phi) = \frac{1}{N}\sum_{i=1}^{N} \nabla_z\left[\log P_\theta(x^{(i)},z) - \log q_\phi(z)\right]\Big|_{z=z^{(i)}} * \nabla_\phi f_\phi(\xi^{(i)})$$

$$z^{(i)} = f_\phi(\xi^{(i)})$$

2.2　应用实例—VAE

2.2.1　重参数化技巧—神经网络求分布

这里只是简短介绍一下为什么现在的生成模型用神经网络去拟合数据的分布。我们从很小的一个实例出发，也就是上文提到的例子。假设我要从分布 $y \sim \mathcal{N}(\mu, \sigma^2)$ 去采样。为啥强调采样呢？其实从正向来看有了分布，分布本身就是一个数学公式，真正我们实际遇到的都是一个一个的样本。那我们可以用一个分布 $\xi \sim q(\xi)$，通过这两个分布的映射 $f : \xi \longrightarrow Y$ 来实现样本 Y 的采样。具体的 $q(\xi)$ 可以取 $\mathcal{N}(0,1)$，只要能找到这样的 f，其实 $q(\xi)$ 取啥都可以。

所以最终的结果是从一个给定的分布 $q(\xi)$ 通过一个映射 f 就可以实现都目标分布 P_{data} 的采样. 因此可以利用神经网络强大的拟合能力从给定分布不断学习 f，来逼近未知的 P_{data}。
那其实就是现在深度生成模型如 VAE，Diffusion Model 的编码过程。

2.2.2 VAE 解析

一句话，VAE 就是就是通过自编码器结构，加上变分推断的方式的生成模型。本质上就是连续维的 GMM。GMM 通过离散的隐变量 z, 来建模样本的生成方式。而考虑到样本的生成只依赖有限个隐变量表达能力有限，因此将隐变量扩展到无限维，那就直接假设这个隐变量服从高斯分布。其实具体的这个隐变量服从什么连续分布其实不重要。只不过高斯分布有普遍性。另外 $P_\phi(X|Z)$ 也可以直接假设，VAE 同 GMM 一样，直接假设 $P_\phi(X|Z) \sim \mathcal{N}(\mu_\phi(z), \sigma^2_\phi(z))$, 也就是服从正太分布。这样在解码阶段，那通过重参数化技巧我可以用网络训练一个函数 $D_\phi(\xi|z), \xi \sim \mathcal{N}(0,1)$ 直接生成一个样本 $\hat{x}$, 但考虑到解码只要输出样本就行了，也不用那么麻烦，因为做了正太分布假设，直接输出 $\hat{x} = \mu_\phi(z) \triangleq f_\phi(z)$, 其中 f 就是一个神经网络。整体的框架如下图所示。因此 VAE 其实就是拓展版的 GMM，只

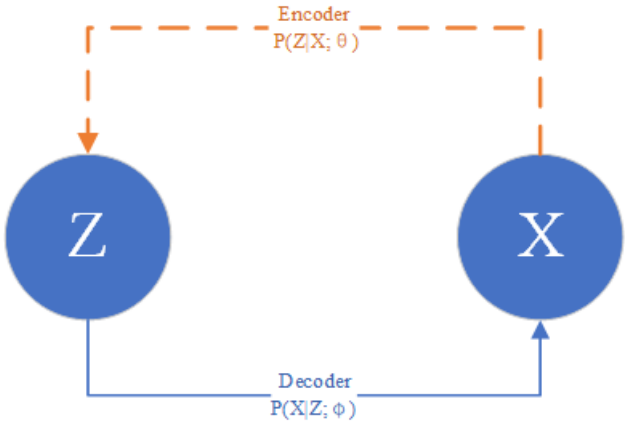

图 2.1: 基于隐变量概率生成模型的整体框架

是模型的参数比如 σ, μ 都通过神经网络来获得，假设的隐变量的先验分布 $q(z) \sim \mathcal{N}(0,1)$ 不再是离散型，而是连续型，并且似然函数的假设都是一样的 $p_\phi(x|z)$ 也服从高斯分布，直接用一个网络拟合他的 $\mu_\phi(z)$ 来表示解码后的样本。剩下的就是求后验 $P_\theta(z|x)$, 这里用了变分推断，采用另一个正太分布 $q_\theta(z|x)$ 来逼近。(这里也是可以理解的，因为假定了似然是正太，且先验分布也是正太，因此后验也是正太)。

当然了这个正太分布 $q_\theta(z|x)$ 就用了重参数化技巧（用两个网络直接拟合 $\mu_\theta(z|x), \sigma_\theta(z|x)$, 然后采样 $\xi \sim \mathcal{N}(0,1)$ 计算 $z = \mu + \sigma \cdot \xi$)，详细结构见下图的 Encoder 部分。经过上文的推理，那么损失函数的设计天然就如图片中展示的那样。

具体推导就类似 EM 算法哪里，其中的 E 步得用变分推断。也就是广义

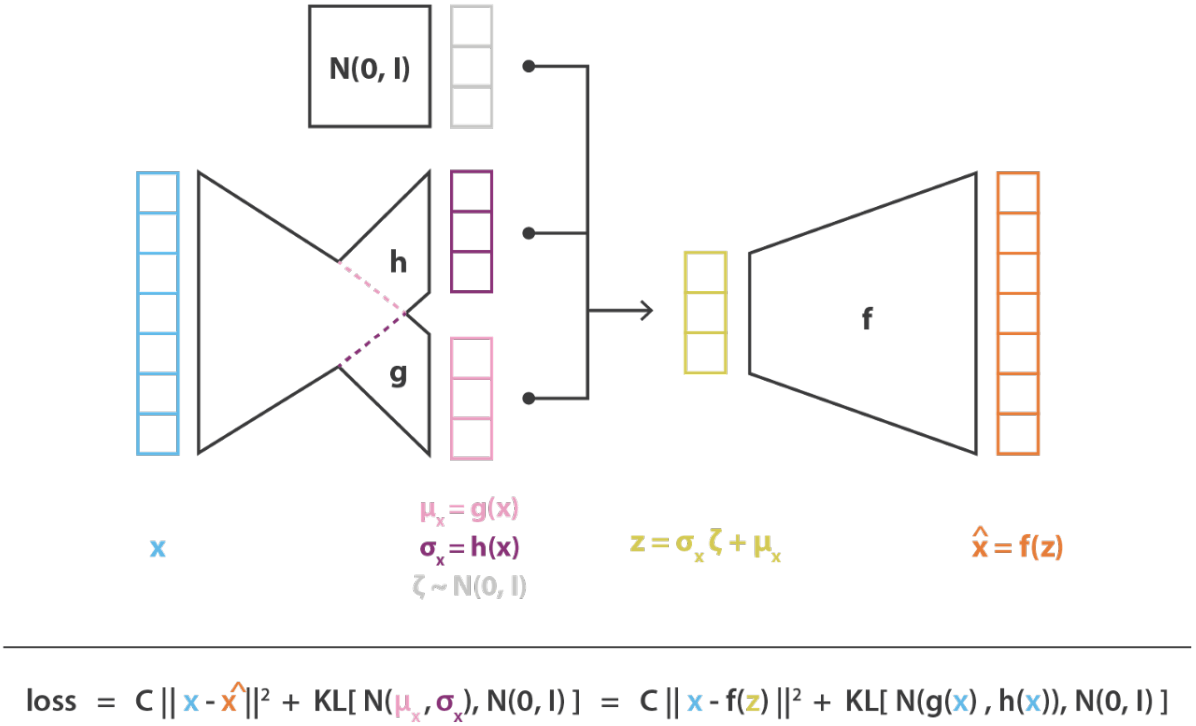

图 2.2: VAE 细节框图

EM 算法。不过不是坐标轮换式的求解，而是直接梯度下降。简而言之就是最大化 ELBO。所以目标函数及梯度更新和 SGVI 一样。

$$\begin{aligned}\log P(X) &= \log P(X,Z) - \log P(Z|X) \\ &= \log \frac{P(X,Z)}{q_\theta(z|x)} - \log \frac{P(Z|X)}{q_\theta(z|x)}\end{aligned}$$

两边对 $z \sim q_\theta(z|x)$ 求期望，可得

$$\log P(X) = ELBO + \mathcal{D}_{KL}[q_\theta(z|x)||P(Z|X)] \triangleq \mathcal{L} + \mathcal{D}_{KL}[q_\theta(z|x)||P(Z|X)]$$

$$\begin{aligned}\mathcal{L} = ELBO &= E_{q_\theta(z|x)}[\log \frac{P(x,z)}{q_\theta(z|x)}] \\ &= E_{q_\theta(z|x)}[\log P(x,z|\theta)] + H[q_\theta(z|x)] \\ &= E_{q_\theta(z|x)}[\log p_\phi(x|z) * q(z)] + H[q_\theta(z|x)]\end{aligned}$$

另一方面 (进一步化简)，

$$\begin{aligned}\mathcal{L} &= \int_z q_\theta(z|x) * \log(\frac{q(z)p_\phi(x|z)}{q_\theta(z|x)})dz \\ &= \int_z q_\theta(z|x) * \log(\frac{q(z)}{q_\theta(z|x)})dz + \int_z q_\theta(z|x) * \log(p_\phi(x|z))dz \\ &= -\mathcal{D}_{KL}[q_\theta(z|x)||q(z)] + \mathbb{E}_{q_\theta(z|x)}[\log p_\phi(x|z)]\end{aligned}$$

训练的目标就是：

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \mathcal{L} = \arg\max_{\theta,\phi} \left\{-\mathcal{D}_{KL}[q_\theta(z|x)||q(z)] + \mathbb{E}_{q_\theta(z|x)}[\log p_\phi(x|z)]\right\}$$

一项一项看，先看第一项的最小化 KL 散度。加上之前的假设 $q(z) \sim \mathcal{N}(0,1)$，近似推断用的近似函数 $p_\theta(z|x) \sim \mathcal{N}(\mu_\theta, \sigma_\theta)$，因此第一项的 KL 散度其实为：

$$\mathcal{D}_{KL}[q_\theta(z|x)||q(z)] = \mathcal{D}_{KL}[\mathcal{N}(\mu,\sigma^2)||\mathcal{N}(0,1)] = \frac{1}{2}(-\log\sigma^2 + \sigma^2 + \mu^2 - 1)$$

从另一种角度，这个 KL 散度项相当于一种正则. 让你近似推断的后验概率防止都收敛到一个点，也就是解码器对不同的 x 编码有的多样性要求. 请注

意 $q_\theta(z|x)$ 就是你的编码器。
第二项求期望，就利用重参数化技巧。请注意，因为网络的编码器就是 $q_\theta(z|x)$, 网络的解码器就是 $p_\phi(x|z)$, 因此它是先编码后解码的过程。

$$\mathbb{E}_{q_\theta(z|x)}[\log p_\phi(x|z)] = \mathbb{E}_{q_\theta(z|x)}[\log \frac{1}{\sqrt{2\pi}\sigma_\phi} e^{\frac{(x-\mu_\phi)^2}{2\sigma_\phi^2}}]$$

按照前面说的，我们的解码器直接输出期望也就是 $\mu_\phi = f_\phi(z|z = \xi * \sigma_\theta + \mu_\theta), \xi \sim \mathcal{N}(0,1)$, 并且为了更进一步简化结果，进一步假设 $\sigma_\phi \equiv c$, 为常数!
因此公式有

$$\begin{aligned}\mathbb{E}_{q_\theta(z|x)}[\log p_\phi(x|z)] &= \mathbb{E}_{q_\theta(z|x)}[\log \frac{1}{\sqrt{2\pi}\sigma_\phi} e^{\frac{(x-\mu_\phi)^2}{2\sigma_\phi^2}}] \\ &= \mathbb{E}_{\xi\sim\mathcal{N}(0,1)}[C_1 + c||x - \hat{x}||_2^2]\end{aligned}$$

因此通过各种假设及推导，就有了标准变分自编码的的目标函数:

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \left\{c||x - \hat{x}||_2^2 - \mathcal{D}_{KL}[q_\theta(z|x)||q(z)]\right\}$$

其中 c 为假定的似然分布的方差常量，在损失函数里面变成一种 trade-off 系数。整个损失函数体现着近似推断中近似的后验分布是先验信息和样本信息之间的权衡!
另外列举两个参考理解的链接:

半小时理解 VAE，其英文原版，李宏毅视频课的某博主笔记

正太分布的距离度量: 正太分布的各种距离

第三章　Monte Carlo Markov Chain Method

3.1　Markov Process

3.2　采样

3.2.1　Metropolis-Hashing 采样

3.2.2　Gibbs 采样

3.2.3　Langevin 采样

第四章　条件随机场模型的原理及应用

4.1　概率图模型及相近模型导言

条件随机场克服了 MEMM 模型因为隐变量采用有向图模型带来的局部归一化问题。**局部归一化**将导致 *label bias problem* 问题。label bias problem 问题的一种直接原因是所求的条件概率模型的熵比较小（训练得到的有向概率图模型局部概率转移 $\to 1$），导致对观测的 x 重视不够。而有向概率图形成的概率分布相对与无向图加了一些先验知识了，容易使得训练得到的条件概率 $\widehat{P}(X|Y)$ 熵下降（解释请查看手推机器学习 up 主的 CRF 的第三个视频）

具体地，条件随机场中隐变量的建模采用无向图模型。这里引入一条无向图概率模型中的结论：

Hammersley-Clifford 定理： 概率无向图的联合概率分布 $P(Y)$ 可以表述成如下形式

$$P(Y) = \frac{1}{Z}\prod_{C}\psi_C(Y_C)$$

$$Z = \sum_{Y}\prod_{C}\psi_C(Y_C)$$

Note:C 是无向图的最大 *clique*,Y_C 是 C 的节点对应的随机变量，$\psi_C(Y_C)$ 是 C 上定义的严格正函数，一般定义为指数函数如 $\psi_C(Y_C) = exp\{-E(Y_C)\}$

4.2　线型链条件随机场模型详解

线型链条件随机场广泛应用于**标注**问题。建模对象为条件概率 $P(Y|X)$,Y 表示输出变量或者状态序列或者就是要求的标记序列。简而言之，就是通过学习（极大似然估计方式如 EM 迭代、变分推理、深度学习）学出概率模型 $\widehat{P}(Y|X)$, 然后给定一个观测序列 x, 得到输出序列 $\widehat{y} = argmax_y \widehat{P}(y|x)$ 如 Vitebi 算法 **定义（线型链条件随机场）** 设 $X = (X_1, X_2, \cdots, X_n), Y =$

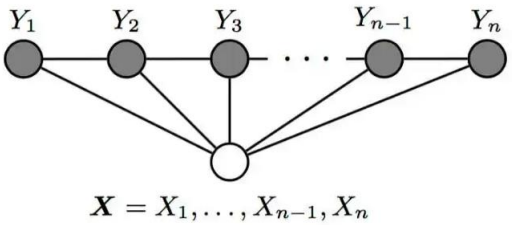

图 4.1: 线型链条件随机场模型

$(Y_1, Y_2, \cdots, Y_n)$ 为线性链表示的随机序列，给定 X 下，$P(Y|X)$ 构成条件随机场，即满足马尔科夫性：

$$P(Y_i|X, Y_1, Y_2, \cdots, Y_{i-1}, Y_{i+1}, \cdot, Y_n) = P(Y_i|X, Y_{i+1}, Y_{i-1})$$

$$\forall i = 1, 2, \cdots, n (i = 1, n\text{时只考虑单边})$$

对于线性链条件随机场，建模的对象是 $P(Y|X)$ 条件概率，由定义可知给定 X，$P(Y|X)$ 为无向图概率模型，则通过 Hammersley-Clifford 定理，以及此无向图概率模型最大集团为 2 的事实，将得到如下的建模：

$$
\begin{aligned}
P(Y|X) &= \frac{1}{Z}\prod_{C}\psi_C\left(Y_C\right) \\
&= \frac{1}{Z}\prod_{C} exp\{-f_C(Y_C)\}(\Rightarrow \text{最大团为 2}) \\
&= \frac{1}{Z} exp\{\sum_{i} f_C(y_{i-1}, y_i, x)\}(\Rightarrow \text{模型简化, 只用一个势函数, 并拆分成三部分}) \\
&= \frac{1}{Z} exp\{\sum_{i} f_1(y_{i-1}, y_i, x) + f_2(y_{i-1}, x) + f_3(y_i, x)\} \\
&= \frac{1}{Z} exp\{\sum_{i} f_1(y_{i-1}, y_i, x) + f_3(y_i, x)\}(\Rightarrow f_2\text{从重复建模和标注顺序角度舍去}) \\
&= \frac{1}{Z} exp\{\sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, x) + \sum_{i,l} \mu_l s_l(y_i, x)\}
\end{aligned}
$$

t_k, s_l 为我们专为任务定义好的特征方程，$\{\mu_l\}_l, \{\lambda_k\}_k$ 为待学习的参数 $\Rightarrow$ 为了更好的拟合 ψ 函数也就是我们的模型 $P(Y|X)$

向量化表达如下

$$
P(Y = y|X = x) = \frac{1}{Z(x, \lambda, \mu)} exp\{\lambda^T \sum_{t=1}^{T} f(y_{t-1}, y_t, x) + \mu^T \sum_{t=1}^{T} g(y_t, x)\}
$$

$$
x, y \in \mathbb{R}^T, T\text{为序列的长度；}\ f(y_{t-1}, y_t, x) = \begin{pmatrix} f_1 \\ \vdots \\ f_k \end{pmatrix} g(y_t, x) = \begin{pmatrix} g_1 \\ \vdots \\ g_l \end{pmatrix}
$$

再令 $\theta = \begin{pmatrix} \lambda \\ \mu \end{pmatrix}$,$H(x, y) = \begin{pmatrix} \sum_{t=1}^{T} f(y_{t-1}, y_t, x) \\ \sum_{t=1}^{T} g(y_t, x) \end{pmatrix} \triangleq \begin{pmatrix} F_1(x, y) \\ \vdots \\ F_{k+l}(x, y) \end{pmatrix} \in$ $\mathbb{R}^{k+l}$, 得最终模型为

$$
P(Y = y|X = x) = \frac{1}{Z(x, \lambda, \mu)} exp\left(\theta^T H(y, x)\right)
$$

4.3　条件随机场模型学习及推理

学习任务就是根据给定的数据 $\left\{x^{(i)}, y^{(i)}\right\}_{i=1}^{N}, x^{(i)}, y^{(i)} \in \mathbb{R}^T$ 得到模型，也就是得到模型的参数 θ:

$$\widehat{\theta} = \arg\min_{\theta} \prod_{i=1}^{n} P(y^{(i)}|\theta, x^{(i)})$$

这里就不详细介绍了，就是可行域为连续空间的最优化过程采用梯度法或者其他方法

推断 (*Inference*) 任务就是给定模型后根据模型以及输入得到一些其他的结果，最典型的有两种一种是给定模型 θ 得到边缘分布 $P(Y_t|\theta, X)$, 另一个最常见的也叫做解码过程可以称为最大后验(MAP)$y = \arg\max_{y} P(y|\theta, x)$

4.3.1　前向-后向算法求边缘分布

先做一些假定，假定状态序列 y 的每一个状态 $y_t = s \in \mathcal{S}$, 模型给定后为了简化表达公式采用势函数 $\psi(y_{t-1}, y_t, x)$ ，并给序列补上确定的头和尾即 $y_0 = start, y_{T+1} = stop$

更详细一点就是 $\psi = exp\left\{\sum_k \lambda_k * t_k + \sum_l \mu_l * s_l\right\}$，并且令 $\forall x$,$\psi(y_0 = start, y_1 \in \mathcal{S}|x) = 1$,$\psi(y_T + 1 = start, y_T \in \mathcal{S}|x) = 1$, 不影响任何概率和 $Z(X)$.

边缘分布就是的概率公式为 $f_X(x) = \int f(x, y)dy$ 因此所求的目标就是:

$$\begin{aligned} P(y_t = s|\theta, x) &= \frac{1}{Z} \int_{y_1,\cdots,y_{t-1},y_{t+1},\cdots,y_T} \left\{\prod_{i=1}^{T} \psi(y_{i-1}, y_i, x)\right\}_{y_t=s} dy_1, \cdots, dy_{t-1}, dy_{t+1}, \cdots, dy_T \\ &= \frac{1}{Z} \left(\int_{y_1,\cdots,y_{t-1}} \left(\prod_{i=1}^{t-1} \psi(y_{i-1}, y_i, x)\right) \psi(y_{t-1}, y_t = s, x) dy_1, \cdots, dy_{t-1} \right) * \\ &\quad \left(\int_{y_{t+1},\cdots,y_T} \psi(y_t = s, y_{t+1}, x) \prod_{i=t+2}^{T+1} \psi(y_{i-1}, y_i, x) dy_{t+1}, \cdots, dy_T \right) \end{aligned}$$

当然了上述公式其实就是累次积分，利用这个概率图的最大团为 2，并且从 Y_t 处分开的特点，当然 $\int \longleftrightarrow \sum$, 显然我们不能直接利用上述公式直接计算，计算量太大，还有很多重复计算，这种东西找一个递推式最方便。从上式可以看出我们的计算要分两个部分：$1 \longrightarrow t-1, t+1 \longrightarrow T$, 类比于 HMM 的前向后向算法定义 $\alpha_t(y_t = s|x), \beta_t(y_t = s|x)$ 分别代表上式的左右两部分即“从 1 到 t 的前部分标记且 t 时刻标记为状态 s 的序列的非规范化概率”和“t 时刻标记为 s 且从 i+1 到 T 的后半部分标记序列的非规范化概率”。因此很自然的（根据上述公式的累次积分计算或 CRF 的概率图模型）有如下的递推公式成立：

$$\begin{aligned}
\alpha_t(y_t = s|x) &= \int\limits_{y_1,\cdots,y_{t-1}} \left(\prod_{i=1}^{t-1} \psi(y_{i-1}, y_i, x)\right) \psi(y_{t-1}, y_t = s, x) dy_1, \cdots, dy_{t-1} \\
&= \int\limits_{y_{t-1}} \psi(y_{t-1}, y_t = s, x) \int\limits_{y_1,\cdots,y_{t-2}} \left(\prod_{i=1}^{t-2} \psi(y_{i-1}, y_i, x)\right) \psi(y_{t-1}, y_t = s, x) \\
&= \int\limits_{y_{t-1}} \psi(y_{t-1}, y_t = s, x) \alpha_{t-1}(y_{t-1}|x) dy_{t-1} \\
&= \sum_{y_{t-1} \in \mathcal{S}} (\alpha_{t-1}(y_{t-1}|x) * \psi(y_{t-1}, y_t = s, x))
\end{aligned}$$

同理可得

$$\beta(y_t = s|x) = \sum_{y_{t+1} \in \mathcal{S}} (\beta_{t+1}(y_{t+1}|x) * \psi(y_t = s, y_{t+1}, x))$$

前向后向算法的递推公式如上所示，最终所要求解的目标表达式如下

$$P(y_t = s|\theta, x) = \frac{1}{Z(x,\theta)} \alpha_t(y_t = s|x) * \beta_t(y_T = s|x)$$

其中 $Z(x,\theta) = \mathbf{1}^T \beta_1(\mathcal{S}|x) = \mathbf{1}^T \alpha_T(\mathcal{S}|x)$, 推导如下

考虑到 $P(Y = y|X = x) = \frac{1}{Z(x)} \prod\limits_{t=1}^{T} \psi(y_{t-1}, y_t, x)$ 因此对于归一化 Z 的求解如下：

$$
\begin{aligned}
Z(x) &= \sum_{y} \prod_{t=1}^{T} \psi(y_{t-1}, y_t, x) \\
&= \int\limits_{y_1,\cdots,y_T} \left(\prod_{i=1}^{T} \psi(y_{i-1}, y_i, x) \right) dy_1, \cdots, dy_T \\
&= \int\limits_{y_1,\cdots,y_T} \left(\prod_{i=1}^{T} \psi(y_{i-1}, y_i, x) \right) \psi(y_T, y_{T+1} = stop, x) dy_1, \cdots, dy_T \\
&= \int\limits_{y_1,\cdots,y_T} \psi(y_0 = start, y_1, x) \left(\prod_{i=1}^{T} \psi(y_{i-1}, y_i, x) \right) dy_1, \cdots, dy_T \\
&= \alpha_{T+1}(y_{T+1} = stop|x) = \sum_{s \in \mathcal{S}} \alpha_T(y_T = s|x) \\
&= \beta_0(y_0 = start|x) = \sum_{s \in \mathcal{S}} \beta_1(y_1 = s|x)
\end{aligned}
$$

4.3.2　条件随机场的预测任务

预测任务这就代指解码任务，在学习完模型参数 θ 和给定一个观测序列 $x \in \mathbb{R}^T$ 后找出 $y^* = \arg\max_y P(y|\theta, x) = \arg\max_y \theta^T H(y, x) = \arg\max_y \sum_{t=1}^{T} \theta^T f(y_{t-1}, y_t, x)$ 这里的 f 实际是 3.2 节中 f, g 的合并，是 $k+l$ 维的函数，之所以写成对序列位置 t 的求和的形式，就是想利用线型链条件随机场的结构利用 *Viterbi* 算法从位置角度动态规划得求出最大非归一化概率并回溯得到标签序列（状态序列）。

记 $\delta_t(s)$ 表示位置 t 下状态为 s 非规范化概率的最大值（因为其他 T-1 个位置是有随机性，计算出那个最大的费规范化概率），显然由动态规划求最大值的思想有下面的递推公式：

$$\delta_t(s) = \max_{s' \in \mathcal{S}} (\delta_{t-1}(s') + \theta^T f(y_{t-1} = s', y_t = s, x))$$

如果从“势”的含义解读就是当前时刻的最大势就等于上一个时刻各状态

的最大势加上对应状态转移的势的最大势；如果从概率角度就是当前时刻落到状态 s 的概率等于上一个时刻落到 s' 最大的概率和由 $s' \longrightarrow s$ 状态转移的概率之积的最大值。概率对应于规范化的势，并且还有个 exp, 所以非规范的和 $\longleftrightarrow$ 概率之积。

显然可以定义 $\Psi_t(s) = \arg\max_{s' \in \mathcal{S}}(\delta_{t-1}(s') + \theta^T f(y_{t-1} = s', y_t = s, x))$ 表示当前位置 t 对应状态 s 最大势应该由上一位置状态 $\Psi_t(s)$ 获得。即方便回溯。

当然了初始时 $\delta_1(s) = \theta^T f(y_0 = start, y_1 = s, x), \forall s \in \mathcal{S}$ 其本质是对 $\delta_0(s)$ 有要求，即只有 start 处的势为 1, 即对应特征函数值为 0，其他处的势为 0, 即对应特征函数值为 $-\infty$，这样显然都得从开始 start 处计算。

最终我们要求的最大概率值 $P^* = \max_{s \in \mathcal{S}} \delta_T(s)$, 并且最后时刻的序列状态就是 $y_T^* = \arg\max_{s \in \mathcal{S}} \delta_T(s)$, 接下来就是回溯得到 $y^* = (y_1^*, y_2^*, \cdots, y_T^*)$，回溯过程如下：

$$y_t^* = \Psi_{t+1}(y_{t+1}^*), i = T-1 \longrightarrow 1$$

Note: 为什么初始值不都设置为 *0* 或者都设置成一样的呢？就像上一节前向后向算法哪里序列前后分别加了 *start,stop* 并设置势函数值为 *1* 对邻近状态？

(1). 最重要的一点是任务不同。这里的任务是标签序列，你的特征函数是人工或者训练设计的，因此这个给定的特征函数要覆盖第一个词和最后一个词以更好的完成任务。例如以词性识别任务来讲第一个词的标签动词、名词、副词、代词等的概率是不一样的，正常语序来讲副词在第一个词中出现的概率肯定低。这个信息必然在这个特征函数里有体现也就是一种先验，所以你要用上这个有用的且废了九牛二虎之力人工设计或者训练出知识；

(2). 但就前一节的来讲，因为我要求的是积分求概率，所以其实给的最初始和最末尾的势函数值可以随便做，只要把对应的 Z 求好，把前向后向算法迭代初始的值算法就 *OK*。显然为何不放一个简单的 *1* 呢。

4.4　Bert-Bilstm-CRF 实体命名实战

其实实体命名识别可以看成一种分类任务,之所以不采用单纯的 $softmax$ 去分类主要是标签是有一定的上下文依赖性，比如采用 **BIO** 去标记（所谓 *BIO* 标记其实就是一些无关的词用 *O* 记，一些实体如“草莓”开始的“草”用 *B-fruit* 记, 其他的字用 *I-fruit* 记）显然 I-fruit 不会在 B-fruit 之前。CRF 可以捕捉局部的上下文，因此显然也可以用单向的 RNN（GRU，LSTM）去做。但一般要对词进行 *embedding* 操作，常采用一个 RNN 或者直接用 *Bert* 预训练模型再稍微 $fine-tune$ 一下，并且用深度学习去做 CRF，那些势函数或者更准确来讲那些特征函数（势函数可以理解成 exp 后的特征函数，之所以 exp, 是 exp 形式是最大熵模型的结果）就是用 Bi-lstm 学习出来的，所以最后的一层就一般用 CRF 了因为增加了模型多样性，而且你都用 LSTM 那做起来就是增加了层数而已，端到端的学习效果不佳，不同的任务其实用多种模型串联而不是都用同一个做到底有利于不同模型学出不同的子任务，整体效果也更加，并且实践也证明 CRF 的有效性。

4.4.1　基础模型 Bilstm-CRF 解析

原始的论文发表于 2016 年, 这个论文的创新点就是端到端的利用 CRF 去做命名实体识别。核心就是：通过词 *embedding* 成向量后经过 B *i-lstm* 层，*Bi-lstm* 的输出（每个位置的隐藏变量 h）经过一个全连接将隐藏变量映射到标签集 S 的维度上，这里 S 包含 *'start'* 和 *'stop'* 的标记，也就是形成了一个发射得分矩阵 P，同时随机生成一个状态得分矩阵 A，讲 P，A 看作 *CRF* 层的参数，其实更具体的说将 A 和产生 P 的 *Bi-lstm* 参数和词表示的 *embedding* 的参数为模型的参数，利用 *CRF* 学习过程（梯度下降）更新相关的参数。具体地，$P \in \mathbb{R}^{T*|S|}$ 这个矩阵的每一个元素就是前文讲特征函数的 $\lambda^T f_k(y_{t-1}, y_t, x)$；$A \in \mathbb{R}^{|S|*|S|}$ 的每一个值就是前文讲的特征函数 $\mu^T g_l(y_t, x)$。

根据前几节的基础，和上文的想法有 $s(X,y)=\sum_{t=0}^{T}A_{t,t+1}+\sum_{t=1}^{T}P_{t,y_t}$ 表示给定观测序列和标签序列 $x,y\in\mathbb{R}^T,T$ 表示句子长，的整体特征函数，因此对应的势函数 $\psi(y|x)=exp\{s(X=x,y)\}$。所以通过梯度下降算法，最小化负的似然函数，可获得最优的参数，然后通过 *Viterbi* 算法解码得到给定观测序列的最优标签即 $y^*=\arg\max_y P(y|\theta^*,x)$。当然模型最优参数 θ^* 的获得如下

$$\begin{aligned}\theta^* &= \arg\min_\theta -\log(P(y|X,\theta)\\ &= -\log(\frac{e^{s(X,y)}}{\sum_{\widetilde{y}\in Y_X}e^{s(X,\widetilde{y})}})\\ &= \log(\sum_{\widetilde{y}\in Y_X}e^{s(X,\widetilde{y})})-s(X,y)\end{aligned}$$

因此,训练过程就是求出后面两项,最后一项好求就是一个循环见代码"_score_sentence(self, feats, tags)" 这个函数，第一项就是用到前面前向后向算法求 Z 实际上为 $(\log(Z))$。不过代码中的 α 和上节中的 α 不同，代码中的其实是特征函数也就是多了一个 log 操作，那我就用 β 代替上一节的 α, 保持代码中的 α 表述。所求第一项就是:

$$\begin{aligned}\log(Z)&=\log(\beta_{T+1}(y_{T+1}=stop))=\log(\sum_{s\in\mathcal{S}}\beta_T(y_T=s|x)*\psi(y_T=s,y_{T+1}=stop,x))\\ &=\log(\sum_{s\in\mathcal{S}}exp\{\alpha_T(y_T=s|x)\}*exp\{trans(y_T=s,y_{T+1}=stop,x)*1\})\\ &=LSE\{\alpha_T(y_T=s|x)+trans(y_T=s,y_{T+1}=stop,x)+0\}\end{aligned}$$

最后的 +0 其实是 *emiss* 部分对应于矩阵 A，由于是 $T+1$ 时刻没有了所以只有 *trans* 对应的矩阵 P 部分。LSE(Log-Sum-Exp) 中的一项 $\alpha_T(y_T=s|x)$,

递推公式有

$$\begin{aligned}\alpha_{t+1}(y_{t+1}=s|x) &= \log(\beta_{t+1}(y_{t+1}=s|x)) \\ &= \log(\sum_{s'\in\mathcal{S}} \beta_t(y_t=s'|x) * \psi(y_t=s', y_{t+1}=s, x)) \\ &= LSE\{\alpha_t(y_t=s'|x) + trans(y_t=s', y_{t+1}=s, x) + emiss(y_{t+1}=s, x)\}\end{aligned}$$

初始时刻有

$$\alpha_0(y_0=s|x) = \begin{cases} 0 & s=start \\ -\infty & otherwise \end{cases}$$

这里程序 $-\infty$ 用的-10000。因为 α 就是 $\log(\psi)$ 后的势函数，在上一节的前向后向推导中加了 $\psi(y_0=start, y_1\in\mathcal{S}|x)=1$ 不影响计算且对标准任务 make sense, 所以如上设置，start 对应的势函数为 1 也就是 α 为 0，其他状态势函数为 0，也就是 $\alpha=-\infty$ 基础版整体算法参见pytorch_ 官方 Demo 其中前向算法代码：

```
def log_sum_exp(vec):
 max_score = vec[0, argmax(vec)]
 max_score_broadcast = max_score.view(1, -1).expand(1, vec.size()[1])
 return max_score + \
     torch.log(torch.sum(torch.exp(vec - max_score_broadcast)))

def _forward_alg(self, feats):
     #前向算法求Z
     # Do the forward algorithm to compute the partition function
     init_alphas = torch.full((1, self.tagset_size), -10000.)
     # START_TAG has all of the score.
     # init_alphas[0][self.tag_to_ix[START_TAG]] = 0.
     # 势函数开始-》其他为1，因为这里势函数为exp（score），
     # 由于exp 包含在LSE里面 因此现在为0，其他地方为-无穷
```

```
15
16          # Wrap in a variable so that we will get automatic backprop
17          forward_var = init_alphas
18
19          # Iterate through the sentence
20          for feat in feats:
21              alphas_t = []  # The forward tensors at this timestep
22              for next_tag in range(self.tagset_size):
23                  # broadcast the emission score: it is the same
24                  # regardless of the previous tag
25                  emit_score = feat[next_tag].view(
26                      1, -1).expand(1, self.tagset_size)
27                  # the ith entry of trans_score is the score
28                  # of transitioning to
29                  # next_tag from i
30                  trans_score = self.transitions[next_tag].view(1, -1)
31                  # The ith entry of next_tag_var is the value for the
32                  # edge (i -> next_tag) before we do log-sum-exp
33                  next_tag_var = forward_var + trans_score + emit_score
34                  # The forward variable for this tag is log-sum-exp of
35                  # all thescores.
36                  alphas_t.append(log_sum_exp(next_tag_var).view(1))
37              forward_var = torch.cat(alphas_t).view(1, -1)
38          terminal_var = forward_var + self.transitions[\
39                         self.tag_to_ix[STOP_TAG]]
40          alpha = log_sum_exp(terminal_var)
41          return alpha
```

上面用来两层循环，完全可以利用矩阵（tensor）讲第二层循环去掉。

$$\alpha_{t+1}(y_{t+1}=s_1)=LSE(\begin{pmatrix}\alpha_t(y_t=s_1)\\ \vdots \\ \alpha_t(y_t=s_{|\mathcal{S}|})\end{pmatrix}+\begin{pmatrix}trans_t(y_{t+1}=s_1,y_t=s1)\\ \vdots \\ trans_t(y_{t+1}=s_1,y_t=s_{|\mathcal{S}|})\end{pmatrix}+\begin{pmatrix}emiss_t(y_{t+1}=s_1)\\ \vdots \\ emiss_t(y_{t+1}=s_1)\end{pmatrix})$$

仅一项可以看出，等式右端第一项为公共项（对每个状态来讲），最后的发射得分都一样每个元素都相同，因此要想获得行向量 $(\alpha_{t+1}(y_{t+1}=s_1),\cdots,\alpha_{t+1}(y_{t+1}=s_{|\mathcal{S}|}))$，只需将第一项沿着列维度复制标记集合大小 $|\mathcal{S}|$ 次，然后第二项其实就是状态状态转移矩阵的转置 A^T, 第三项其实就是求的发射特征在时间或者说位置 $T+1$ 下的行向量按照沿着行的方向复制 $|\mathcal{S}|$ 次。实现见参考代码:

```
1    def _forward_alg2(self, feats):
2        #前向算法求Z
3        init_alphas = torch.full((1, self.tagset_size), -10000.)
4        # START_TAG has all of the score.
5        init_alphas[0][self.tag_to_ix[START_TAG]] = 0.
6        # Wrap in a variable so that we will get automatic backprop
7        forward_var = init_alphas #1*|S|
8        # Iterate through the sentence
9        for feat in feats:
10
11           forward_var = torch.logsumexp(
12               forward_var.view(-1,1).expand(self.tagset_size,\
13               self.tagset_size)\
14                       +self.transitions.permute(1,0)+
```

```
                feat.expand(self.tagset_size, self.tagset_size), dim=0
            ).view(1,-1)
        terminal_var = forward_var + \
                       self.transitions[self.tag_to_ix[STOP_TAG]]
        alpha = torch.logsumexp(terminal_var.view(1,-1),dim=1)
        return alpha
```

4.4.2 Bert-Bilstm-CRF

参考代码：Bert-Bilstm-CRF 实例 其中前向算法代码：

```
def log_sum_exp_batch(log_Tensor, axis=-1): # shape (batch_size,n,m)
    return torch.max(log_Tensor, axis)[0] + \
        torch.log(torch.exp(log_Tensor-torch.max(log_Tensor, \
        axis)[0].view(log_Tensor.shape[0],-1,1)).sum(axis))

def _forward_alg(self, feats):
    '''
    this also called alpha-recursion or forward recursion,
    to calculate log_prob of all barX
    '''

    # T = self.max_seq_length
    T = feats.shape[1]
    batch_size = feats.shape[0]

    # alpha_recursion,forward, alpha(zt)=p(zt,bar_x_1:t)
    log_alpha = torch.Tensor(batch_size, 1,\
```

```
            self . tagset_size ). fill_ (-10000.).to( self . device )
        #[batch_size, 1, 16]
        # normal_alpha_0 : alpha[0]=Ot[0]* self . PIs
        # self . start_label has all of the score . it is log ,0 is p=1
        log_alpha [:, 0, self . start_label_id ] = 0

        # feats: sentances -> word embedding -> lstm -> MLP -> feats
        # feats is the probability of emission , feat . shape=(1,tag_size)
        for t in range(1, T):
            log_alpha = (log_sum_exp_batch(self. transitions + log_alpha,\
              axis=-1) + feats[:, t ]). unsqueeze(1)
            #lse后没了一维，所以补上

        # log_prob of all barX
        log_prob_all_barX = log_sum_exp_batch(log_alpha)
        return log_prob_all_barX
```

第五章　Diffusion Model

虽然很想在变分推断那一章介绍完 Diffusion Model, 但它关联太广, 值得单拿出来。

5.1　Score Based Model

5.2　DDPM 推导

5.3　SDE 视角下的 Diffusion Model

5.4　ODE 视角下的 Diffusion Model

5.5　Flow Matching

5.5.1　Flow based Model

第六章　Kalman Filter