实验主题

本实验中，你将跟随指导，完成VAE原理的基本推导理解，搭建基本的深度学习网络，完成一个基本的深度学习任务的训练代码框架。前两部分主要目的为：了解VAE的基本原理与数学推导，了解深度学习模型的设计以及代码实现，完成基本的深度学习训练代码，了解需要哪些组件。我们不会过分要求实验效果和调参等具体操作，也不需要你的各种原理上的改善，更多是一个手把手的教学，希望你能够仔细跟随实验指导阅读完成每一部分，评分原则只与完成度相关！最后一部分会强调自己设计模型，评分与模型能力有很大关系！

0 补充知识(30pt)

0.1 VAE的优化目标：KL散度 (20pt)

我们首先需要的是，如何去衡量我们的分布与目标分布的相似度呢？这里就引入了一个概念：KL散度

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx = E_{x\sim p(x)}[\log \frac{p(x)}{q(x)}]$$

首先，请你证明一些KL散度的基本性质

- 任意两个分布p, q，$D_{KL}(p||q) \geq 0$，取0时当且仅当$p = q$ **(10pt)**

  > 提示1：可以使用引理：$x - 1 \geq \log x$
  >
  > 提示2：也可以使用Jensen不等式：若$f(x)$为凸函数($\nabla^2 f(x) \geq 0$)，则$E[f(x)] \geq f(E[x])$，取等时$f(x) = const$

- 推导一下两个一维正态分布（高斯分布）的KL散度计算结果，$p(x) \sim \mathcal{N}(\mu_1, \sigma_1^2), q(x) \sim \mathcal{N}(\mu_2, \sigma_2^2)$ **(10pt)**

  > 提示1：一维高斯分布的概率密度函数为:$p(x) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(x-\mu)^2}{2\sigma^2})$
  >
  > 提示2：只需要给对数拆成两项，分别计算即可，不需要太多使用概率密度函数的表达式进行积分，而是合理利用期望和方差简化运算

- 这里我们给出高维的结论（不需要你证明，后续可能用到），假设$p(\mathbf{x}) \sim \mathcal{N}(\mu_1, \Sigma_1), 1(\mathbf{x}) \sim \mathcal{N}(\mu_2, \Sigma_2)$，这里的$\mu$为n维向量，$\Sigma$为n维方阵，则

  $$D_{KL}(p||q) = \frac{1}{2}\log \frac{|\Sigma_2|}{|\Sigma_1|} + \frac{1}{2}tr(\Sigma_2^{-1}\Sigma_1) + (\mu_1 - \mu_2)^T\Sigma_2^{-1}(\mu_1 - \mu_2) - \frac{1}{2}n$$

0.2 EM算法数学原理(10pt)

> 这部分中关于极大似然估计的内容应该是在概统课程中有讲过的，所以不做问题

EM算法本质上其实也是一种MLE（极大似然估计），不过有些许不同。我们希望从采样的样本数据x来估计这个分布的参数θ，也就是最大化$p(\theta|x)$，极大似然估计的想法就是

$$\theta^* = \arg\max_\theta p(\theta|x) = \arg\max_\theta \frac{p(\theta)p(x|\theta)}{p(x)} = \arg\max_\theta p(x|\theta) = \arg\max_\theta \log p(x|\theta)$$

但现在在EM算法中有些许不同，我们没办法直接得到$p(x|\theta)$，因为x不是能够直接建模的变量，而是需要从一个可建模可观测或者说可控制的隐变量z（服从已知分布$q(z)$）过渡得到。所以我们现在要优化的目标需要进一步展开，需要一步全概率公式即可

$$L(\theta) = \log p(x|\theta) = \int_z q(z) \log p(x|\theta) dz = \cdots (your\ proof) \cdots = ELBO + \int_z q(z) \log \frac{q(z)}{p(z|x,\theta)} dz$$

- 补充上面的证明，并推导出ELBO是什么 **(10pt)**

> 提示1：完全可以直接做个差看看ELBO是什么
>
> 提示2：通过条件概率公式，给$p(x|\theta)$拆成含z的概率的比值

后面一项是一个$D_{KL}(q(z)||p(z|x,\theta))$，这一项会在当$q(z)$和$p(z|x,\theta)$相同时取0。所以EM算法的两步骤其实是在做这样的事：EM算法中的隐藏分布$q(z)$是一个完全可控的分布，所以我们希望能够同时优化q,θ，让ELBO直接作为优化目标，写成数学形式就是

$$max_{q,\theta} ELBO$$

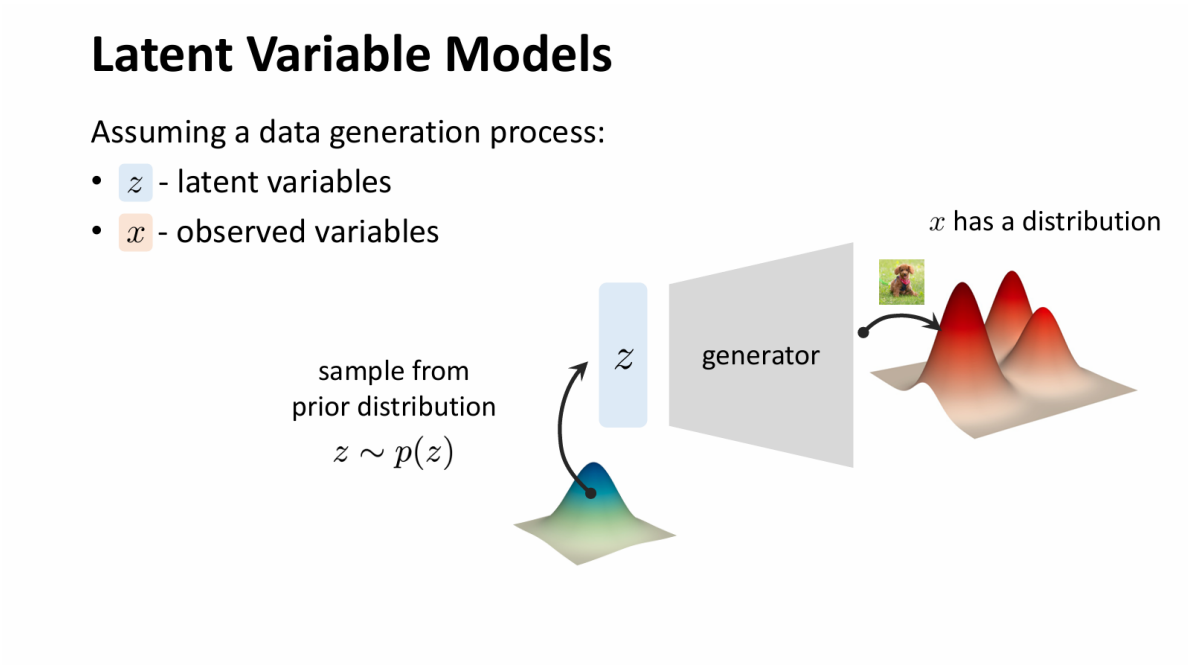
- E-step：我们直接令$q^{(t+1)}(z) = p(z|x,\theta^{(t)})$，这里的$\theta^{(t)}$表示当前的参数（需要不断更新），这样我们的后一项直接为0了
- M-step：我们优化这个ELBO即可$\theta^{(t+1)} = \arg\max_\theta J(\theta, q^{(t+1)}) = \arg\max_\theta ELBO$

1 VAE原理以及推导 (10pt)

1.1 图像生成任务的基本原理

基本假设：我们认为，图像在数学层面上 (C(channel，RGB是3，灰度图会是1) × W (高度)× H (宽度)) 是服从一个高维分布的。我们用随机变量$\mathbf{x}$来表示。而对于我们给定的图像数据集里(这个数据集可大可小)，我们认为我们的分布为$p_{data}(\mathbf{x})$。我们的任务就是建模这个分布，用我们带有可训练的参数的$p_\theta(\mathbf{x})$去尽可能逼近它。

1.2 VAE架构与Motivation



我们希望从一个我们可控的简单分布z（一般可以是一个简单的标准高斯分布），过渡到我们需要逼近的$p_{data}(x)$，而建模这个过程的$p_\theta(x|z)$，就是我们的模型要做的

1.3 损失函数的推导 (10pt)

根据上述分析，我们的优化目标可以写成

$\theta^* = \arg\min_\theta D_{KL}(p_{data}||p_\theta)$

- 推导$\theta^* = \arg\max_\theta \mathbb{E}_{x\sim p_{data}}[p_\theta(\mathbf{x})]$**(10pt)**

这个式子对于实际的操作就方便太多了，这里的期望我们可以通过采样来近似，我们从p_{data}中直接采样即可，然后通过对数似然的方法优化，也就是从数据集中随机采样$\{\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \cdots, \mathbf{x}^m\}$，来最大化$\sum_{i=1}^m \log p_\theta(\mathbf{x}^i|\theta)$

但是还有个问题，我们是无法从Input的$\mathbf{x}$直接得到最后的Output，我们在使用模型做生成的时候，只能从中间的$\mathbf{z}$，我们假设服从分布$q(\mathbf{z})$ (实际多数为标准高斯分布)，来生成。也就是说，我们无法直接使用极大似然估计的方法优化p_θ，为了考虑到中间的$\mathbf{z}$，我们就需要模仿**EM算法**的优化方法

$$\mathcal{L}(\theta) = \log p_\theta(\mathbf{x}) = \int_{\mathbf{z}} q(\mathbf{z}) \log p_\theta(\mathbf{x}|\theta) d\mathbf{z} = \cdots (your\ proof) \cdots = ELBO + D_{KL}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x},\theta))$$

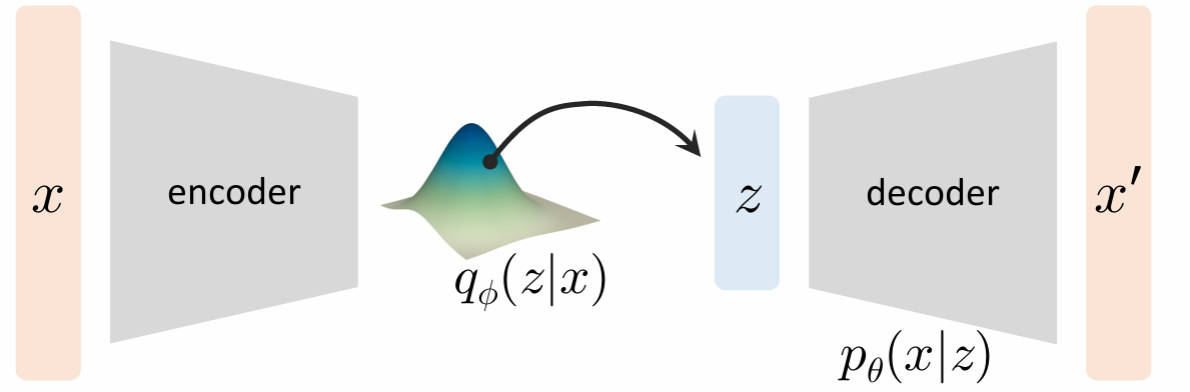
一方面与EM算法相同，我们只想要去优化ELBO。但另一方面，我们很难用当前的$p(x,\theta)$（这个我们也写不出来）去估计$q(z)$。某种程度上，这一项是无法通过人为来计算修改的，所以就直接不管了，相当于我们去优化一个下界

> 其实本质上也可以用EM算法去做生成任务，emmm不过效果有限，感兴趣的同学可以自行了解

那么我们就给ELBO拆开，就可以写成我们需要的优化目标了

$$\min_{\theta,\phi} -ELBO = \mathcal{L} = \underbrace{\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[-\log p_\theta(\mathbf{x}|\mathbf{z})]}_{reconstruction} + \underbrace{D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))}_{regularization}$$

我们用两个网络去建模我们需要的分布，一个$p_\theta(\mathbf{x}|\mathbf{z})$，叫做Decoder，一个$q_\phi(\mathbf{z}|\mathbf{x})$，叫做Encoder。完整的架构如下：



1.5 图像生成任务的具体使用

推导了VAE的基本原理以及损失函数后，我们再来看看在图像生成任务中如何使用这个模型。

我们将所有的分布都用高斯分布来建模，其中我们的损失函数中一共涉及三个分布：1.隐变量/压缩后的信息$p(\mathbf{z})$, 2.Decoder$p_\theta(\mathbf{x}|\mathbf{z})$, 3.Encoder$q_\phi(\mathbf{z}|\mathbf{x})$

- $p(\mathbf{z})$我们一般直接建模成一个标准高斯分布$\mathcal{N}(\mathbf{0}, I)$（有一些相关工作研究这个，感兴趣可以研究下）
- $p_\theta(\mathbf{x}|\mathbf{z})$我们假设成$\mathcal{N}(\mu_\theta(\mathbf{z}), \sigma^2 I)$，其中$\mu$是神经网络的输出，$\sigma$是一个人为设定的超参数

> 至于为什么σ是超参数而不是神经网络输出，一方面在于最后的损失函数中，σ只是正则化项的一个系数，另一方面在于我们实际在使用模型进行生成的时候，重点在于这个分布的期望，可以理解为模型所认为*最理想*的生成图像，也是我们要的。当然，我们可以在其上加上任意的方差

- $q_\phi(\mathbf{z}|\mathbf{x})$我们假设成$\mathcal{N}(\mu_\phi(\mathbf{x}), \sigma_\phi(\mathbf{x})I)$，我们神经网络的输出就是$\mu, \sigma$这里我们假设协方差矩阵为单位阵形式，可以很大程度上减少模型输出，有效防止过拟合（当然你也可以调整输出看看效果区别）

最后我们的损失函数可以化简为（这里不用你来推导，感兴趣可以自行尝试，只是一个代入表达式化简的过程，高斯分布之间的KL散度上面已经给出过）

$$\mathcal{L}(\theta, \phi, \mathbf{x}) = \sum_{i=1}^{d} \frac{1}{2}(-1 + (\sigma_\phi^{(i)})^2 + (\mu_\phi^{(i)})^2 - \log(\sigma_\phi^{(i)})^2) + \frac{1}{2\sigma^2}||\mathbf{x} - \mathbf{z}||_2^2$$

注：这里需要一个叫做重参数化技巧(reparameterization trick)，因为这里的$\mathbf{z}$是隐藏code的一个采样，所以不能直接使用采样的方式来进行使用，否则梯度无法反向传播回去，但我们知道$\mathbf{z}|\mathbf{x}$的分布，所以完全可以合成一下

$\mathbf{z} = \mu_\theta(\mathbf{x}) + \epsilon \cdot \sigma_\theta(\mathbf{x}), \quad \epsilon \sim \mathcal{N}(\mathbf{0}, I)$，这样我们只需要采样$\epsilon$就可以了

2 使用VAE完成MNIST图像生成 (30pt完成度+10pt评测效果)

2.0 数据集介绍与任务介绍

MNIST数据集包含了约70k张28×28的灰度图，内容为手写的数字。具体介绍见ylecun/mnist · Datasets at Hugging Face。这里助教已经处理好了，在dataset文件夹中，训练集、验证集、测试集按照5:1:1的比例设置

我们的任务是，补完VAE深度学习的代码框架，包括模型设计，训练代码以及评测代码等部分，**你需要填写的代码都位于submission.py文件中，请只修改TODO标注的部分**，其他部分的代码都是没有问题的。（如果觉得为了效果或者想做其他尝试而进行修改，可以改动后在提交时告知助教，比如写一个README.md，**并不推荐**）

注：很多部分的信息都是介绍性质，相当于一个手把手的教程，并不需要你去实现代码

2.1 处理数据

首先我们需要实现一个Dataset类，这个类是为了来存储所有训练的数据，我们能够很方便获取训练的数据，处理成适合神经网络输入的情况。这里我们已经在data_utils.py中实现了MNISTDataset类。

在训练过程中，我们需要的是转化成Dataloader类，这样能够更方便深度学习任务的迭代获取，按batch取出我们需要的数据。

2.2 实现你的VAE模型 (15pt)

在submission.py的VAE类中，你需要按照注释完成2.2.1到2.2.4的代码补齐，设计你的模型，并实现encode，decode，forward三个函数。千万注意tensor的形状和数值大小问题，我们的输入输出都应该是[batch_size, 28, 28]的大小（不确定可以手动调整下再处理），每个元素都是[0, 1]之间的torch.float32类型

> 注：助教在model.py中提供了一个建议的demo，**完全可以照搬**，这个demo在合理的调整超参数充分训练下，也**能跑出相当不错的效果**！这一部分更多是一个基本深度学习的教学指导，不会过多强调你的效果之类，做代码补充更多是为了方便大家尝试不同架构，了解不同架构的效果特点等

2.3 实现vae的loss计算 (15pt)

完成vae_loss函数的补齐，按照1.5中已经推导出的最终公式代入计算即可。

> 提示1：注意我们输出的各项是什么
>
> 提示2：注意我们输入输出都是所有batch一起的，最后要按batch取均值，也就是最后输出是一个数据，而非一整个tensor
>
> 提示3：这里的var是超参数，可以动态调整，这里的重构loss除了mse loss之外，也有其他可以的尝试，只要体现出重构图像与原图像之间的直接的相似程度就可以，不过mse loss就足够，我们最后的评测指标其中之一就是mse loss

2.4 训练！

在trainer.py里，助教已经实现了Trainer类，这个类的作用是，集成了整个训练过程中的各种必要功能。比如训练代码，比如存储训练权重，记录训练中的loss，调用验证集实时验证等。这部分不需要你的补充，但希望能够尽量看懂在做什么，这些输入的超参数是控制什么的，尤其是train函数的实现。

在train.py里，助教已经写好了训练代码，这一部分很简单，只要准备好内容超参数，调用trainer类即可

下面，你可以用这行脚本进行训练：

```
python train.py --task VAEwolabel --epochs 10 --latent_dim 20
```

> **请仔细查看train.py中引入的超参数，在你具体进行实验时，可能需要反复调整一些超参数**
>
> 可以写一个.sh文件来方便你的运行，遍历超参数调参等

2.5 测试！

在inference.py中，助教已经完成了所有的inference代码，其中包括两个任务：

- 重构任务：输入图像，经过encoder和decoder后，得到一个重构的图片，计算重构图片与原图片之间的差异大小
- 生成任务，对于每一种label，从标准高斯分布中随机sample我们的隐变量$\mathbf{z}$，通过decoder进行生成

执行脚本进行生成（**调整必要超参数**）

```
python inference.py --task VAEwolabel --latent_dim 20 --checkpoint_path YOUR
CHECKPOINT PATH
```

注：在这里我们测试验证集上的模型效果，实际评分时会在助教自己的测试集上（不可见）进行评测！
但你可以用验证集结果作为参考，若你的改进在验证集上能够明显提升效果，那么大概率在助教测试集上也能够取得更好评分。

3 条件生成任务 (20pt完成度+30pt评测效果)

对于上述的生成任务中，我们会发现，虽然能够生成图像，但似乎有点问题：我们的条件，也就是label和生成的图像不能说联系紧密，只能说毫不相关。这当然合理，**因为我们的模型就没有用到label**，我们训练重构之类的全过程中，都没有用到图像的label。那这样的话，我们就很难做生成任务，因为实际的生成中，都需要有引导有控制地生成，就比如说我们就想让模型画“5”，而不是随机地给我生成任意可能

图像。那就有同学问了，助教助教，有没有简单又能有条件控制的改进呢？有的同学，有的，这样的改进还有很多。不过这一部分需要你自己去设计尝试。

任务：

- 完成submission.py中GenModel的模型设计，forward等函数
- 进行训练和生成，只需要给之前的脚本中，task参数换成Genwithlabel即可

> 提示1：注意必要超参数的引入，比如说模型的latent_dim
>
> 提示2：其实VAE就完全可以做这个工作，不过需要进行小小的改动即可
>
> 提示3：大胆尝试！这个数据集和任务都很简单，基本各种简单或复杂的模型都完全可以做到这个任务，甚至原来的VAE加上一些非模型上的简单处理都可以实现这个任务！

4 评估你的结果

最后，我们当然要看看自己的模型效果如何了，只是肉眼去看肯定不够，需要我们具体的看看模型的各项指标

- MSE score: 这个指标比较简单，就是生成图像与原图之间像素上的mse loss
- SSIM score: 范围在[-1, 1]之间，越接近1越好，代表生成图像与原图像之间的感官相似程度，如果是负的，代表负相关
- FID score: 这里我们用的是简化的FID score，统计各类别下，原图和生成图像的均值和方差，越接近，则分数越低

> 注：原始的FID score是对图像经过了Inception v3网络后得到的latent code进行分布提取，再来计算分数，但我们这里为了图方便，而且模型本身也够简单，所以直接从简单的方法来了

调整必要参数后运行grade.py即可进行评测

我们的计算公式为：

- 对于VAE生成任务：$Grade = \min((ssim * 5 + \max((0.1 - mse), 0) * 50) * bonus, 10)$
- 对于Gen生成任务：
  $Grade = \min((ssim * 10 + \max(0.1 - mse, 0) * 100 + \max(10 - fid, 0)) * bonus, 10)$
- 其中这个$bonus$是对压缩率的影响的一个系数，压缩率$R = 1 - \frac{latent_dim}{784}$，$bonus = 0.5 + R$

> 设置bonus的意义在于，能够在保证重构、生成质量的情况下，体现出图像压缩的功能，防止比如说，隐藏层的维度就设置成784，encoder,decoder就是两个单位阵，来满足完全一比一照搬的情况（这样照搬虽然能做重构任务，但做不了生成任务，而且也毫无实际意义）

推荐阅读

- Deep Generative Models. by Kaiming He. MIT S987
- VAE原论文
- Variational Autoencoders. Stanford CS