自编码器与变分自编码器

~~界外章节: 章节 9:自编码器与变分自编码器~~
矩阵分解系列教程迎来了尾声

引入

我们之前一直在做矩阵分解,
矩阵分解的内核是, 把高维的信息整合, 通过一些正则化手段降维, 以此提取主要特征, 我们的 qr 分解提取了正交特征, 特征值 (奇异值) 分解提取了各维度主特征, 都是一个降维的过程, 这类分解统称为因子分解
上一章的非负矩阵分解通过将矩阵分解为两个非负矩阵的乘积，能有效揭示数据的潜在特征和结构, 这里我们会有更有效的方法

现在让我们来看看比较新的降维(升维)手段

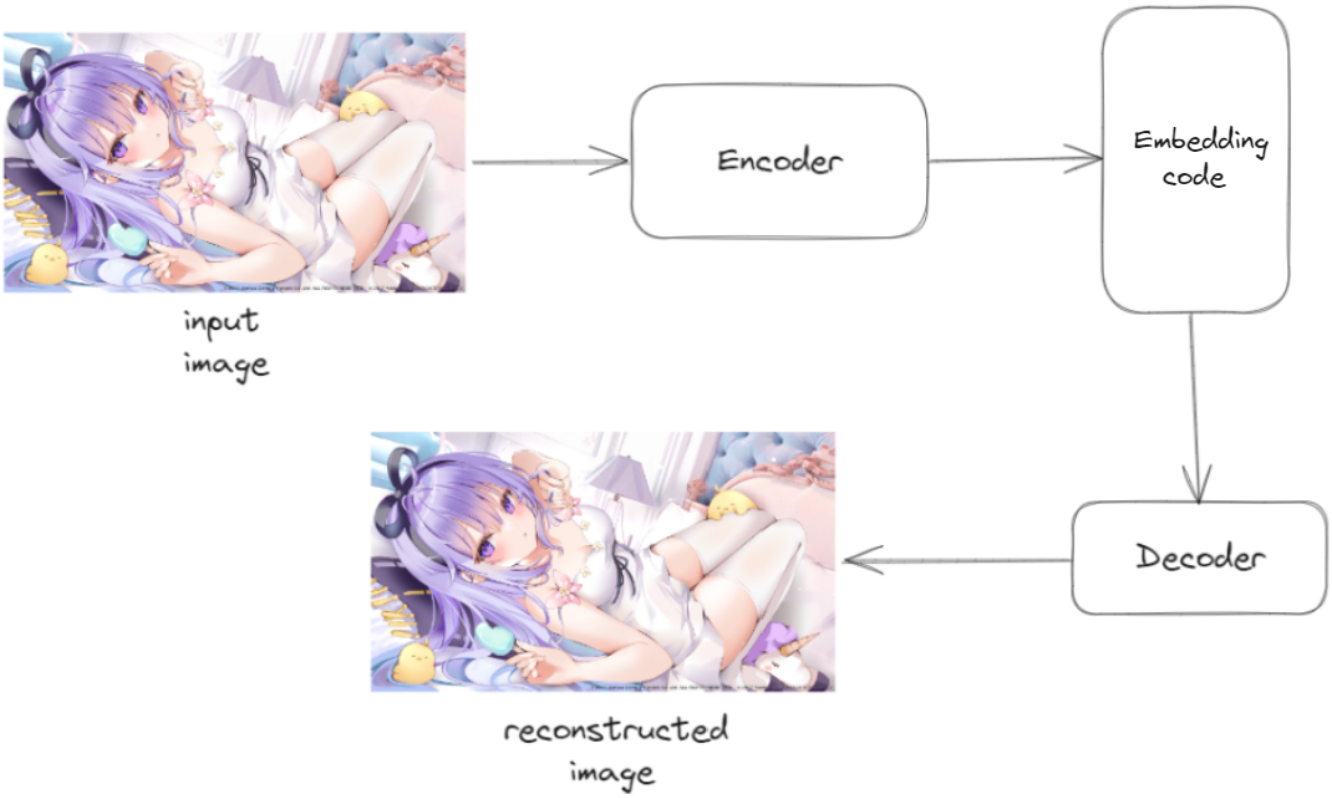
要讲变分自编码器, 我们首先要讲自编码器才行
自编码器 ($Autoencoder$, AE (~~Applied energy~~)) 是一种神经网络, 用于学习数据的表示方式(编码), 是一种~~无监督~~学习架构
他的内核和矩阵分解是一样的, 只是失去了有效的模式化的表示方法 (没有公式可以表达了)
就像大自然的很多东西, 我们没有办法用公式完全地解析
尽管如此, 我们仍然是有办法知道编码后的内容大概代表了什么信息, 不过这里不作赘述, 可以自己看论文

- 论文: 1904.05742 1804.02812 1905.05879
  它通过一种更灵活的方式提取矩阵的潜在规律 (这是不规则的), 并且可以以不同的维度转换信息 (而对输入输出没有要求)
  PCA 是一种线性降维, 而这是一种非线性降维,可以捕捉复杂的非线性相关性
  因此, 它用于降维, 特征提取, 数据重构

首先, 我们会介绍自编码器的架构, 然后是实现方法, 各种相关算法, 以及有意思的变分自编码器
最后, 我会放上我的 $colab$ 的代码, 方便大家运行

自编码器架构

学习神经网络, 首先要清楚这个网络的架构, 然后再考虑具体的求解方法
神经网络由**编码器** ($Encoder$) 和**解码器** ($Decoder$) 组成
它的平凡的实现是: 把输入复制到输出
例如，给定一个手写数字的图像，自编码器首先将图像编码为低维的潜在表示，然后将该潜在表示解码回图像
自编码器学习压缩数据，同时最大程度地减少重构误差



很像 $Bert$, 你可以把 $Bert$ 的前 6 层看作 $Encoder$, 后 6 层看作 $Decoder$ (如果是 12 层那个的话)
尽管 $Bert$ 比这玩意晚了十几年才出来 ()

$Decoder$ 可以看作一个 $Generator$, 吃一个向量, 产生一个东西

编码器

编码器把输入的数据转换为一个更紧凑, 更低维的表示, 其实就是数据的降维
它把数据映射到一个隐含的特征空间, 是一个向量 (大多数情况下, 也可以不是)
例如, 它可能会把矩阵 $\begin{pmatrix} 6.12 & 183 \\ 777 & 555 \end{pmatrix}$ 映射到 $(9999 \quad 0 \quad 132 \quad 123)$, 这当然是我随便想的, 但这也是有可能的,因为编码器内部数据就是一个向量, 里面的每一个具体的位置, 都隐含了很多意义(以及和其他元素的联系), 而不是传统意义上的位置信息
这个例子想说明的是, 自编码器不是通过某些数学规律来提取特征的, 而是通过在大量矩阵数据输

入的学习中, 学到了内部的潜在规律, 就像你无法说明你是由什么数学原理而学会了说话一样, 唯一的解释是: 大量的经验, 在经验的积累中, 在某一刻产生了质变

中间的向量 ($code$) 包含了很多信息, 如果是图片, 就包含了图片的主要信息, 亮度之类的
如果是声音, 就包含了声音里几个主要的波什么的
如果是文章, 就包含了文章的主要内容之类的
对于声音,我们可以通过做 $Feature\ Disentangle$, 来知道哪些维度代表了说话人的特征, 哪些部分代表了话语的内容, 其他也是类似的

编码过程可以表示为:

$$Embedding = h(x)$$

解码器

解码器把内部特征空间混乱不堪的数据重新解释, 它学习如何重构输入, 并且尽可能准确地复原数据, 可以输出和输入相同的格式的内容, 也可以输出不同格式的内容
就像人的记忆一样, 它解码的过程就是在对记忆进行回忆

解码过程可以表示为:

$$r = f(Embedding) = f(h(x))$$

其中, 两个函数 f 和 h 其实都是一个 nn (神经网络), 使用矩阵运算:

$$y = Wx + b$$

其中 W 为权矩阵, b 称为 $bias$ (偏置)

网络训练

自编码器网络的训练机理为: 最小化重构误差
这个过程使得自编码器能够捕捉并学习输入数据的重要特征，同时去除不必要的噪声或冗余信息
如果输入矩阵 A, 输出矩阵 B, 那么训练过程就是一个优化模型, 优化目标为:

$$\min_{\text{网络参数}}(||A - B||_{loss})$$

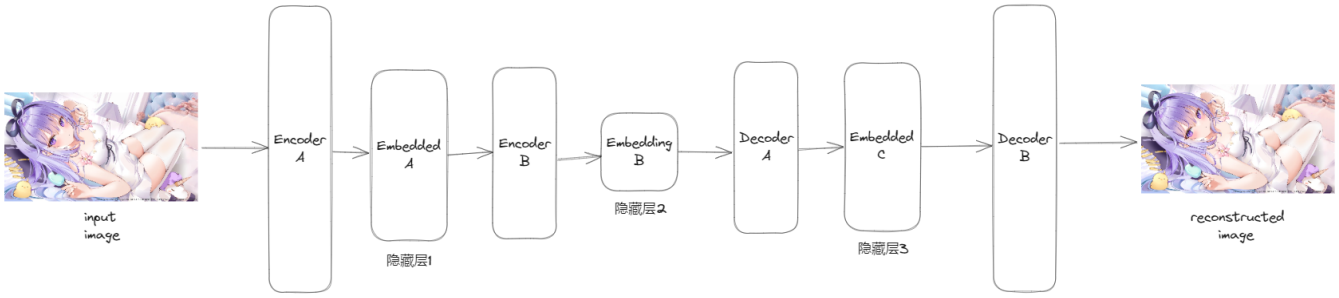
当然这个误差表示有很多种,一般是均方误差(MSE), 也可以用平方和, 也可以用别的

我们通过对比输入和输出 , 求出预测误差，进行反向传递, 更新网络参数，逐步提升自编码的准确性

求解这个优化模型一般用梯度下降家族的方法 ($Adam$ 或者 $RMSprop$)
为了避免过拟合, 应该使用 $Early\ Stopping$ 或者正则化 ($L1$ 或 $L2$)

堆栈自编码器

有些时候我们需要多层自编码器 (如后面的分类任务), 这个时候我们的网络就应该深一点
如图:

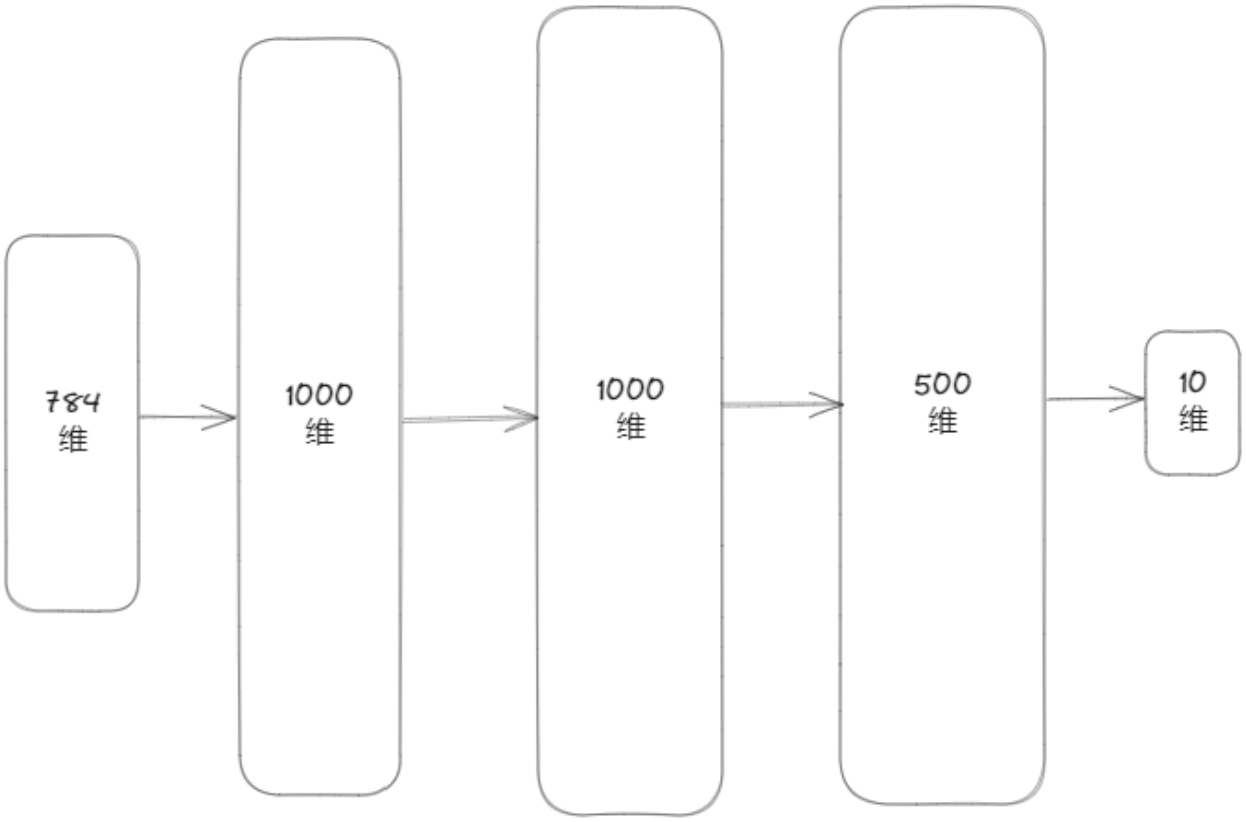


这张图片是有三个隐藏层的堆栈自编码器, 层数多了就变成深度自编码器
注意, 图片里的方块大小是有意义的, 这意味着我们的两个编码过程在不断降低特征的维度
而我们的解码过程在不断提升特征的维度

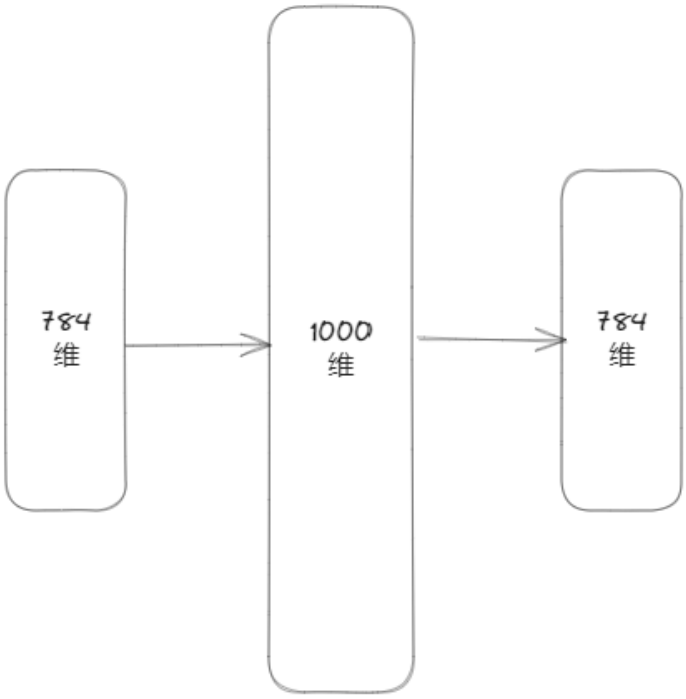
应用

预训练

很久之前由于深度网络是随机初始化的, 所以训练很慢效果也不好, 就没有引起重视
直到开始使用一些手段来初始化比较好的各个网络层级
假如有一个这样的神经网络, 它的网络架构如下:



我们怎么预训练第一层的网络呢? 使用一个自编码器:



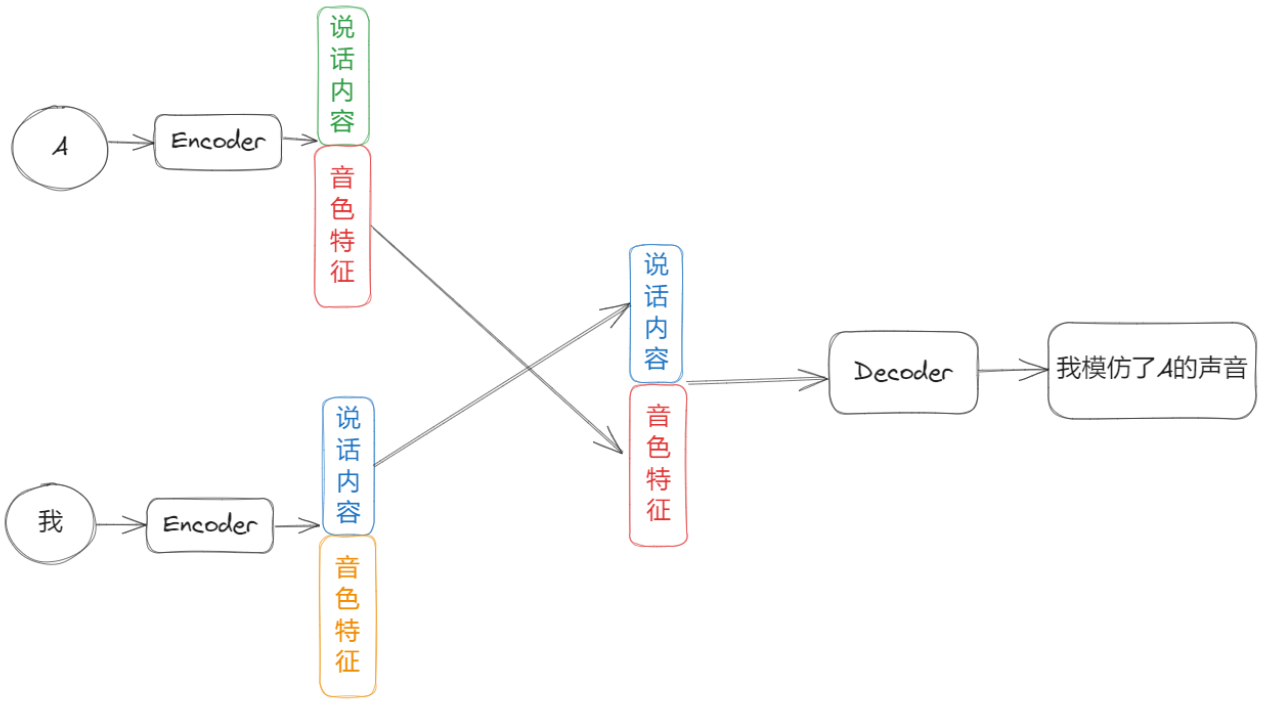
把这个训练到收敛, 我们就取这个自编码器的 $Encoder$ 作为初始参数

压缩

之前说到我们通过 $Encoder$ 得到的东西是一个低维的向量, 这其实就可以看作一种压缩, 而 $Decoder$ 做的事情就是解压缩
但是这个压缩是 $Lucy$ 的, 也就是说会失真

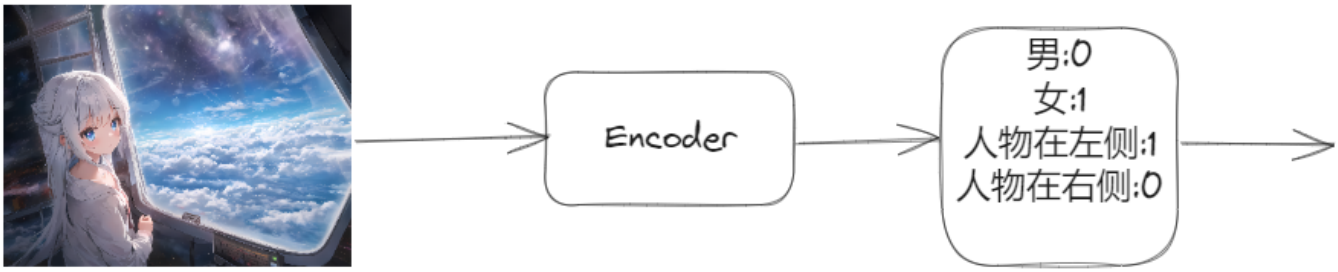
VITS

VITS 就是这个架构的, 不知道是啥? 查一下就知道了
过去我们要模仿别人的声音, 需要做监督学习, 就是需要有标注的数据
比如我想模仿 A 的声音, 我需要的数据有什么呢?
我说一句"早上好", 必须让 A 也来说一句"早上好"
并且这样的成对数据需要很多, 但这是不现实的, 如果 A 不会中文呢? 这就做不到了
我们这个时候就可以用自编码器, 如果我们做了 $Feature\ Disentangle$, 就能把 $code$ 分成下面那样:



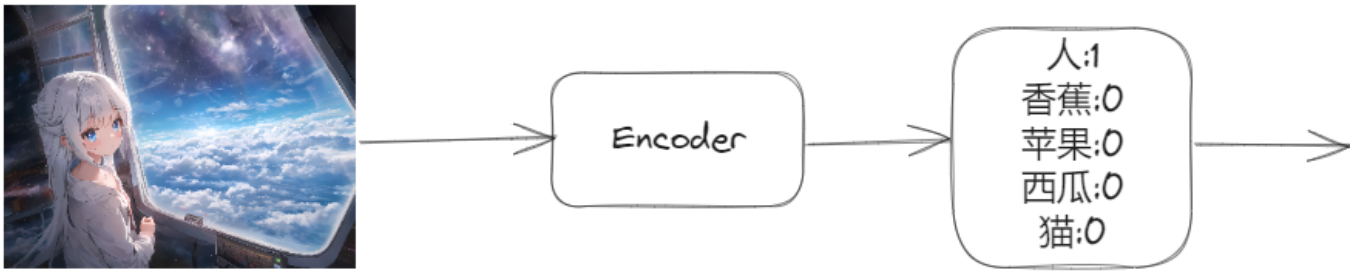
分类器

之前我们说到 $Embedding$ 或者是 $code$ 是一个向量, 如果换成别的呢?
如果我们换成一个 $Binary$, $Binary$ 是一个逻辑数组, 里面的内容都是 $bool$ 型的, 只能取 0 或 1
类似于决策树, 我们可以根据很多特征来分类:



这样就根据一个特征分类了, 把很多个自编码器类似于决策树一样堆起来, 就能做分类了

如果我们换成一个 $One-hot$, $One-hot$ 是一个逻辑数组, 它只有一个位为 1, 其他位都为 0

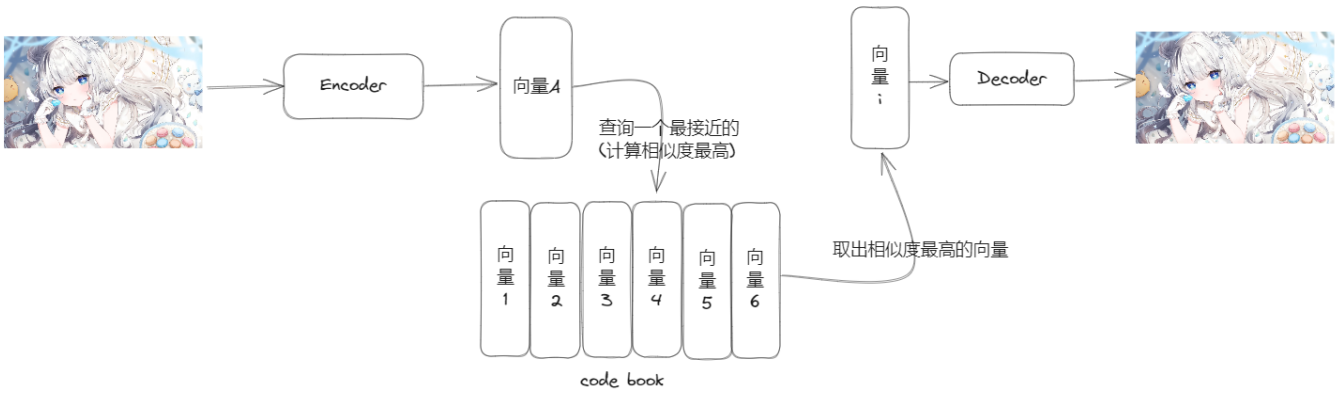


通过这个我们可以实现 $UNsupervise$ 的分类, 也就是无标签分类

在分类时, 损失函数应该使用交叉熵损失

VQVAE

$Vector\ Quantized\ Variational\ Auto-encoder$ --基于离散隐变量的生成模型
这个才是最出名的
其实就是在我们的网络架构中添加了一个 $code\ book$ 具体操作如图:



在把一个图片用 $Encoder$ 处理之后, 我们不直接把结果送到 $Decoder$, 再计算误差

而是从准备的 $code\ book$ 中找到相似度最高的向量, 然后把那个向量送到 $Decoder$, 再计算误差
通过最小化误差, 我们会更新我们的 $Encoder$, $Decoder$, 还有 $code\ book$

这样的好处是, 我们的 $code\ book$ 有机会学到本质的东西
如果是图片, 训练后我们的 $code\ book$ 里可能是图片的类型, 风景图人物图啊之类的
如果是声音, 我们的 $code\ book$ 里可能是最基础的发音之类的

- $Attention\ Is\ All\ You\ Need$

文字

我们甚至可以用一串文字代替 $Embedding$, 这段文字, 很有可能就是这篇文章的摘要
但是这里的 $Encoder$ 和 $Decoder$ 就需要是 $Sequence - To - Sequence$, 因为我们输入文章, 输出文字, 都是一串一串的
这样的 $Auto - encoder$, 就是 $Sequence - To - Sequence - To - Sequence$ 的

这样做的好处是, 我们不需要标注就能让机器自己学会做摘要

你信了吗? 然而实践告诉我们这样行不通
在训练的过程中, $Encoder$ 和 $Decoder$ 会发明自己的暗号!
在提取摘要后, 里面会出现一段 (也许是分散的) 暗号, 这段暗号是你看不懂的, 但是 $Decoder$ 可以通过这段乱七八糟的暗号知道这段文章是啥, 它们就这样完成了训练! 而结果是这个模型会提取出一段乱七八糟的暗号而不是摘要

我们需要一个额外的模型称为 $Discriminator$, 它用人写的文本训练, 作用是判断一段文字能不能构成语言
我们需要使 $Encoder$ 不仅可以编码成一个东西后能扔给 $Decoder$ 还原, 还必须把这个编码后的东西扔给 $Discriminator$ 校验, 必须让 $Discriminatot$ 认为这构成语言, 我们希望这样能强迫 $Encoder$ 可以提取大纲

懂一点点机器学习的朋友可能会说, 这中间产生文本还要扔给 $Discriminator$ 这做起来好麻烦啊, 这网络怎么搭呢? 我的评价是, 没办法 $train$ 的问题就用 rl 硬做就行了

- 在我看来, 这根本就是 $Cycle\ GAN$

TREE

还有一个拿 $TREE$ 当 $Embedding$ 的案例, 这太亏贼了, 架构颠佬恐怖如斯
就是一段文字变成 $TREE$, 再拿这个 $TREE$ 还原文字

- 论文: 1904.03746

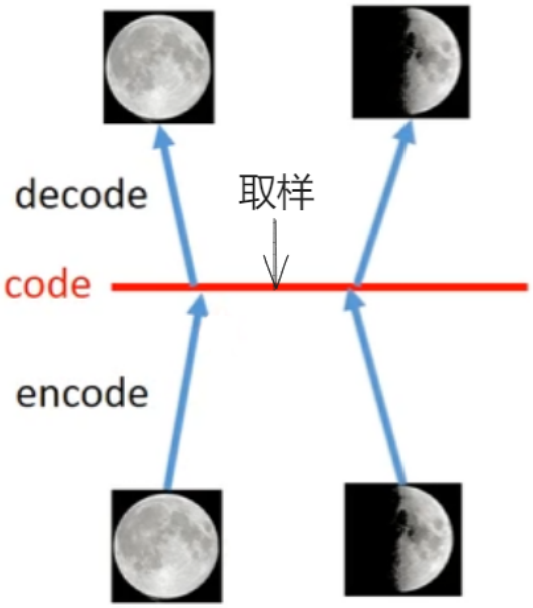
变分自编码器 VAE

原文: [1312.6114] Auto-Encoding Variational Bayes (arxiv.org)
除了 GAN 以外, 还有 VAE 也是生成模型
$VAE - Variational\ Auto - encoder$, 看名字就知道它和 $Auto - encoder$ 很有关系

它其实就是把 $Auto - encoder$ 的 $Decoder$ 拿出来当成 $Generator$, 但是也做了其他事情
下面细说

引入

我们知道, 我们的 $Decoder$ 可以看作一个生成模型, 根据一个向量, 生成一个东西
考虑以下情形:

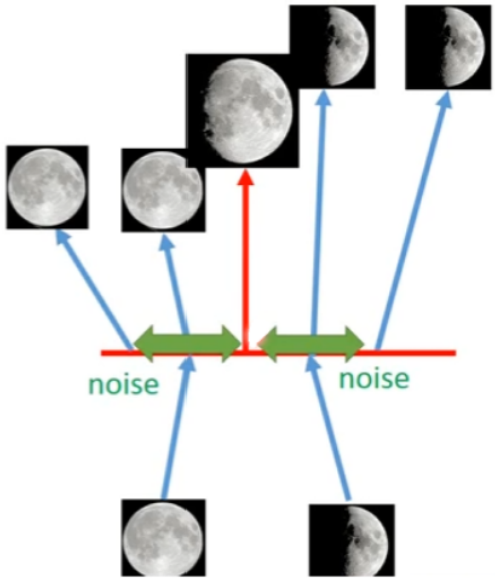


我们一系列图片, 是月亮的不同形态, 现在我们假设我们把月亮的图片编码后的 $Embedding$ 是一个一维的向量 (也就是一个数, 一条直线)

好, 现在我们把月圆的地方的 $code$ 放进 $Decoder$, 就会返回一张月圆的照片
同样地, 把半月的地方的 $code$ 放进 $Decoder$, 就会返回一张半月的照片
那么问题来了, 如果我们把这两个点中间的点放进 $Decoder$ 会怎么样呢?
我们当然是期望返回一张介于月圆和半月之间的月亮的图, 但是事实上
在绝大部分情况下, 这样生成的都是一些没有意义的照片, 运气好的话, 我们可能可以获得一些有意义的照片

为什么会这样呢? 这是因为我们没有对隐变量 (也就是 $Embedding$ 或者 $code$) z 的分布进行估计
直线空间 $\mathbb{R}$ 已经很大了, 我们不知道有意义 (可以通过 $Decoder$ 转化为有意义的图) 的 z 到底在这段直线上的哪个地方, 分布在哪个区间
由于取值有无穷多个, 而有意义的只有有限多个, 我们还不知道有意义的分布在哪, 这样找有意义的 z 就像大海捞针, 所以我们需要对隐变量 z 进行建模, 要知道在哪些区间下可以生成哪些(类别的)图片

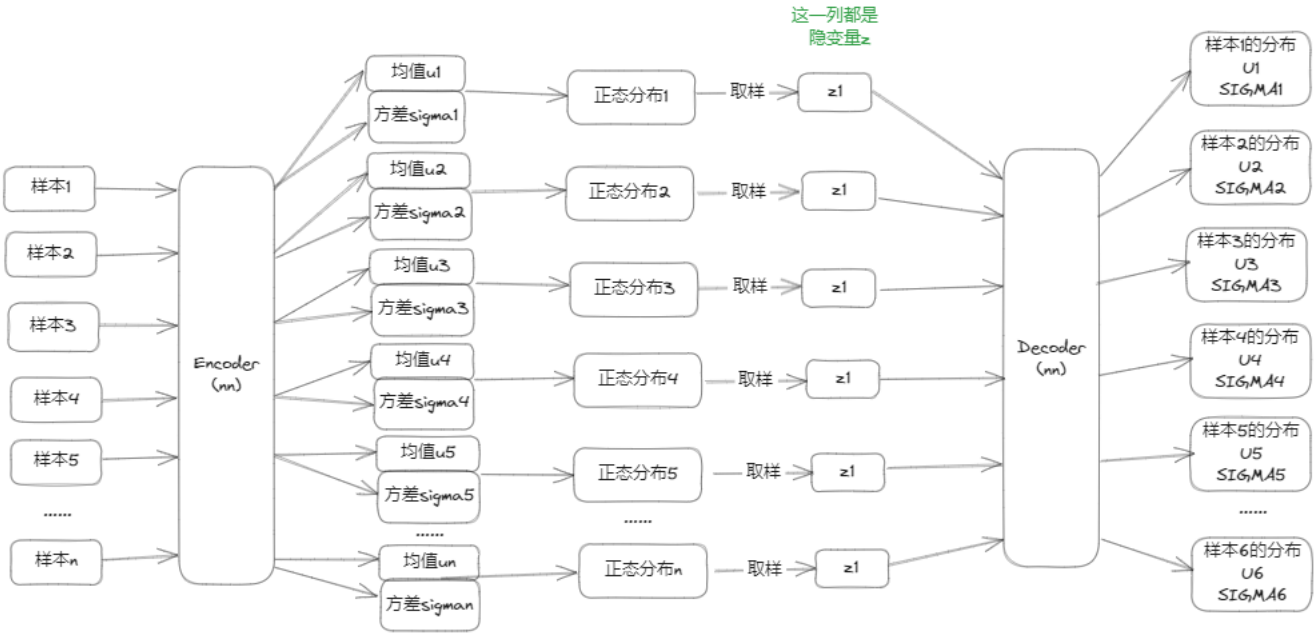
也就是说, 我们的模型应该能够适应一定范围的 $noise$, 即就算我们在这两个点中间取样, 模型的结果也不会偏离太多 (至少也应该是一张月亮)



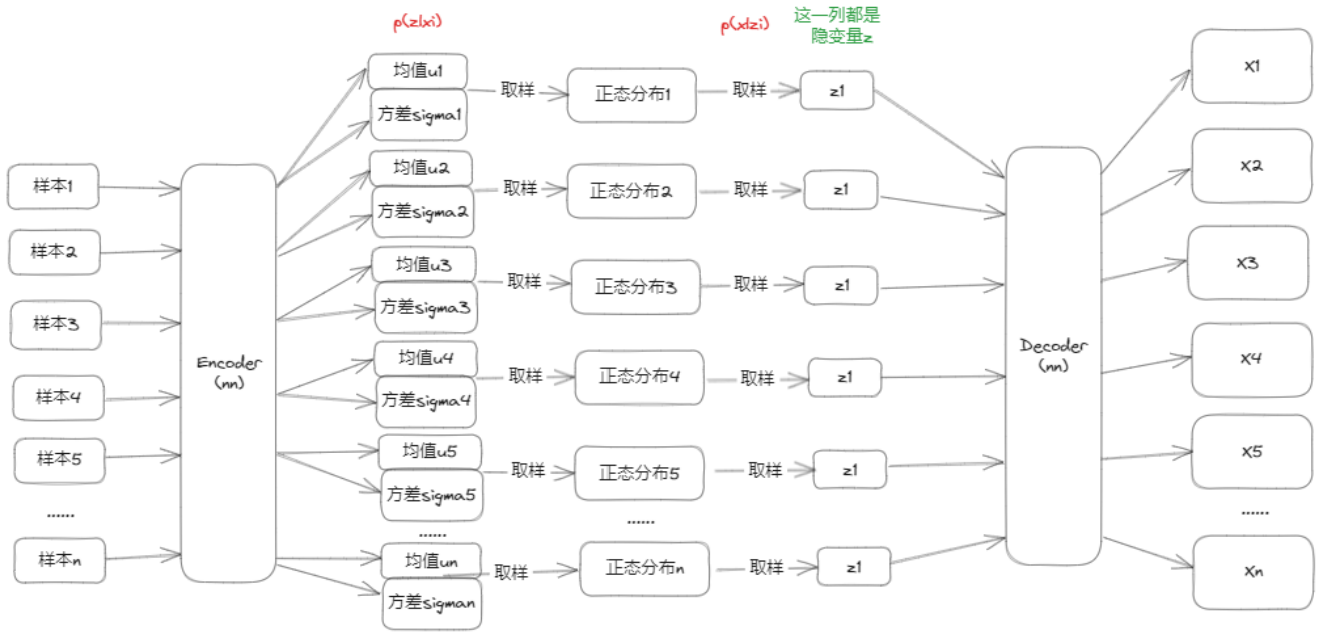
这就引出了变分自编码器的设计

架构

变分自编码器把给每一个样本使用 $Encoder$ 学习一个分布, 再在学习的分布上随机采样, 将采样结果送入 $Decoder$, $Decoder$ 会尽量还原样本原始的分布
基本的架构如下:



但是通常我们不需要求到样本的分布, 而是生成一个新的样本就够了, 于是网络变成这样:



这就是我们主要考虑的架构

这个结构就和自编码器差不多了, 只是中间多了一步根据每个样本拟合正态分布, 再随机取样的过程

1. 向 $Encoder$ 输入一个数据点 x_i, 通过 $Encoder$ (nn) 我们得到 x_i 对应的隐变量 z 服从的后验分布 $p(z|x_i)$ 的参数 u_i 和 σ_i^2
2. 根据 u_i 和 σ_i^2 我们可以生成对应的正态分布, 从这个正态分布中采样得到一个 z_i, 这个 z_i 就应该是和 x_i 强相关的
3. 向 $Decoder$ 输入一个隐变量点 z_i, 通过 $Decoder$ (nn) 我们得到 z_i 对应的生成样本点 X 服从的条件分布 $p(X|z_i)$ 的参数 u_i' 和 $\sigma_i'^2$

4. 我们可以从这个分布 $p(X|z_i)$ 中采样, 也可以直接拿均值 u_i' 作为生成的数据

符号参照表:

X : 样本随机变量
x_i : 样本随机变量的一个取值
z : 隐变量随机变量
z_i : 隐变量随机变量的一个取值
$p(z)$: 随机变量z符合的先验分布
$p(X|z_i)$: 已知z_i的情况下, X的条件分布
$p(z|x_i)$: 已知样本x_i下, z的后验分布
$p(x)$: 样本分布的近似
q : 用来区分, 其实$q = p(z|x_i)$

推导

假设: z 的分布

- 思想: 由于 z 的分布是很不确定的, 我们只能在 $\mathbb{R}^n$ 上随机取样碰运气, 为什么我们不先假设 z 符合一个简单的, 取值很有限的分布呢? 这样就能把采样的空间压缩的足够小
- 正态分布恰恰有这样的性质, 它在两侧的取样概率是很小的, 也就意味着值大多在中间

于是我们要求z 符合分布 $z \sim \mathcal{N}(0, I)$, 其中 I 为单位阵, 意味着 z 符合多元正态分布, z 是一个随机变量:

(一定是正态分布吗? 其实不然, 只是正态分布既方便计算, 又便于推导, 它可以是任何东西)
好的, 我们现在引入了随机变量的概念, 把 z 转化为了随机变量, 由于 z 是由样本 X 编码的, 我们要求样本 X 也为随机变量
下面我们引入一些记号:

X : 样本随机变量
x_i : 样本随机变量的一个取值
z : 隐变量随机变量
z_i : 隐变量随机变量的一个取值

还有一些记号:

$p(z)$：随机变量z符合的先验分布
$p(X|z_i)$：已知z_i的情况下，X的条件分布

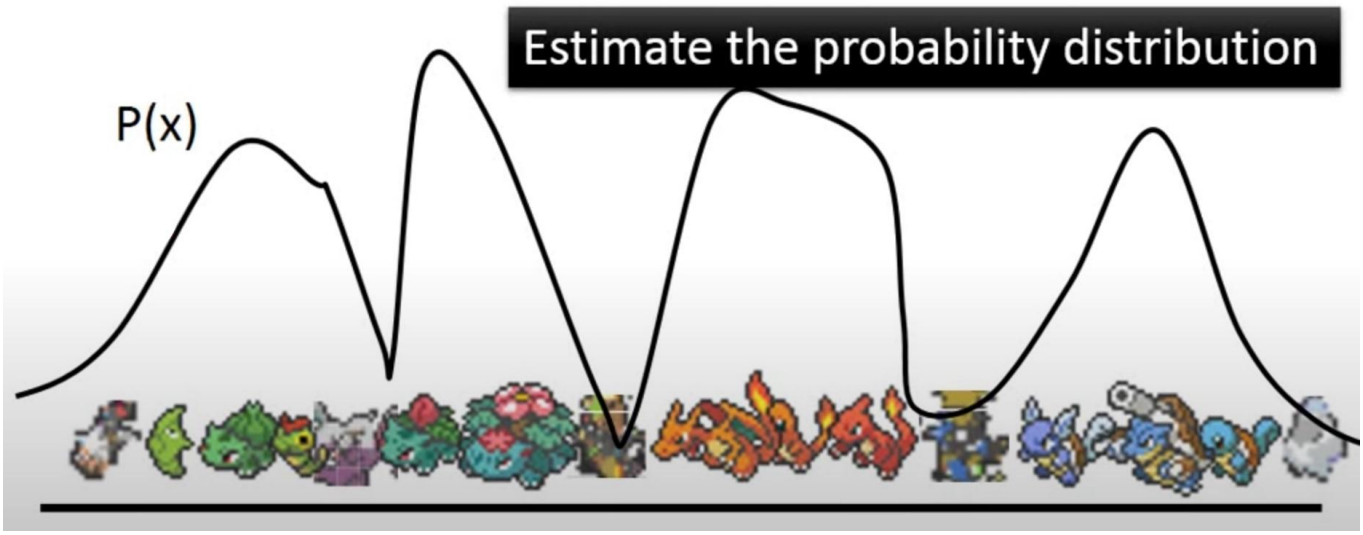
那么我们之前架构中提到的过程就变为:

1. 在先验分布 $p(z)$ 中随机采样得到一个 z_i
2. 根据 z_i, 从条件分布 $p(X|z_i)$ 中随机采样得到一个数据点 x_i, 这个数据点就是我们 $Decoder$ 生成的结果, 在大部分情况下, 我们采样的地方为均值 μ, 我们的 σ 作为一个超参数被抛弃
   这样就能做到一件事
   --在已知两个分布之后, 通过采样来生成数据

目标

我们的生成的最终目标是什么?
我们就是想在一个很接近真实分布 $P_{real}(X)$ 的分布 $p(x)$ 中采样, 生成一些有意义的数据
我们引入记号:

$p(x)$：样本分布的近似



比如这个图里有很多宝可梦 (看到这里应该已经有人知道是哪位老师的课了), 我们希望在 $P(x)$ 值比较大的地方取点, 得到一些 $P(x_i)$ 值比较大的样本点 x_i, 然后这些样本点就很有可能是宝可梦 (也许是杂交的)

假设: X 的分布

那么我们怎么用数学表示这个奇怪的分布 $P(x)$ 呢?

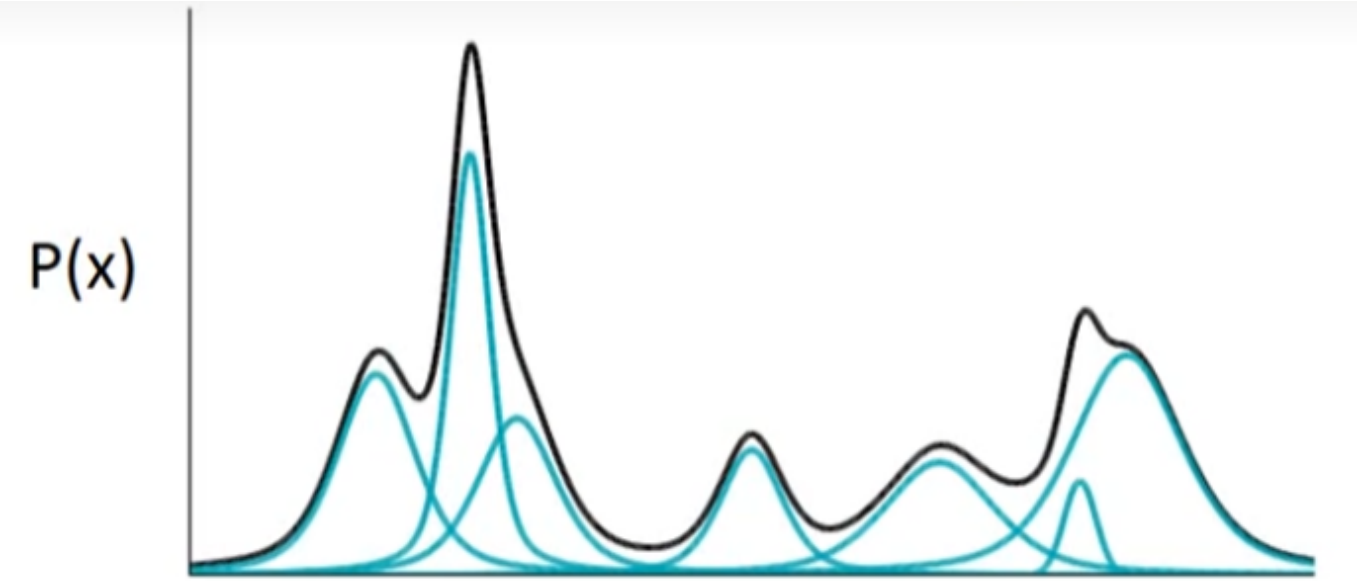
- 使用高斯混合模型

我们假定只要给定某个 z_i, X 都服从各维度独立的多元高斯分布, 即

$$p(X \mid z_i) = \mathcal{N}(X \mid \mu_i'(z_i;\theta), \sigma_i'^2(z_i;\theta) * I)$$

这表示取定一个 z_i, 它就是高斯混合模型中的一个子正态分布

高斯混合模型就能很好地近似这种多峰的情况, 而且由于它是由很多正态分布叠加形成的, 我们可以很简单地处理 (不一定效果最好, 但是能行)



可以看到, 里面有很多子正态分布, 高斯混合模型就是由很多子正态分布叠加形成的

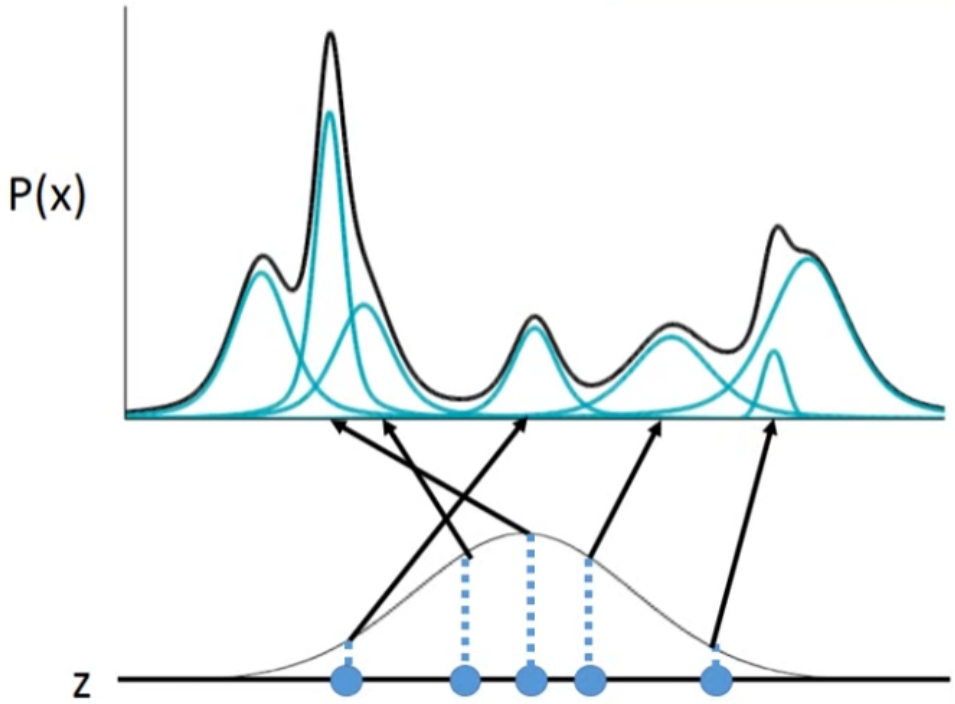
我们之前提到, $p(X|z_i)$: 已知z_i的情况下, X的条件分布, 由于 z_i 的取值是连续平滑的, 我们理所当然有:

$$X\text{的分布} = \int z\text{的分布} \times \text{在某个}z\text{下}X\text{的条件分布}$$

即:

$$p(x) = \int_z p(X|z)p(z)dz$$

类似一个全概公式
表现为:



即我们取一个 z 就确定了一个 $p(x)$ 中的子正态分布

现在我们先不讲具体的每个分布怎么求, 就先假定已经求到了, 那么现在是不是只要我们采样很多个 z_i, 就能通过积分求得 $p(x)$ 了?
可以是可以, 但是代价非常大, 因为 x_i 的维度往往很大, z_i 的维度也很大, 这就导致对于某个 x_i, 与之强相关的 z_i 很少很少, 我们很难取样到, 很有可能我们取样到的 z_i 和哪个样本都没关系

于是从正态分布 $p(x)$ 中直接采样得到 z_i, 再用来估计 $p(x)$, 是几乎不可行的
下面我们来解决这个问题

修正 z 的分布, 提高采样效率

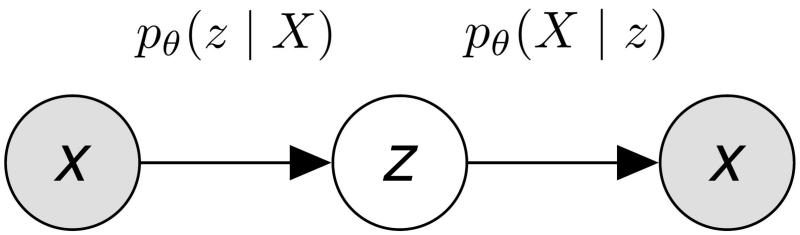
我们为了解决很难采样到和 x_i 强关联的 z_i 这个问题, 引入新的符号:

$$p(z|x_i)：已知样本x_i下, z的后验分布$$

直觉地, 在这个分布下采样得到的 z 应当都和 x_i 有很强的关系

由于我们先前的推导都是假定 z 为正态分布的基础上进行的, 我们理应要求这个后验分布 $p(z|x_i)$ 也应该符合正态分布 (修正的正态分布)

用图来表示的话, 就是这样的策略:



这样我们采样的 z 都和 x_i 有很强的关系, 就省去了很多采样步骤, 提高了效率

计算各个分布

我们之前一共提到四个分布, 我们分别来考虑怎么求:

$$
\begin{aligned}
&p(z): \text{随机变量} z \text{符合的先验分布} \\
&p(X|z_i): \text{已知} z_i \text{的情况下}, X \text{的条件分布} \\
&p(z|x_i): \text{已知样本} x_i \text{下}, z \text{的后验分布} \\
&p(x): \text{样本分布的近似}
\end{aligned}
$$

- $p(z)$
  由于这个 z 的先验分布我们已经用 $p(z|x_i)$ 取而代之了 (为了更好地取样), 所以不用求了
- $p(X|z_i)$
  我们采样一个 z_i, 要求 X 的分布, 其实就是给定一个向量, 输出均值和方差的问题
  这种问题我们交给 nn (神经网络) 去做, 没有思路, 难算, 就用神经网络拟合出来, 就是这么简单粗暴, 因为我们知道这个分布是一个正态分布, 所以只要拟合参数就行了
- 这个算法叫做变分贝叶斯算法 (变分就变分在这)
  这里其实就是 $Decoder$ 的部分, $Decoder$ 要做的就是根据一个 z_i 拟合出分布
  在 $WGAN$ 中, 我们也是这样解决问题的
- $p(z|x_i)$
  我们采样一个 x_i, 要求 z 的分布, 其实也是给定一个向量, 输出均值和方差的问题
  我们一样交给 nn
- 这个算法叫做变分贝叶斯算法 (变分就变分在这)
  这里其实就是 $Encoder$ 的部分, $Encoder$ 要做的就是根据一个 x_i 拟合出分布
- $p(x)$
  这是我们对样本真实的分布使用高斯混合模型做的一个近似, 我们有一个公式求它:

  $$p(x) = \int_z p(X|z)p(z)dz$$

  只要我们求到了 $p(X|z)$ 和 $p(z)$, 我们就能通过积分算出来这个分布

重参数技巧

原文就是"$Reparameterization\ Trick$", 我们来看看这个是用来干啥的

我们的架构中, 从神经网络训练的角度来看, 前向传播是可以做的, 因为后面每一步需要接受的参数我们都能算出来, 然而, 我们在前向传播的过程中, 使用了一个"从后验分布 $p(z|x_i)$"中随机采样的操作, 这个随机采样的操作怎么反向传播?
肯定不能, 直接从这个分布采样会导致无法通过梯度下降法来优化参数，因为采样操作本身是不可微的
我们得加一个参数, 来实现"可以反向解释的"随机采样

得到分布 $p(z|x_i)$ 之后, 我们先从多元正态分布 $N(0,I)$ 中采样得到一个 ϵ_i, 进行如下计算:

$$z_i = \mu_i + \sigma_i \cdot \epsilon_i$$

其中 ϵ 是可微的, 它的分布是正态分布
这样反向传播就可以跑通了

变分贝叶斯推断

我们之前说, 如果我们知道 $p(z|x_i)$ 这个后验分布是正态分布, 我们用神经网络去拟合出这个 u_i 和 σ_i, 这个过程就是变分贝叶斯推断
我们来看看这个过程具体怎么做:
要计算后验概率, 我们有贝叶斯公式, 注意, 分母等价于一个连续的全概公式

$$p(z|x) = \frac{p(z)p(x|z)}{\int p(z)p(x|z)dz}$$

分子是好算的, 因为我们的先验概率 $p(z)$ 假设为了标准正态分布, 似然分布假设为了正态分布, 其中两个参数就是我们的模型需要学出来的

对于分母现在有两种思路:

1. 用一个相对简单的分布 $q(z|x)$ 去近似 $p(z|x)$,这是变分推断要做的事
2. 尝试对$p(z|x)$进行取样，得到样本后，自然可以计算期望，这是 $MCMC$ 的思路

我们这里使用变分推断:
这里我们需要知道 KL 散度 (后面会介绍),
如果 KL 散度越小, 说明我们的变分分布越接近真实的后验分布
于是, 我们只要转成优化问题, 优化模型参数使得 KL 散度最小就行了
这就是变分推断的过程

损失函数

我们在讨论损失函数之前, 需要一些预备知识:

信息论

我们要用到信息论中的信息熵, 交叉熵, 相对熵 (KL 散度)
信息论是很有意思的, 这里提一下: 信息等于编码

信息熵

我们的信息熵可以看作一个事件包含的信息量, 也可以看作惊奇程度
确定性越大, 熵越大, 定义为:

$$H(x) = -\sum_{x} \log p(x)$$

其中 $p(x)$ 是事件 x 发生的概率, 我们可以发现, 概率越小的事件信息熵越大

- 信息论中, 表示该信息需要的编码长度

如果 X 是一个离散型随机变量, 那就要求平均编码长度了, 表示为:

$$H(p) = -\sum_{i=1}^{n} p(x_i) \log p(x_i)$$

其实就是一个加权平均

交叉熵

现在我们考虑一个分布 q, 我们希望用分布 q 去逼近分布 p
我们定义交叉熵为:

$$H(p, q) = -\sum_{i=1}^{n} p(x_i) \log q(x_i)$$

这意味着我们通过错误的分布 q 进行编码, 算出来的期望编码长度
当我们的拟合效果很好时, $q \to p$, 这个时候交叉熵就逼近 p 的熵, 也就是逼近 p 的最优编码长度

也就是交叉熵越小, 两个分布越接近

相对熵 (KL 散度)

$KL\ Divergence$
为了区分两个分布, 我们记 $q = p(z|x_i)$ 为 $Encoder$ 计算出的后验分布
定义如下:

$$D_{KL}(p||q) = -\sum_{i=1}^{n} p(x_i) \cdot \log \frac{q(x_i)}{p(x_i)}$$

这样可能不是很好看, 把它拆开:

$$\begin{aligned} D_{KL}(p\|q) &= -\sum_{i=1}^{n} p(x_i) \cdot \log \frac{q(x_i)}{p(x_i)} \\ &= -\sum_{i=1}^{n} p(x_i) \cdot \log q(x_i) + \sum_{i=1}^{n} p(x_i) \cdot \log p(x_i) \\ &= H(p, q) - H(p). \end{aligned}$$

意味着利用拟合的分布 q 去编码 X 得到的期望编码长度减去 X 的最佳编码长度
也就是利用 q 编码 X 所带来的额外编码长度 (可以理解为误差的一个量化)

这意味着, 优化 KL 散度, 就是优化两个分布之间的误差
当 $KL散度 \to 0$, 我们的误差也 $\to 0$

- 这意味着我们可以使用 KL 散度作为损失函数的一部分

极大似然估计

我们希望在分布 $p(X|z_i)$ (也就是最终的分布的一个子正态分布) 中概率大的地方对应了我们的 x_i, 这就是极大似然估计的思想
就是我们希望我们模拟的这个混合高斯模型能最接近真实的分布情况
即我们要最大化

$$\log p(X) = \sum_{x} \log p(x)$$

损失函数框架

看完本文再来品味这个框架是最佳食用方式
首先根据我们的目标, 是希望通过混合高斯模型拟合出 z 生成 x 的分布, 自然需要用到 MLE (极大似然估计), 我们需要让这个拟合分布最接近真实的分布, 这就是我们想法的出发点

然后, 我们根据这个框架, 通过公式的变换,可以拆分成两个子问题

1. 我们希望 $Encoder$ 拟合的后验分布能最接近真实的分布
2. 我们希望最大化 $Decoder$ 的拟合的混合高斯分布对真实分布的似然度

真正的损失函数

由于我们采样的 x 实际上可以有无穷多个值, 就变成积分, 再变换一下:

$$
\begin{aligned}
\log p(X) &= \int_z q(z \mid X) \log p(X) dz \quad \text{全概公式} \\
&= \int_z q(z \mid X) \log \frac{p(X, z)}{p(z \mid X)} dz \quad \text{贝叶斯定理} \\
&= \int_z q(z \mid X) \log \left(\frac{p(X, z)}{q(z \mid X)} \cdot \frac{q(z \mid X)}{p(z \mid X)} \right) dz \\
&= \int_z q(z \mid X) \log \frac{p(X, z)}{q(z \mid X)} dz + \int_z q(z \mid X) \log \frac{q(z \mid X)}{p(z \mid X)} dz \\
&= \ell(p, q) + D_{KL}(q, p) \\
&\geq \ell(p, q) \quad KL\text{散度非负}
\end{aligned}
$$

再调整一下顺序:

$$
\ell(p, q) = \log p(X) - D_{KL}(q, p)
$$

这意味着只要我们最大化 ℓ 就能最大化似然度 $\log p(X)$ 并且最小化 KL 散度 $D_{KL}(q, p)$
这意味着我们的 $Encoder$ 拟合的后验分布 $p(z|x_i) = q$ 会最接近我们真实的后验分布 p
不然 $Encoder$ 可能会输出一些没什么意义的分布

为了方便符号表示, 我们引入记号来区分先验分布和后验分布:
这也是变分贝叶斯推断的思想, 引入一个变分分布 $q_\phi(z|X)$ 来近似真实分布 $q_\theta(z|X)$

$$
q_\phi(z \mid X) : \text{近似的后验分布}
$$

$$
q_\theta(z \mid X) : \text{真实的后验分布}
$$

$$
\begin{aligned}
\ell(p_\theta, q_\phi) &= \int_z q_\phi(z \mid X) \log \frac{p_\theta(X, z)}{q_\phi(z \mid X)} dz \\
&= \int_z q_\phi(z \mid X) \log \frac{p_\theta(X \mid z) p(z)}{q_\phi(z \mid X)} dz \quad \text{贝叶斯定理} \\
&= \int_z q_\phi(z \mid X) \log \frac{p(z)}{q_\phi(z \mid X)} dz + \int_z q_\phi(z \mid X) \log p_\theta(X \mid z) dz \\
&= -D_{KL}(q_\phi, p) + \mathbb{E}_{q_\phi}[\log p_\theta(X \mid z)]
\end{aligned}
$$

左边的 KL 散度我们一般称为 $Latent\ Loss$, 相当于一个正则项, 提高模型泛化能力的
我们之前已经假设了 $q_\phi(z \mid X)$ 和 $p(z)$ 均服从高斯分布, 于是我们可以计算解析解:

先考虑一维情况:

$$
\begin{aligned}
D_{KL}(\mathcal{N}\left(\mu, \sigma^2\right) \| \mathcal{N}(0,1)) &= \int_z \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right) \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right) \\
&= \int_z \left(\frac{-(z-\mu)^2}{2\sigma^2} + \frac{z^2}{2} - \log\sigma\right) \mathcal{N}\left(\mu, \sigma^2\right) dz \\
&= -\int_z \frac{(z-\mu)^2}{2\sigma^2} \mathcal{N}\left(\mu, \sigma^2\right) dz + \int_z \frac{z^2}{2} \mathcal{N}\left(\mu, \sigma^2\right) dz - \int_z \log\sigma \mathcal{N}\left(\mu, \sigma^2\right) dz \\
&= -\frac{\mathbb{E}\left[(z-\mu)^2\right]}{2\sigma^2} + \frac{\mathbb{E}\left[z^2\right]}{2} - \log\sigma \\
&= \frac{1}{2}(-1 + \sigma^2 + \mu^2 - \log\sigma^2)
\end{aligned}
$$

其中 $\mathbb{E}[]$ 表示信息熵
推广到 d 维:

$$
D_{KL}\left(q_\phi(z \mid X), p(z)\right) = \sum_{j=1}^{d} \frac{1}{2}(-1 + {\sigma^{(j)}}^2 + {\mu^{(j)}}^2 - \log {\sigma^{(j)}}^2)
$$

其中 ${a^{(j)}}^2$ 代表向量 a 的第 j 个元素的平方

现在只剩最后一个问题了: 怎么求右边那项 $\mathbb{E}_{q_\phi}\left[\log p_\theta(X \mid z)\right]$
这一项称为重构损失 ($Reconstruction\ Loss$), 我们从 $q_\phi(z \mid X)$ 中采样多个 z_i 来近似求解这一项
即:

$$
\mathbb{E}_{q_\phi}\left[\log p_\theta(X \mid z)\right] \approx \frac{1}{m} \sum_{i=1}^{m} \log p_\theta\left(X \mid z_i\right)
$$

其中 $z_i \sim q_\phi\left(z \mid x_i\right) = \mathcal{N}\left(z \mid \mu\left(x_i ; \phi\right), \sigma^2\left(x_i ; \phi\right) * I\right)$
然后我们假设数据 x_i 是 K 维的, 就能用 MSE , 得到:

$$
\begin{aligned}
\log p_\theta\left(X \mid z_i\right) &= \log \frac{\exp\left(-\frac{1}{2}(X-\mu')^{\mathrm{T}} {\Sigma'}^{-1}(X-\mu')\right)}{\sqrt{(2\pi)^k |\Sigma'|}} \\
&= -\frac{1}{2}(X-\mu')^{\mathrm{T}} {\Sigma'}^{-1}(X-\mu') - \log\sqrt{(2\pi)^k |\Sigma'|} \\
&= -\frac{1}{2}\sum_{k=1}^{K} \frac{(X^{(k)} - {\mu'}^{(k)})^2}{{\sigma'}^{(k)}} - \log\sqrt{(2\pi)^K \prod_{k=1}^{K} {\sigma'}^{(k)}}
\end{aligned}
$$

这样我们单独损失函数的每一块都算好了

为了损失函数反应样本量, 取一个平均,于是损失函数就表示为:

$$
\begin{aligned}
\mathcal{L} &= -\frac{1}{n}\sum_{i=1}^{n}\ell(p_\theta, q_\phi) \\
&= \frac{1}{n}\sum_{i=1}^{n} D_{KL}\left(q_\phi, p\right) - \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{q_\phi}\left[\log p_\theta(x_i \mid z)\right] \\
&= \frac{1}{n}\sum_{i=1}^{n} D_{KL}\left(q_\phi, p\right) - \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}\log p_\theta\left(x_i \mid z_j\right)
\end{aligned}
$$

我们通过从 $q_\phi(z \mid x_i)$ 中采样 m 次 z_j, 来逼近 $\mathbb{E}_{q\phi}\left[\log p_\theta(x_i \mid z)\right]$

你可能会问: 为什么计算 $Encoder$ 的时候我们不用采样的方法, 而是使用了贝叶斯变分推断去优化 KL 散度, 而计算 $Decoder$ 的时候却使用采样的方法

这是因为我们其实是从 $q_\phi(z \mid x_i)$ 中采样得到 z_j, 随着网络的训练，近似后验 $q_\phi(z \mid x_i)$,很快就会比较接近真实的后验分布, 这样一来，我们有很大可能能够在有限次数的采样中，采样到与 x_i 关联的 z_j, 所以可以这样采样

而在 $Encoder$ 那里, 我们还没有计算出 z_i, 就无从知道哪个 x_i 和哪个 z_i 相关, 所以没有办法使用采样的方法 (因为空间是很大的, 采样无法模拟真实情况)

我们把前面算的似然度代进去展开得到:

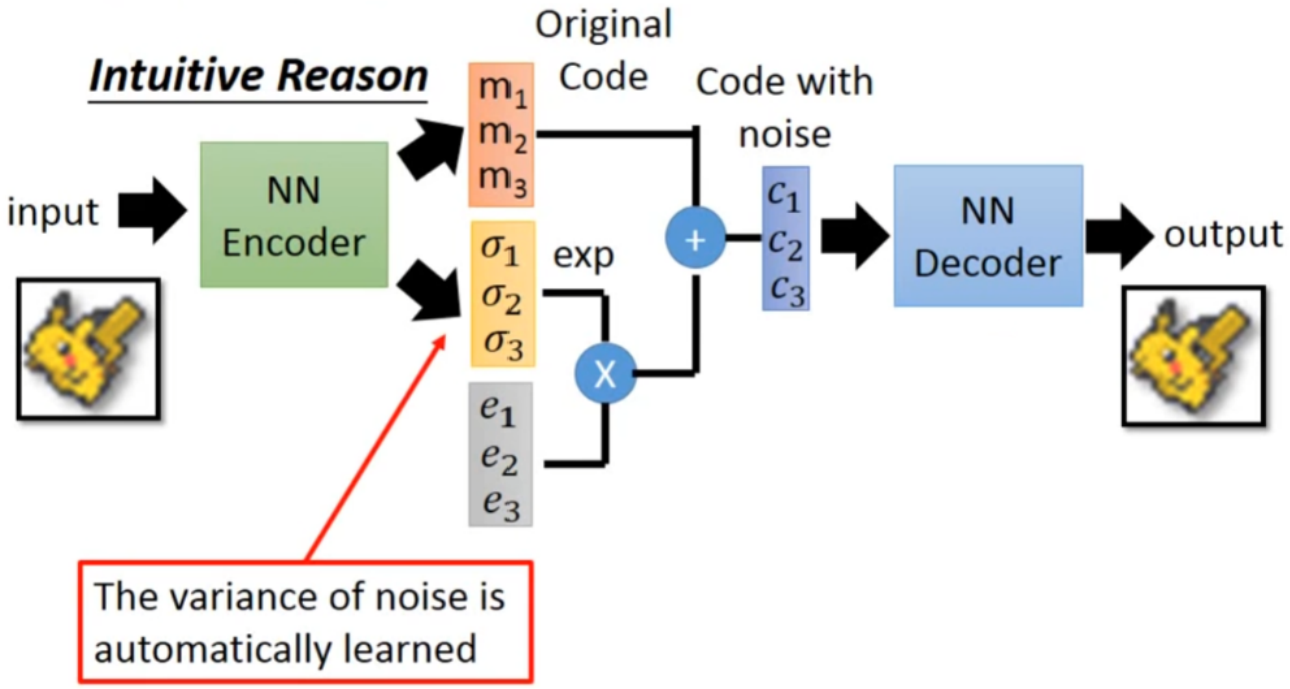
$$
\begin{aligned}
\mathcal{L} &= \frac{1}{n}\sum_{i=1}^{n} D_{KL}\left(q_\phi, p\right) - \frac{1}{n}\sum_{i=1}^{n}\log p_\theta\left(x_i \mid z_i\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{d}\frac{1}{2}(-1 + \sigma_i^{(j)^2} + \mu_i^{(j)^2} - \log\sigma_i^{(j)^2}) \\
&- \frac{1}{n}\sum_{i=1}^{n}\left(-\frac{1}{2}\sum_{k=1}^{K}\frac{(x_i^{(k)} - \mu_i'^{(k)})^2}{\sigma_i'^{(k)}} - \log\sqrt{(2\pi)^K\prod_{k=1}^{K}\sigma_i'^{(k)}}\right)
\end{aligned}
$$

我们是不需要用到超参数 σ' 的, 不妨假设为 $\frac{1}{2}$, 于是就得到了论文里很漂亮的损失函数:

$$
\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{d}\frac{1}{2}(-1 + \sigma_i^{(j)^2} + \mu_i^{(j)^2} - \log\sigma_i^{(j)^2}) + \frac{1}{n}\sum_{i=1}^{n}\|x_i - \mu_i'\|^2
$$

其中，x_i 代表第 i 个样本，是 Encoder 的输入
μ_i 和 σ_i^2 是 Encoder 的输出，代表 $z \mid x_i$ 的分布的参数
z_i 是从 $z \mid x_i$ 中采样得到的一个样本，它是 Decoder 的输入
μ_i' 是 Decoder 的输出，代表利用 z_i 解码后对应的数据点 $\tilde{x}_i$

现在我们可以对这个模型有新的观点了:



我们输入样本, $Encoder$ 拟合出后验分布, 然后我们施加噪声采样 (重参数技巧), 将采样结果送入 $Decoder$ 拟合出真实分布, 不过我们一般只要均值而不要方差, 直接输出均值

右边一项, 就是希望最小化 $Decoder$ 产出和 $Encoder$ 输入的差距, 这和我们的自编码器的思想是一样的

到这里，我们终于得到了在假设先验、后验、似然均是高斯分布的情况下，VAE 最终的损失函数, 可喜可贺, 可喜可贺

最后, 我们只要用随机梯度下降去优化这个问题就行了 (其他也可以)

代码

代码我写在 $colab$ 上了, 方便大家如果没有电脑也能学习, 点击链接访问:

- 香草味自编码器:
  自编码器讲解代码 - Colab (google.com)
- 卷积变分自编码器
  cvae.ipynb - Colab (google.com)

参考文献: 原论文

系列完