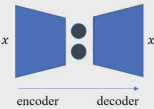


# Introduction to Generative Model

## Variational AutoEncoder (VAE)

- An autoencoder is a feed-forward neural network whose job is to take **an input  $x$**  and **predict  $x$** .
- Trivial (short-cut) solutions exist: a neural network can learn identity mapping  $x = f(x)$
- Bottleneck architecture



- 最起初的时候自编码器是用于**数据降维**，**可视化图片**中的表现。也可以用来压缩文件，编码文字等。
- 我们关注的是其生成模型的用途。最简单的自编码器只考虑线性变换，就使用矩阵乘法即可。

$$output = VUx, U \in R^{k \times d}, V \in R^{d \times k}, x \in R^{d \times 1}$$

，其中 $x$ 经过矩阵 $U$ 的乘法成为隐藏层hidden state，之后经过矩阵 $V$ 的作用恢复。我们一般将 $k \ll d$ ，即认为模型其实进行了维度压缩工作。因此为了确保encode和decode之后结果变化最小，使用 $minimize ||VUx - x||_2$ 的方式即可定义损失。满足约束的值有众多，通过扩倍和缩倍可以找到无数满足要求的解。这个问题的解就是PCA主成分分解。

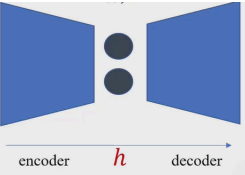
- 将上述的线性问题泛化，得到这个问题的根本描述是：

$$f(g(x)) = x$$

，其中 $h = g(x)$ 被认为是 $x$ 的潜表达。 $f$ 和 $g$ 不应该太难，否则容易直接学会一一映射，造成过拟合。这种压缩方式与mp3和jpeg的压缩是不一样的，压缩方式是学出来的，所以不一定稳定。

- **Vanilla Autoencoder不是生成模型**，但是如果我们修改模型架构，将encoder之后的结果约束为一个符合高斯分布的 $h \sim N(0, 1)$ ，经过decoder之后与输入的相似度很接近。此时放弃encoder，将输入改成一个标准正态分布，只取用decoder，则这就是一个合理的生成模型。
- 上述的想法很不错，但是我们需要明白：如何让 $h$ 在训练之后变成一个已知的分布？

- Autoencoder
  - Encoder:  $h = g(x)$
  - Decoder:  $x' = f(h)$



- f and g are **deterministic** function

- Variational Autoencoder
  - Encoder:  $h \sim g(x)$
  - Decoder:  $x' = f(h)$

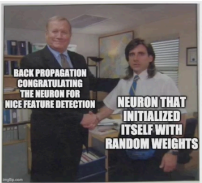
- g is a **stochastic** function
- f is a deterministic function
- h is a **random variable**

- 首先，我们需要定义**Stochastic Function(随机方程)**，即输入是一个固定值，但是输出是从一个分布之中采样得到的（实现方法是encoder编码器负责估计出均值和方差，使用这两个值构建一个满足要求的分布，再在这个已知的分布中进行采样）。
  - Variational Autoencoder: forward process
    - Input:  $x$
    - $[u, \sigma] = g(x, \theta_{enc})$
    - $h \sim \text{Normal}(u, \sigma^2)$
    - Output:  $x' = f(h, \theta_{dec})$
  - Loss design: evaluating the difference between  $x$  and  $x'$ 
    - Mean square error:  $L = ||x - x'||_2$
    - Other loss function can be applied
  - Parameter ( $\theta_{enc}$  and  $\theta_{dec}$ ) updates
- 这个图中绝大部分已经很容易理解，唯一需要再思考的是如何更新参数 $\theta_{enc}$ 和 $\theta_{dec}$ 。 $\theta_{dec}$ 是很容易更新的，只需要通过之前定义的损失函数求偏导即可，比较困难的是 $\theta_{enc}$ ，通过同样的代替和计算我们可以得知，encoder本身是可导的，decoder本身也是可导的，但是从encoder的结果构造正态分布并且采样出hidden state的过程是不可导的。

- Parameter ( $\theta_{enc}$  and  $\theta_{dec}$ ) updates
  - How to update parameter  $\theta_{dec}$ ?

$$L = ||x - x'||_2 = ||x - f(h, \theta_{dec})||_2$$
$$\Rightarrow \frac{\partial L}{\partial \theta_{dec}} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial \theta_{dec}} \quad \checkmark$$

- $L$  is differentiable with respect to  $f$
- $f$  is differentiable (almost everywhere) with respect to  $\theta_{dec}$

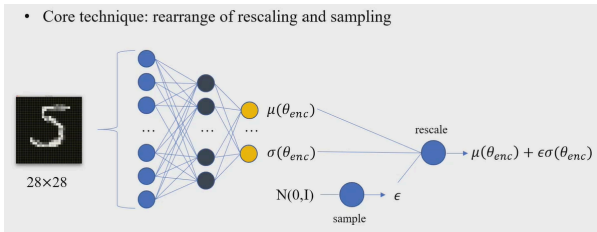


-

- Parameter ( $\theta_{enc}$  and  $\theta_{dec}$ ) updates
  - How to update parameter  $\theta_{enc}$ ?

$$L = ||x - x'||_2 = ||x - f(h, \theta_{dec})||_2, where \ h \sim Normal(g(x, \theta_{enc}))$$

- How to compute  $\frac{\partial L}{\partial \theta_{enc}}$ 
  - $L$  is differentiable with respect to  $h$
  - $g$  is differentiable with respect to  $\theta_{enc}$
  - $g$  is NOT differentiable with respect to  $h$ !**



- Variational Autoencoder
  - Input:  $x$
  - $[u, \sigma] = g(x, \theta_{enc})$
  - $\epsilon \sim Normal(0, I)$
  - $h = \mu + \epsilon \sigma$
  - Output:  $x' = f(h, \theta_{dec})$

$$L = ||x - x'||_2 = ||x - f(h, \theta_{dec})||_2 \\ = ||x - f(g_u(x, \theta_{enc}) + \epsilon g_\sigma(x, \theta_{enc}), \theta_{dec})||_2$$

- All parameters are trained by gradient descent end-to-end

为此我们使用**Reparameterization trick(重参数化)**的方式，如上右图。其主要的核心是由于采样是无法求导的，而高斯分布有特殊性， $N(\mu, \sigma)$ 采样的结果等效于 $N(0, 1)$ 中采样 $\epsilon$ ， $\mu + \sigma^* \epsilon$ 为结果。因此可以将算式改为下面的样子，这样就是处处可导的。

注：解决的是采样挡梯度的问题，直接对复杂的高斯分布求导是错误的，但是如果把随机性移到与参数无关的epsilon之上，就可以对另外两个参数正常求导，每次前向传播的时候使用Monte-Carlo approximation即可。

- Variational Autoencoder
  - Input:  $x$
  - $[u, \sigma] = g(x, \theta_{enc})$
  - $\epsilon \sim Normal(0, I)$
  - $h = \mu + \epsilon \sigma$
  - Output:  $x' = f(h, \theta_{dec})$

- Reconstruction loss  $L = ||x - x'||_2$
- Regularization loss  $L_r = KL(N(\mu(x), \sigma(x))||N(0, I))$
- Overall loss =  $L + \lambda L_r$

**Regularization Loss**：为了确保encoder之中的 $\mu$ 和 $\sigma$ 分布不要过于奇怪导致模型学不到东西，其需要使用KL散度来规范化参数的值，确保值与标准正态分布接近一些。这样在decoder之中可以从标准正态分布之中去采样，decoder可以直接采用使用。

请注意，**epsilon的正态分布和regularization的标准正态分布是不一样的**。第一个的目的是重参数化，第二个的目的是生成的参数更像标准正态分布，使得能从 $N(0, 1)$ 中直接采样使用decoder。最后使用

$$Overall = L + \lambda L_r$$

的方式来计算总共的损失，总共Reconstruction Loss和Regularization Loss。

这里需要补充：**为什么需要Regularization Loss?**，如果我们不加以限制 $\mu$ 和 $\sigma$ 的参数范围，其会倾向于对于每个训练样本记录一个点( $\sigma=0$ )，即直接对点进行映射，存储在潜空间之中。这样潜空间的大部分地区都是没有被涉及的，重建任务会表现很好，但是一到生成任务，其接收到的信息是来自标准正态分布的，大概率不在点为构造的潜空间之中，因此容易直接生成乱码。如果将其化归到标准正态分布之中，其生成范围更有序，更容易生成在指定的区域之中。

### Probabilistic View of VAE:

我们一般假设一个数据集的内容分布是存在的，但是是未知的，因此可以从一个分布之中iid的取样，VAE确实认为数据集中的内容符合一个分布，认为数据的生成过程有latent code，即潜表达。我们无法建模出 $x$ 的分布，但是我们能够建模出潜表达 $z$ 的分布(latent code distribution)，这个分布我们称为先验分布**prior distribution**。

- Most works assume data ( $x$ ) is sampled from a fixed but unknown distribution
- Some works care about “how the data is generated”
- Data generation process
  - Assume there is a latent code  $z$  that guides the generation of  $x$
  - Assume  $z$  follows a distribution  $p(z)$ , called the prior distribution

Sample $z_i$ from $p(z)$	We have dataset $D = \{x_i\}$
Sample $x_i$ from $p(x z_i)$	How to learn $p(x z)$ ?
Obtain dataset $D = \{x_i\}$	

我们一般假设 **$p(z)$ 是一个简单且显然的分布**，例如高斯分布。求解这个分布的方式不难。数据集我们也已知，这代表着我们也知道 $p(x)$ 的分布信息。但是真正的难点是在于如何求解 $p(x | z)$ ，即在已知 $z$ 的情况下如何求解出 $x$ 的分布信息。（这就是decoder的工作流程）

我们很显然地采用**最大似然估计**，即找到一个分布参数 $\theta_1$ ，使得其能够最大化 $p(x)$ 的likelihood，即最能拟合这个分布。

$$p(x, z) = p(x|z)p(z) \quad p(x) = \int p(x, z)dz, \text{ 因此 } p(x) = \int p(x|z)p(z)dz$$

$$P(x) = \int p_{\theta_1}(x | z) p(z) dz \implies \max \sum \log P(x_i)$$

- 注：我们认为 $p(x|z)$ 是好算的，因为这就是VAE的decoder的计算结果，但是之所以不能使用上面的公式直接求出 $P(x)$ 的原因是 $p(z)$ 是一个复杂的多维向量，对其全部积分是难以求出的。蒙特卡洛也是有错误的，为此我们才需要转换思路，使用下面的方式求出解。
- 我们通过一些先验的假设解决这个问题。首先的假设就是：给定一个固定的 $z$ ，只有一小部分 $X$ 中的点有非零的 $p(x|z)$ ，这个对应于网购的时候，选定一个标签，只有一小部分的商品是符合这个标签的。因此我们设计了一个分布/函数 $q_{\theta_2}(z|x)$ ，这个分布叫做**Variational Posterior Distribution**，即给定一个数据集，找到其中满足特定标签的样本。

- Maximum likelihood method

$$P(x) = \int p_{\theta_1}(x|z)p(z)dz \longrightarrow \text{maximize } \sum \log P(x_i)$$

$$\log P(x) = \int_z q(z|x) \log P(x) dz = \int_z q(z|x) \log \frac{p(x,z)}{p(z|x)} dz$$

$$= \int_z q(z|x) \log \frac{p(x,z)}{q(z|x)p(z|x)} dz = \int_z q(z|x) \log \frac{p(x,z)}{q(z|x)} dz + \int_z q(z|x) \log \frac{q(z|x)}{p(z|x)} dz$$

- 第一个等号 $\log P(x) = \int_z q(z|x) \log P(x) dz$ 是恒等式( $\int_z q(z|x) dz = 1$ )，提取出log部分之后里面就是一个概论分布的求积分，结果是1。利用贝叶斯公式我们可以将logP(x)拆分，上下同乘一个值，之后拆分。其中紫色部分是KL散度 ( $KL(p||q) = \int p(x) \log \frac{p(x)}{q(x)} dx$ )，即 $KL(q(z|x)||p(z|x))$ ，其一定非负 (Jensen不等式)。前面的绿色部分被称为**Evidence Lower Bound(ELBO)**，这个里面的lower bound是likelihood的lower bound，即似然的下限。通过控制这个值，我们能够近似出似然。把非负数的KL divergence去除掉，因此可以得出

$$\log P(x) \geq \int_z q(z|x) \log \frac{p(x,z)}{q(z|x)} dz$$

- 我们接着把ELBO中的联合概率拆分为条件概率： $p(x,z) = p(z)p(x|z)$

$$\log P(x) \geq \int_z q(z|x) \log p(x|z) dz + \int_z q(z|x) \log \frac{p(z)}{q(z|x)} dz$$

- 这两项是有意义的，第一项是一个期望，其含义和reconstruction loss是一样的，都是先给定x，计算出采样出 $q(z|x)$ ，之后再在 $\log p(x|z)$ 中进行采样还原出原函数，这个值其实就是**reconstruction loss**。第二项灰色的就是KL散度，其含义就是**regularization loss**。

$$\log P(x) \geq \int_z q(z|x) \log p(x|z) dz + \int_z q(z|x) \log \frac{p(z)}{q(z|x)} dz$$

$$E_{z \sim q(z|x)} [\log p(x|z)]$$

Function  $q$  is the encoder and function  $p$  is the decoder, this term is the reconstruction performance

- 注：蓝色部分的含义是，先通过encoder编码，在其中采样一个 $z$ ，然后decoder给出在这个 $z$ 下重建出x的对数概率。这也就是重建x的概率分布，即reconstruction loss。灰色部分是 $-KL(q(z|x)||p(z))$ ，一般我们都认为先验 $p(z)$ 是标准正态分布，在这种情况下即是将编码器生成的概率拉向标准正态分布。因此我们最小化KL散度，最大化重建似然，就可以最大化P(x)实现我们的目标。

### Problems in VAE:

- 不能更深、隐层的维度对结果的影响极大、不能做深度估计、只能是系统的一部分，不能是独立的系统。

### DAE(Denoising Auto Encoder)

- VAE是在representation阶段增加noise（通过encode出 $\mu$ 和 $\sigma$ 之后进行采样，产生noise），DAE则是在input部分直接增加noise。
- DAE相比VAE的优势在于模型规模可以更大，网络结构可以更深：VAE的input是固定的，如果模型一旦过大，其会形成一一映射，进而没有损失无法训练，而DAE在输入之中加入了扰动，很难形成一一映射。

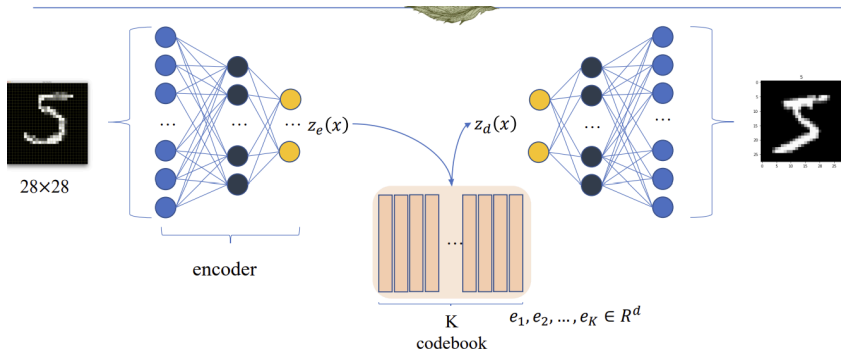
- Denoising Autoencoder
  - Input:  $x$
  - $x = AddNoise(x)$
  - Output:  $x' = f(x, \theta_{model})$

- Reconstruction loss

- masked autoencoder把输入信息masked当作扰动，之后使用一个极大的encoder来编码图片，一个极小的decoder来解码。DAE不被认为是传统意义的生成模型，其能实现的是50%-100%的补全，而不是从0%开始的直接补全。

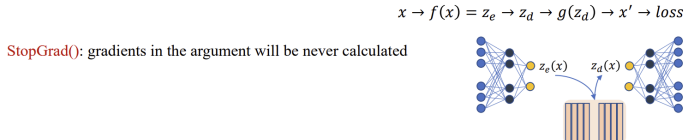
### VQVAE(Vector-quantized VAE)

- 目标：将一张图片处理成为一系列的word tokens，



- VQVAE的核心在于，在encoder得到一个向量之后，将得到的向量和codebook字典里的向量找最近的一个，最近的一个就是查到的向量，将这个向量作为 $z_d(x)$ ，传入decoder希望能还原出原图片。

- If we use  $\|x - g(z_d)\|$  as the loss, encoder parameters cannot be updated
- If we use  $\|x - g(z_e)\|$  as the loss. The objective has changed and the gradient is not correct. Wrong grad for all parameters
- Solution: set  $z_d = z_e + \text{StopGrad}(z_d - z_e)$

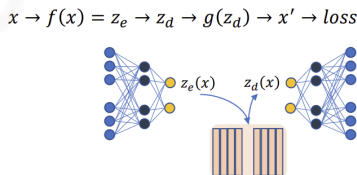


- 在反向传播的时候会遇到与VAE一样的问题，无法将反传的梯度传回起始位置。第一种方式 ( $\|x - x'\|$ ) 是不管前面的encoder参数，只反传后面的decoder的参数，这样是有问题的，更新不到位。第二种方式 ( $\|x - f(z_e)\|$ ) 是不管argmax的查表操作，直接反传，这样的方式也不好，其优化目标改成了使得输入和编码后的结果尽量接近，更新的参数甚至是错误的。
- 第三种方式是使用**StopGrad**，其核心目的是其中的参数的梯度不会被更新， $z_d = z_e + \text{StopGrad}(z_d - z_e)$ ，**Loss term**是 $\|x - f(z_e + \text{StopGrad}(z_d - z_e))\|$ ，此时正向传播是正确的，反向传播可以正常更新decoder的参数。**Codebook loss**是 $\|z_e - \text{StopGrad}(z_d)\| + \beta \|z_d - \text{StopGrad}(z_e)\|$ ，二者损失相加即是最终的更新的损失函数值。
- Loss term负责更新encoder和decoder，其正向传播是正确的，但是反向传播的时候认为  $z_d = z_e$ ，即认为encoder的输出就是decoder的输入，这样可以正确更新encoder和decoder。Codebook Loss像一根橡皮筋， $\|z_e - \text{StopGrad}(z_d)\|$ 这一项保持 $z_d$ 不变，使得 $z_e$ 拉向 $z_d$ ，因此目的是把encoder的输出拉向codebook； $\|z_d - \text{StopGrad}(z_e)\|$ 这一项保持 $z_e$ 不变，因此目的是把codebook的值向encoder的输出靠近。这样即可对齐。

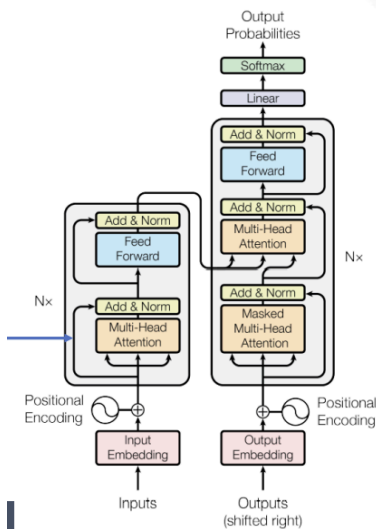
- We hope the codebook is meaningful
- Codebook loss

$$\|z_e - \text{StopGrad}(z_d)\| + \beta \|z_d - \text{StopGrad}(z_e)\|$$

- VQVAE loss = reconstruction loss + codebook loss



## Transformer Model



- Attention的核心目的是获取文本关联和词关联，即不同词语之间的联系。经过注意力层之后我们希望结果包含了更多的surrounding information。
- 哈希表(Hash Table)是一种注意力机制，当query进入的时候，与所有的key进行搜索，找到最符合query的 $k^*$ 输出其对应的value。Attention可以被理解成一种软性的Hash table，即给出一个Query之后，每个Key都要与其进行匹配度计算，将这些相似度进行线性加权得到结果的得分。



- Each word is represented as a vector  $x_i$
- $q_i = x_i W^Q$  ,  $k_i = x_i W^K$  ,  $v_i = x_i W^V$

word	Q	K	V	Score	Softmax
Natural	$q_1$	$k_1$	$v_1$	$q_1.k_1/\sqrt{d_k}$	$x_{11}$
Language		$k_2$	$v_2$	$q_1.k_2/\sqrt{d_k}$	$x_{12}$
Processing		$k_3$	$v_3$	$q_1.k_3/\sqrt{d_k}$	$x_{13}$

Assuming  $q_1, k_1$  follow standard normal distribution, we have

$$E[q_1.k_1] = E\left[\sum_{i=1}^{d_k} q_{1,i}k_{1,i}\right] = \sum_{i=1}^{d_k} E[q_{1,i}k_{1,i}] = \sum_{i=1}^{d_k} E[q_{1,i}]E[k_{1,i}] = 0$$

Assuming  $q_1, k_1$  follow standard normal distribution, we have

$$Var[q_1.k_1] = E[q_1.k_1]^2 = E\left[\left(\sum_{i=1}^{d_k} q_{1,i}k_{1,i}\right)^2\right] = \sum_{i=1}^{d_k} E[q_{1,i}^2k_{1,i}^2] = \sum_{i=1}^{d_k} E[q_{1,i}^2]E[k_{1,i}^2] = d_k$$

- 这个图有一点需要格外注意：输入是**single query**，与全部的Key进行作用。为什么分母要除以 $\sqrt{d_K}$ ？
- $\sqrt{d_k}$ 的含义如下：** 我们假设 $q_i$ 和 $k_i$ 都服从标准正态分布，我们可以设 $q, k$  的维度均为 $d_K$ ，因此我们有 $E[q_1 * k_1] = 0$ ，期望是0。在计算方差的时候，还是假设二者都是从标准正态分布中采样得到，二者乘积的方差是 $d_k$ 。即如果不normalize的话，对于一个2048维的向量，其采样出来的分布是一个 $N(0, 2048)$ ，这样的分布softmax之后值会极其接近1或者1，因此在初始化为正态分布之后，模型极其难以学习到信息。在normalize之后模型初始化的分布接近标准正态分布，这样更有利于模型自主学习出构建QKV的方式。因此， $d_k$ 代表的是embedding之后词向量的维度。

### The compact form of attention

- Input of attention:  $X = [x_1, x_2, \dots, x_n] \in R^{n \times d}$ 
    - For the first layer attention,  $X$  consists of word embeddings
    - For intermediate layer attention,  $X$  is the output of the previous layer
  - Obtain Query, Key, Value matrices :  $Q = XW^Q \in R^{n \times d_k}, K = XW^K, V = XW^V$
  - Calculate correlation matrix:  $E = QK^T/\sqrt{d}$
  - Normalize correlations:  $A = softmax(E)$
  - Linear combination: output =  $AV$
- parallelizable matrix computations

- Attention可以通过**矩阵并行**的方式进行，一次输入一个大矩阵直接计算。从原理上而言，attention是一个非线性变换，其核心的目的就是收集外加组织信息。因此在经过transformer之后还需要一层FFN(前馈神经网络)。在Add&Norm区域之中使用residual残差网络进行相加，即新的输出是旧的输出外加上旧的结果经过非线性变换之后的结果。Norm代表的是Layernorm归一化。

### Residual connection

Previous (general) network architecture

$$\begin{aligned} x_1 &= f_0(x) \\ x_{l+1} &= f_l(x_l) \\ y &= f_L(x_L) \end{aligned}$$

Residual network architecture

$$\begin{aligned} x_1 &= x + f_0(x) \\ x_{l+1} &= x_l + f_l(x_l) \\ y &= f_L(x_L) \end{aligned}$$

- 查阅pytorch文档，这几种normalization的官方文档是一模一样的。(batchnorm, groupnorm, instancenorm, layernorm)。但是正如下图所示，不同normalization方式的不同是取决于归一化维度的不同。
- Layernorm是对embedding dimension进行的归一化。如果我们认为张量 $T[i][j][k]$ 代表的含义是第i个句子里的第j个token的第k个元素。这个理解方式不同于多模态学习中提及的NCHW这样的CNN的表示法，需要单独理解。

### Layer normalization

- Variables are formulated by Tensors
  - Inputs:  $x$
  - Intermediate variables:  $z_l$
- Tensor shape: batch\_size \* sequence\_length \* embed\_dim
  - $T[i][j][k]$** : the  **$k$ -th element** in the embedding of **the  $j$ -th token** in the  **$i$ -th sentence**

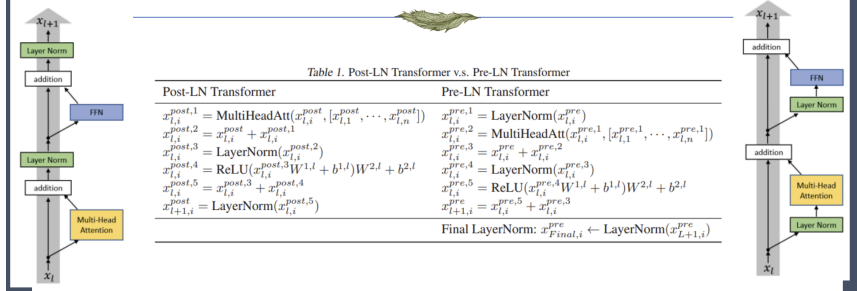
- 那么Layernorm是将embedding dimension的信息全部normalize。即 $T[i][j][:]$ ，而batchnorm则是前两个维度的normalize，即 $T[:, :, k]$ 。
- layernorm和batchnorm的不同点在于：**layernorm是对一个batch里一个例子的所有维度进行归一化，而batchnorm是对一个batch里所有例子的一个维度进行归一化。正如下面的例子展示的一样，batch=2, 句子长度是3，embedding=4，在这种情况下layernorm的做法是一个token计算一次均值和方差，最后结果维度是 $2 * 3$ ；batchnorm的做法是(1,5,9,2,1,0)计算一次均值和方差，最终的生成结果是4。

假设我们有一个 **batch=2**, **句子长度=3**, **embedding 维度=4** 的输入 (非常小, 方便算) :

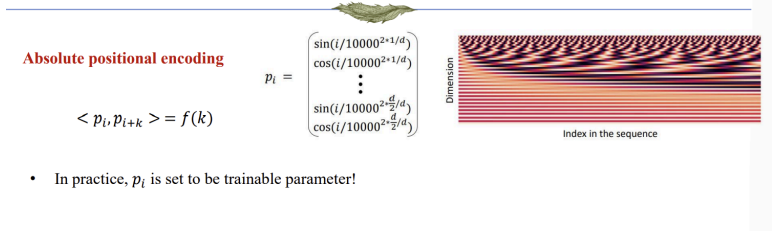
```
less
句子1:
token1: [1, 2, 3, 4]
token2: [5, 6, 7, 8]
token3: [9, 10, 11, 12]

句子2:
token1: [2, 4, 6, 8]
token2: [1, 3, 5, 7]
token3: [0, -1, -2, -3]
```

## Two different structures



transformer有两种不同的范式，即**Post-LN Transformer**和**Pre-LN Transformer**，这两种范式的不同是取决于LayerNorm所放置的位置不同，左侧的Post-LN范式确保任何输入都至少经过一个非线性变换才能走到终点；右侧的Pre-LN范式则有些输入不用经过非线性变换也能到达终点。一开始人们使用的是左侧的Post-LN Transformer，但是这个模型训练起来难度极大，不同的初始化方式会对结果有极大的影响，包括学习率这样的参数。现在人们常用Pre-LN Transformer。



- 从梯度本身可以进行理解，Pre-LN方式的梯度比较稳定，但是Post-LN方式的梯度不太稳定，这可能就是最终的原因。
- 位置编码positional encoding**：绝对位置编码、相对位置编码、旋转位置编码。
- Absolute positional encoding**：
- 实现方式是直接把词的位置 $p_i$ 加到词编码后的结果之后 $x_i$ 之中，希望大模型能够直接通过词的编码信息分析出不同词所处位置的不同。绝对位置编码里面一般使用类似正弦余弦函数的方式来编码位置。这个方式有问题，具体而言在经过矩阵乘法的变换之后原本有位置描述的不同向量变换后就不再具有位置关系。而且无法运用在长序列之中，也无法拓展到更多不同模态。

## Problems

- The unlearnable absolute position encoding generalizes poorly on longer sequences.  
 $\langle p_i, p_{i+k} \rangle = f(k)$   
 $\langle W^Q p_i, W^K p_{i+k} \rangle \neq f(k)$
- Learnable absolute position encoding cannot be used on longer sequences.
- Unable to extend to more complex scenarios (such as images and graph structures)

- Relative positional encoding**：
- 由于绝对位置编码具有各种问题，包括多模态任务上直接使用绝对位置编码是不合理的。T5模型首先使用了**加性相对位置编码**，即构造了一个B矩阵，其斜对角线的值是完全一样的，即  
 $A(x) = \text{softmax}(XW_Q(XW_K)^T + B)$ 。
- 除了加性的相对位置编码，还有**乘法的相对位置编码**，即使用旋转矩阵R作用在QK矩阵的计算之中。  
 $A(x) = \text{softmax}(XW_QRXW_K)$ ，R矩阵是一个旋转矩阵，RoPE就是这样的一个矩阵，其运用在Llama之中，是开源大模型的最主流的方式。

$$A_{\text{RPE}}^h(X) = \text{softmax} \left( XW_Q^h (XW_K^h)^T + B \right)$$

$\text{softmax} \left( \frac{\begin{bmatrix} \text{Q} \end{bmatrix} \times \begin{bmatrix} \text{K}^T \end{bmatrix}}{\sqrt{d_k}} + \begin{bmatrix} \text{B} \end{bmatrix} \right) \begin{bmatrix} \text{V} \end{bmatrix}$

Examples: Rotary Position Embedding (RoPE)

$$A_{RoPE}^h(X) = \text{softmax}(XW_Q^hR(XW_K^h))$$
$$R_{ij} = f(i-j) = \begin{pmatrix} \cos(i-j)\theta_1 & -\sin(i-j)\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin(i-j)\theta_1 & \cos(i-j)\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos(i-j)\theta_2 & -\sin(i-j)\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin(i-j)\theta_2 & \cos(i-j)\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos(i-j)\theta_{d/2} & -\sin(i-j)\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin(i-j)\theta_{d/2} & \cos(i-j)\theta_{d/2} \end{pmatrix}$$

$R$  is the rotary matrix with pre-defined parameters  $\theta_m = 10000^{-2(m-1)/d}, m \in [1, 2, \dots, d/2]$

Autoregressive Model

General setting

Goal: model  $P(X = x)$

↓

$P(X_1 = x_1, \dots, X_d = x_d)$

↓ Conditional probability (chain rule)

$P_\theta(X_1 = x_1, \dots, X_d = x_d)$

$= P_\theta(X_1 = x_1)P_\theta(X_2 = x_2, \dots, X_d = x_d | X_1 = x_1)$

$= P_\theta(X_1 = x_1)P_\theta(X_2 = x_2 | X_1 = x_1)P_\theta(X_3 = x_3, \dots, X_d = x_d | X_1 = x_1, X_2 = x_2)$

$= P_\theta(X_1 = x_1)P_\theta(X_2 = x_2 | X_1 = x_1)P_\theta(X_3 = x_3 | X_1 = x_1, X_2 = x_2) \dots P_\theta(X_d = x_d | X_1 = x_1, \dots, X_{d-1} = x_{d-1})$

**Autoregressive model**

If I have a parametric model with parameter  $\theta$  which can estimate the following thing

$P_\theta(X_i = x_i | X_1 = x_1, \dots, X_{i-1} = x_{i-1})$

- $P(X = x) = P = P_\theta(X_1 = x_1)P_\theta(X_2 = x_2 | X_1 = x_1) \dots P_\theta(X_d = x_d | X_1 = x_1, \dots, X_{d-1} = x_{d-1})$
- 我们希望一个模型能够在已知 $x_1, \dots, x_{i-1}$ 的基础上预测 $x_i$ 。
- 自回归模型使用条件概率来计算联合概率，MLP中使用left-to-right factorization方式生成，而也存在right-to-left factorization。不同的生成方式都是合理的，我们主要关注**从左到右的生成**，这种模型被称之为**language model**。
- 语言模型的定义是：给一部分的文本，可以预测出下一次词。古早的语言模型是**N-gram架构**，模型可以从指定的词里进行选择预测下一个词是什么，为了减少计算量，其在生成第*i*个词的时候只使用与其最近的*k*个词的条件概率来推断。N-gram模型的假设是Markov Assumption，即 $x_i$ 不由距离较远的tokens决定。 $x_i$ 依靠最近的一些信息进行更新。

(oldest non-parametric) N-gram language model

$P(X_i = x_i | X_1 = x_1, \dots, X_{i-1} = x_{i-1})$

↓

$P(X_i = x_i | X_{i-k} = x_{i-k}, \dots, X_{i-1} = x_{i-1})$

↓

$\frac{P(X_i = x_i, X_{i-k} = x_{i-k}, \dots, X_{i-1} = x_{i-1})}{P(X_{i-k} = x_{i-k}, \dots, X_{i-1} = x_{i-1})}$

Markov assumption: assume  $x_i$  doesn't depends on far away tokens

Definition of conditional probability

- Estimate from data:
- $P(X_{i-k} = x_{i-k}, \dots, X_{i-1} = x_{i-1}) \approx \frac{\text{number of } (x_{i-k}, \dots, x_{i-1}) \text{ in the training data}}{\text{number of } k\text{-gram in the training data}}$
- 这个算法的一大问题是空间消耗大，每多考虑一个词都是将搜索空间乘上sentence corpus的信息。下面的图是一个例子，即n=4的时候只能考虑四个词的信息，students opened their出现了100次，其中后面接books的情况出现了70次，laptop出现了30次。因此最终会生成students opened their books。（注意，N-gram=4代表当前词的生成由自己和前面的三个词共同决定，不是由自己和前面的四个词一起决定）

N-gram language model: example

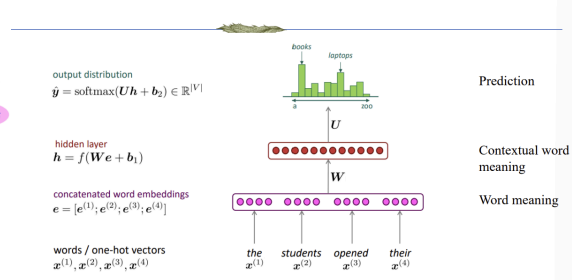
the students opened their

books laptops minds exams

In the corpus, "students opened their" appears 100 times.

- "students opened their books" appears 70 times  
->  $P(\text{books} | \text{students opened their}) = 0.7$
- "students opened their laptops" appears 30 times  
->  $P(\text{laptops} | \text{students opened their}) = 0.3$

A fixed-window neural language model

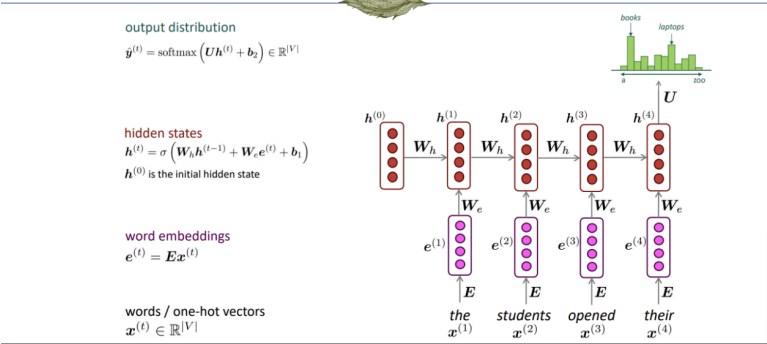


- N-gram模型的不可能三角：**Sparsity, Memory和Accuracy**。如果N增加，则文本准确度ACC提升，更容易生成有语义含义的信息，但是会更耗散内存消耗Memory(计算查找出现次数的难度增加)。同时n增大之后与之匹配的词段数减少，因此estimation会不太准确(Sparsity增加)。如果N减少，使用的语义信息减少，但是内存需要也减少，能够统计学意义上更好的估计，但是预测准确度本身ACC也会下降。
- 为了进一步提高，Bengio提出了**fixed-window neural language model**。使用一个神经网络来估计条件概率。将输入句子的词先过embedding，之后concatenate到一起，之后经过一个线性变换生成概率，取其中概率最高

的一个词作为输出。这个方式的优势是没有记忆存储的问题，没有稀疏性的问题，但是同时对N的设置使之灵活性不足，人们必须指定一个prefixed window才能开始实现。而且在当时表现极其差。

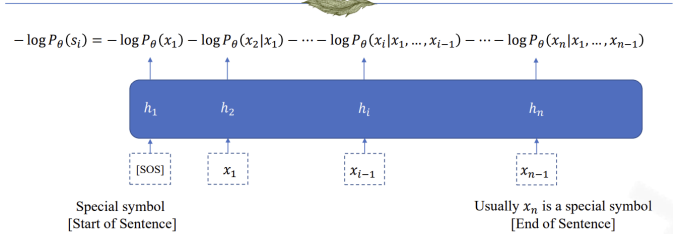
- **RNN-based neural language model**：
- 这个模型能够对所有的前向的文字信息全部考虑，不局限于fixed-window的窗口设计。用一个模块能够实现对过去所有信息的记忆和存储，这个模型需要迭代，因为句子不断有词进入，需要根据现有的情况不断改变，这个模块就是hidden states。将过去的信息和现在的信息融合变换之后得到了新的hidden state，这个信息结合了过去的信息和当前的信息。RNN方法非线性的能力更强，且处理长句子的效果佳。

### RNN-based neural language model



- RNN-based的方法有优势也有劣势，其即训练速度极其缓慢，因为其无法有效并行计算，每个位置的token都需要独立进行估计逐个进行，而且优化难度较高，有梯度爆炸和梯度消失的情况。

### (RNN) LM training objective



- Teacher forcing和next token prediction，这两个概念的含义是给出前面的token的信息，以这些信息为groundtruth，生成下一个token的信息，二者含义是一致的。
- 正是由于RNN结构的这些问题，我们希望使用transformer的结构，但是同样满足自回归的性质，即后一个token的生成只跟前面的token信息有关，而与其之后的无关，因此需要attention\_mask，即下三角矩阵的部分是0，上三角的部分是-inf。通过-inf的方式让不让想模型看到的部分在经过softmax之后的结果是0，即后面的信息对结果是无影响的。这也说明了transformer架构速度比RNN更快的原因，即有大量的并行进行，而不是串行。

Given a sentence  $s$  of length  $n$  (from the test dataset, unseen during training)

$$nLL(s; \theta) = -\log P_{\theta}(s)$$

Per-word negative log-likelihood

$$WnLL(s; \theta) = -\frac{1}{n} \log P_{\theta}(s)$$

Perplexity

$$ppl(s; \theta) = \exp(WnLL(s; \theta))$$

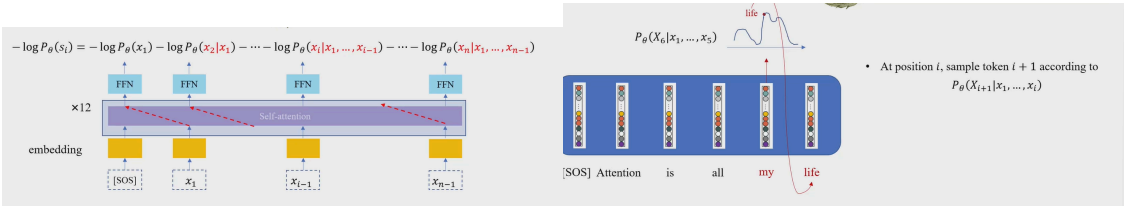
Other metric (generation): BLEU, ROUGE... (learn from NLP courses)

- 现在的语言模型使用**混合精度训练(Mixed Precision Training)**，一般使用fp32, fp16, bf16这三种精度混合进行。矩阵的乘法是一个相对粗糙的运算，因此此时存储的参数多使用较为粗糙的。Layernorm的计算是需要相对精细的，因为在计算的时候可能出现出0的情况。现在的语言模型也使用更大**batch training**方式进行，即使得优化和收敛速度更快，一般来说一个batch收集1M的tokens。有时候也使用**Gradient accumulation**的方式进行，即一个mini-batch前向计算之后反向不计算loss，而是存着梯度，直到token到了指定的个数才计算loss，这种方式能够尽量确保更新参数方式是正确的。为了稳定训练，一般会有warmup和clip和small dropout的方式。
- Evaluation可以使用**Perplexity(ppl)**，不确定性，即

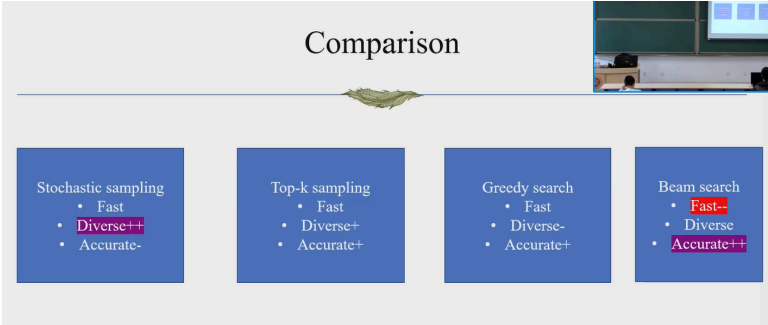
$$nLL(s; \theta) = -\log P_{\theta}(s)$$

- 这个方式不太合理，因为随着句子长度的增加perplexity天然就会更高，因此科学家一般将nLL除以n，即句子长度，最后将这个值取exponential即是结果ppl。  $ppl(s; \theta) = \exp(-\frac{1}{n} \log P_{\theta}(s))$ 。
- 如果没有添加self-attention的mask，则在训练数据上loss的值极其低，这是因为模型只学会了copy，学到的做法是用i+1位置的值放置在i位置的输出上，而完全没有学会自己去预测词。此时**evaluation上的perplexity同样是极其低的，并没有不佳的情况出现！**这是因为这个bug在验证集上同样适用，即copy。只有模型在场景中使用的時候，此时才能看出模型生成是有错误的。





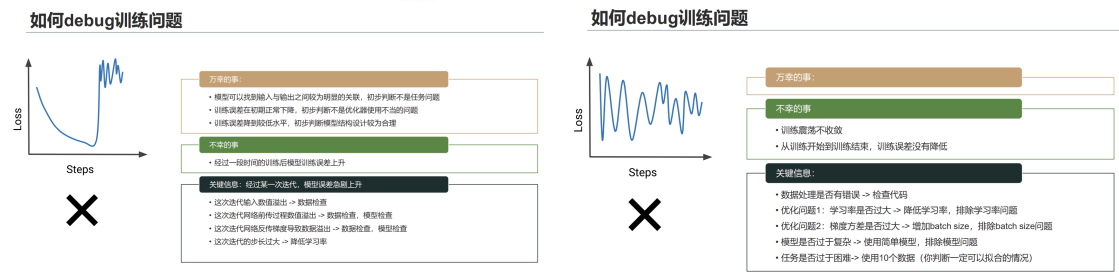
- 对于大模型，在训练之后也有不同的inference方式生成结果。
- 有一种inference方法是stochastic sampling，即通过已知前面的信息来推断后一个词的信息，通过对分布中随机sample出一个token信息。这种方式inference的结果是不太好的，因此人们常用**top-k sample**，即从每一步的概率分布之中选择出**前k个**概率最大的词，在这k个词中uniform均匀分布采样出结果。**top-p sample**的意思是在所有的概率分布之中选择出**概率值大于p**的词，将这些词均匀分布采样出结果。top-k sample可以限制为greedy search之类的方法，即top-1 sample。



- 对比transformer和RNN的**Computational Cost**：对于attention的**训练过程**，第一步是生成QKV矩阵，此时我们不妨设X的维度是 $n * d$ ，Wk, Wq, Wv矩阵的维度是 $d * d'$ ，因此生成的QKV矩阵的维度是 $n * d'$ ，因此第一步的时间复杂度是 $O(n * d^2)$ 。第二步是计算Attention的值，此时的时间复杂度是 $O(n^2 * d)$ ，求完softmax之后用softmax归一化后的值与V矩阵相乘的结果也是 $O(n)$ ，因此attention的花销总计在 $O(n^2)$ 左右( $n \gg d$ )。对于RNN的训练过程，其需要的computation cost是 $O(n)$ 的，但是在具体的时间花销上RNN时间花费更长，这是由于其无法并行，计算速度慢。
- 对于一个RNN模型来说，inference的时间复杂度是 $O(n)$ ，transformer的inference的时间复杂度是 $O(n^2)$ ，即每一位都是 $O(1, 2, 3, \dots, n)$ ，叠加的结果是 $O(n^2)$ 。
- 因此，结论是对于**Computational Cost**来说，**transformer不论训练还是推理都高于RNN**，transformer是 $O(n^2)$ ，而RNN是 $O(n)$ ，但是从**Training Efficiency**来讲，transformer比RNN更为efficient，其矩阵的并行化程度更高。对于**Inference Efficiency**来说，RNN显著高于transformer，因为inference的时候无法并行化计算，因此transformer的 $O(n^2)$ 劣势一览无余(Atten(qt, k1, k2, ..., kt-1))，而RNN的 $O(n)$ 更快。
- 题外话：KV Cache的出现让attention的inference时间复杂度降低到 $O(n)$ ，其核心思路是缓存已经计算好的key value information。之后又出现了**Multi-Head Latent Attention(MLA)** 等技术，其核心是将Key/Value投影到一个低维度的latent向量，只缓存压缩状态，然后在真正需要的时候再恢复展开，这样可以极大减少缓存的占用内存，使得推理中长上下文更可行。

## Debugging

- 找一份靠谱的代码，针对自己的问题做特定的修改，包括修改模型结构，修改目标函数，接入新的下游任务，设计新的优化算法等。但是对于深度学习而言，可解释性比较差，因此很难进行充分的debug。非线性模型，无优化方法可以收敛到全局最优。

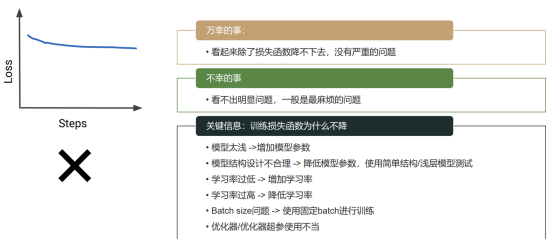


- 上左图展示的模型训练过程中**大部分情况是合理的**，其在初步训练过程中表现良好，loss的收敛很正常，但是突然遇到了一个spike。这个问题是大模型中训练过程中最容易遇到的。形成上述曲线的主要原因如下：1. 输入数值溢出；2. 前向/反向传播过程中有数值溢出；3. SGD优化中的minibatch可能存在统计不足的情况，在学习率较大的时候会反向移动进而使得loss极大。
- 上右图则是**极其坏**的结果，训练震荡完全未收敛，这时候可能的错误有：1. 数据处理是否有错误，即x和y的对应关系错误。2. 学习率是否严重过大，即在波峰和波谷之间一致震荡。3. 梯度的方差是否严重过大，即batchsize过小，导致梯度极其不稳定。4. 模型是否过于复杂，导致模型无法拟合这样的问题。5. 模型的任务是不是过难，考虑使用少量数据进行拟合测试。

如何debug训练问题



如何debug训练问题



- 上左图代表模型训练过程中**基本正常**, 但是训练一段时间之后训练误差不断抖动, 甚至有周期现象出现。这种周期现象可能来自于以下情况。1. 学习率过大, 在一个loss的山谷内不断弹跳, 但是这样的情形极难出现, 需要极其精巧的数据。2. 数据没有随机打散, 即每个epoch里的数据类别是一致的, 这样训练时每个batch只有一类数据, 类与类之间会有差异性。
- 上右图代表模型训练过程正常, 但是模型的表现过差, 这个问题比较复杂, 可能由于模型太浅, 架构不合理, 学习率过低, 学习率过高, batchsize不正确, 优化器和优化器超参数使用不当。
- 上述问题解决之后, 模型的training loss还比较理想, 但是test loss有时候并不正常。

如何debug测试中的问题



如何debug测试中的问题



- 上左图展示的是泛化能力不足导致的问题, 可以降低模型参数, 加入合理的正则项, label smoothing, 降低batchsize (提高随机性), 并且使用预训练模型。上右图展示的情况基本就来自于测试代码的问题, 或者数据处理的问题。

如何debug测试中的问题



- 上图展示的情况是测试的损失函数值极其不稳定, 这样难以选择出效果最好的模型参数。下游任务微调的时候可能会遇到这样的问题, 其实实际上就是测试集的数据量太小。

## GANs (Generative Adversarial Networks)

- 假设随机变量X服从一个复杂的分布Q, 我们应该如何从Q中生成数据?
- 如果我们认为F(x)是Q的累计概率密度函数CDF,  $F(x) = P(X \leq x)$ , 我们可以从一个uniform distribution中采样出Z, 因此 $X = F^{-1}(Z)$ 服从Q分布。( $P(X < t) = P(F^{-1}(Z) < t) = P(Z < F(t)) = F(t)$ ), 这个理论证明**任何**的复杂分布都可以被简单的分布所映射产生。
- 显式模型(explicit model)和隐式模型(implicit model):
- 对于**显式模型**, 模型参数是可知的, 而且知道每一个输入所带来的输出概率, 我们可以通过最大似然的方式来训练模型。自回归模型就是典型的显式模型。对于**隐式模型**, 模型参数是已知的, 但是我们不知道一个输入所带来的输出概率, 也无法使用最大似然的方式来训练模型。GAN是隐式模型, 但是我们应该如何评估生成数据和真实数据之间的差距?
- 对于单点距离, 我们可以使用Euclidean Distance的方式进行衡量, 但是对于分布之间的距离, 我们有多种方式定义距离, 比如说 $f - divergence$ 方法。
- Jensen's inequality**:
- 如果 $f: R \rightarrow R$ 是凸函数, X为随机变量, 则有 $f(E(X)) \leq E(f(X))$ 。直观理解是凸函数会对于离中心远的值惩罚程度更大。因此先平均再求函数值的值会较小。
- 改成积分的测度形式, 即是:

$$f\left(\int x d\mu\right) \leq \int f(x) d\mu$$

- f-divergence**:
- 我们认为第一个分布是P(x), 密度函数是p(x), 第二个分布是Q(x), 密度函数是q(x), 我们找到一个映射f, 满足将 $[0, +\infty] \rightarrow [-\infty, +\infty]$ , 而且满足凸、非负性等性质。我们可以以此定义f-divergence:

$$D_f(P||Q) = \int f(\frac{p(x)}{q(x)})q(x)dx = E_{x \sim Q}[f(\frac{p(x)}{q(x)})]$$

，这个距离描述方式的合理性很好验证。由于f是凸函数，其通过Jensen不等式可以推知：

$$D_f(P||Q) = \int f(\frac{p(x)}{q(x)})q(x)dx \geq f(\int (\frac{p(x)}{q(x)})q(x))dx = f(1) = 0$$

，即证明这个距离是非负的。

- 如果我们认为 $f(t) = t \log t$ ，则我们称距离为 $KL - divergence$ ， $D_f(P||Q) = \int p(x) \log \frac{p(x)}{q(x)} dx$ 。
- 如果我们认为 $f(t) = \frac{1}{2}|t - 1|$ 。则 $D_f(P||Q) = \frac{1}{2} \int |p(x) - q(x)| dx$ 这个被称之为**全变分距离**。
- 如果我们认为 $f(t) = -(t + 1) \log \frac{t+1}{2} + t \log t$ ，我们有 $D_f(P||Q) = KL(P||M) + KL(Q||M)$ ，其中 $M = \frac{P+Q}{2}$ ，这被称之为**JS - Divergence (Jenson - Shannon Divergence)**
- 在GAN的论文中，研究员提出的优化函数是JS-Divergence，认为 $P \sim P^*$ ， $Q \sim P_\theta$ ，其中 $P^*$ 是数据分布， $P_\theta$ 是模型生成的分布。目的即是使得模型生成的分布和数据的真实分布越接近越好。

Learn generative model with (minimizing) Jensen-Shannon-divergence

Minimize  $KL(P_*||\frac{P_* + P_\theta}{2}) + KL(P_\theta||\frac{P_* + P_\theta}{2})$   
 $\Downarrow$   
 Minimize  $E_{x \sim P_*}[\log \frac{2P_*(x)}{P_*(x) + P_\theta(x)}] + E_{x \sim P_\theta}[\log \frac{2P_\theta(x)}{P_*(x) + P_\theta(x)}]$   
 $\Downarrow$   
 Minimize  $E_{x \sim P_*}[\log \frac{P_*(x)}{P_*(x) + P_\theta(x)}] + E_{x \sim P_\theta}[\log \frac{P_\theta(x)}{P_*(x) + P_\theta(x)}]$

- 但是这个式子本身是无法被优化的，我们不知道ground truth的分布 $P_*(x)$ ，也不知道 $P_\theta(x)$ ，对于

$$Minimize \quad E_{x \sim P_*}[\log \frac{P_*(x)}{P_*(x) + P_\theta(x)}] + E_{x \sim P_\theta}[\log \frac{P_\theta(x)}{P_*(x) + P_\theta(x)}]$$

- 我们确实无法计算log后面的那两项值，但是我们可以粗浅的理解一下：如果x极其不可能从真实分布中采样，则 $P_*(x)$ 会接近0，如果x极其不可能从模型参数的分布中采样，则 $P_\theta(x)$ 会接近0，为此我们可以使用一个**binary classifier**： $D_\theta(x)$  来判定x是从真实分布还是模型分布中采样而来，即 $D_\theta(x) = \frac{P_*(x)}{P_*(x) + P_\theta(x)}$
- 通过这种替换，我们把公式改为

$$Minimize \quad E_{x \sim P_*}[\log D_\theta(x)] + E_{x \sim P_\theta}[\log(1 - D_\theta(x))]$$

，之后我们考虑再化简一下，x服从 $P_\theta$ 其实本质上就相当于z服从标准正态， $x = G_\theta(z)$ ，因此

$$Minimize \quad E_{x \sim P_*}[\log D_\theta(x)] + E_{z \sim N(0,I)}[\log(1 - D_\theta(G_\theta(z)))]$$

Learn generative model with (minimizing) Jensen-Shannon-divergence

Minimize  $E_{x \sim P_*}[\log D_{\theta'}(x)] + E_{z \sim N(0,I)}[\log(1 - D_{\theta'}(G_\theta(z)))]$

Training of  $D_{\theta'}$

- Denote the generative model as  $G_\theta(z)$ , which can generate sample from noise  $z$ 
  - Sample K data from the real dataset [positive,  $S_{pos}$ ]
  - Sample K data from  $G_\theta(z)$  with different noises [negative,  $S_{neg}$ ]
  - Fix  $\theta$  and train  $\theta'$  to maximize the accuracy

 Maximize  $\sum_{x \in S_{pos}} [\log D_{\theta'}(x)] + \sum_{z \in S_{neg}} [\log(1 - D_{\theta'}(G_\theta(z)))]$

Learn generative model with (minimizing) Jensen-Shannon-divergence

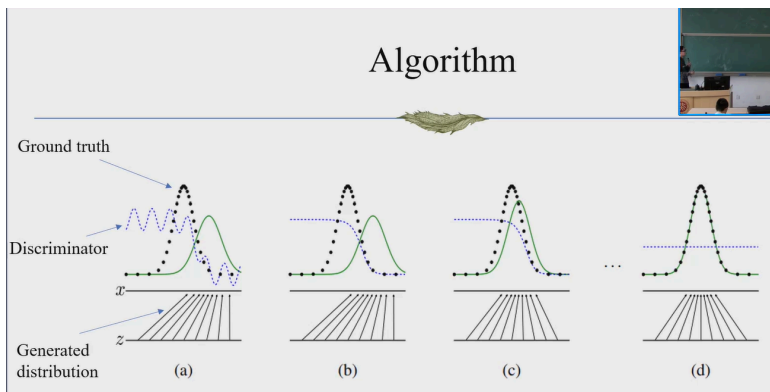
Minimize  $E_{x \sim P_*}[\log D_{\theta'}(x)] + E_{z \sim N(0,I)}[\log(1 - D_{\theta'}(G_\theta(z)))]$

Training of  $G_\theta(z)$

- Sample K data from  $G_\theta(z)$  with different noises [negative,  $S_{neg}$ ]
- Sample K data from the real dataset [positive,  $S_{pos}$ ]
- Fix  $\theta'$  and train  $\theta$  to minimize the loss

$$\sum_{x \in S_{pos}} [\log D_{\theta'}(x)] + \sum_{z \in S_{neg}} [\log(1 - D_{\theta'}(G_\theta(z)))]$$

- 通过这种方式即可进行训练，D又被称之为Discriminator，即区分样本是真是假。G则被称之为Generator，即生成希望能够以假乱真的样本。二者的优化方向是相反的，即生成模型希望能够生成相似真实图片的数据，而判别模型希望能够察觉出二者的不同，因此在真正训练之中通常的做法是固定一个参数，用minibatch的形式先更新优化一组参数，之后再用minibatch的形式更新优化另一组参数。依次更新Discriminator和Generator的信息。



- 上图就详细讲解了对抗优化的过程，第一步到第二步先是Discriminator更新，判别能力增强；第二步到第三步则是Generator的更新，最终达到分布的拟合。
- 之后人们对GAN产生了诸多改进，由于其中有minmax game的成分，其并不是凸性的，优化更新难度较大。而且Discriminator和Generator需要兼容配置，二者的能力需要匹配。更重要的是GAN极易容易**mode collapse**，即生成器为了攻破分类器，其可能会专注于生成一种样本，这样能够骗过分类器。这样生成器能够有很小的loss，因为在特定领域能攻破分类器，但是这并不说明生成器是可观且优秀的。在GAN的代码里使用Adam优化器的时候需要有详实的超参数调整，就是为了克服这种mode collapse情况。
- 1-Lipschitz**:  $|f(x) - f(y)| \leq |x - y| \quad \forall x, y$ ，理解为变化不太快的函数，导数不能超过1，即 $y = x$ 是1-Lipschitz的， $y = 2x$ 就不是1-Lipschitz的。
- 我们上节课讲述了衡量两个分布的方式是使用f-Divergence(Jensen Shannon Divergence)，但是还有其余的方式也可以描述好概率分布的相似性。我们考虑**Integral Probability Metric**，即如果对于任意的 $f(x)$  都有

$$E_{X \sim P} f(x) = E_{X \sim Q} f(x)$$

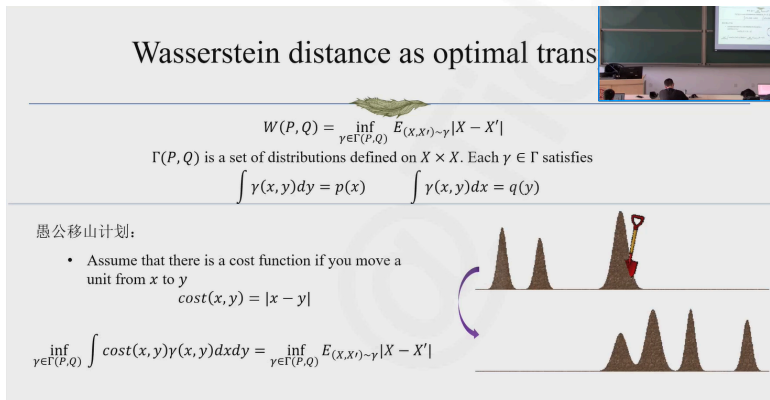
则认为这两个分布P, Q是相同的，但是这个值与所选函数关系极大，为了减少因为所选函数导致的问题，我们定义

$$D_F(P, Q) = \sup_{f \in F} |E_{X \sim P}(f(X)) - E_{X \sim Q}(f(X))|$$

，其中F代表一个函数族。这个值代表了两个分布的距离。一般我们认为 $f \in 1 - \text{LipchitzFunction}$ ，我们将这个距离称之为**WassersteinDist**。我们可以把这个公式改成如下的形式：

$$W(P, Q) = \inf_{\gamma \in t(P, Q)} E_{(X, X') \sim \gamma} |X - X'|$$

，其中 $t(P, Q)$ 是联合分布，满足其两个边缘分布分别是P和Q。定义在 $X * X$ 上，满足 $\int \gamma(x, y) dy = p(x)$ ； $\int \gamma(x, y) dx = q(y)$ 。 $(X, X') \sim \gamma$ 代表从联合分布 $\gamma$ 里抽取一对随机变量，然后对后面的函数 $|X - X'|$ 取期望。



- 利用愚公移山的例子可以进一步理解Wasserstein Distance， $\gamma(x, y)$ 代表要从x位置将物质移动到y位置的重量总和，运费 $cost(x, y) = |x - y|$ ，因此从P位置将物质移动到Q位置的最小移动代价就是Wasserstein Distance。
- $W(P, Q) = \sup_{f \in F} |E_{X \sim P}(f(X)) - E_{X \sim Q}(f(X))|$ ，按照GAN的思路， $P^*$ 代表ground truth，将Q认定为generator生成的数据，将其从标准正态分布之中采样，于是我们使用一个生成器和一个分类器进行，上式化简为： $W(P^*, G_\theta(z)) = \sup_{f \in F} |E_{X \sim P^*}(f(X)) - E_{z \in N(0,1)}[f(G_\theta(z))]|$ ，但是所有满足1-Lipchitz的函数无法枚举，因此考虑使用一个神经网络来体现这些函数，化简为 $W(P^*, G_\theta(z)) = \sup_{\theta'} |E_{X \sim P^*}(f_{\theta'}(X)) - E_{z \in N(0,1)}[f_{\theta'}(G_\theta(z))]|$ ，使用 $\theta'$ 来代替原先的枚举，即利用 $f_{\theta'}$ 来求出所有满足条件的函数族。由于神经网络具有出色的对称性，绝对值可以直接去除，改成 $W(P^*, G_\theta(z)) = \sup_{\theta'} E_{X \sim P^*}(f_{\theta'}(X)) - E_{z \in N(0,1)}[f_{\theta'}(G_\theta(z))]$ ，因此最终这个形式是

$$\text{Minimize}_{G_\theta} \text{Maximize}_{f_{\theta'}} E_{X \sim P^*}[f_{\theta'}(X)] - E_{z \sim N(0,1)}[f_{\theta'}(G_\theta(z))]$$

- 我们把这两个值定义为Generator(G)和Critic(f)。我们能够确定的是：神经网络和1-Lipchitz函数族是不完全一致的，这个Distance可能无法反映真正的Wasserstein Distance。因此我们需要对函数有一些约束，上面我们得知了Lipschitz函数的性质，因此我们可以对函数导数进行限制，即对函数进行**Weight Clipping**，将所有的权重剪裁到指定的值里，这样在有上界的bounded closed domain里，Lipchitz有上界，使得神经网络最终拟合的函数更接近1-Lipchitz性质。
- 但是weight clipping只能确保Lipschitz constant是finite有限值，并不能确保其是一个极小值，因此我们还需要其他的诸如**Gradient penalty**的方法，即认为一个函数是Lipchitz-1函数当且仅当其中几乎全部的点的导数都是1，因此希望神经网络在这些1-Lipchitz点上进行优化，因此改进损失函数成为了：



- $Minimize_{G_\theta} Maximize_{f'_\theta} E_{X \sim P_*}[f'_\theta(X)] - E_{z \sim N(0,1)}[f_{\theta'}(G_\theta(z))] - \lambda E_{\hat{X} \sim \hat{P}}[(\|\nabla_{\hat{x}} f_{\theta'}(\hat{X})\|_2 - 1)^2]$
- $\hat{x} = \epsilon x_{real} + (1 - \epsilon)x_{fake}$ ，即真实坐标和虚假坐标的线性插值，求判别器critic对输入x的梯度向量的大小与1的差距决定惩罚。
- $\lambda$ 之后的项是新增的，这样只有Lipchitz接近1的时候才能使得 $f_{\theta'}$ 更容易到Maximize值，使得模型更容易在这些Lipchitz有限点上更容易被更新。
- GAN和WGAN的出现让**Adversarial**这个词上升到了AI领域里极高的高度，为此人们提出了adversarial attack等算法，通过一些特定noise的方法就能够使得输出有巨大的变化。（详情见Trustworthy AI: 25 Spring 可信机器学习）

## Normalizing Flow

- 我们之前学过了不同的生成模型，包括VAE（浅层，用非序列的方式生成结果），包括Autoregressive（深层，序列性质而且满足MLE的性质），包括GAN等等，但是自回归模型一般很难在视觉任务中产生结果，我们一般使用Normalizing Flow来作用于视觉，可以理解为**自回归模型在视觉领域的应用**。
- $image \sim x; noise \sim z; generator \sim g(z)$ ，从标准正态分布中采样出z，把这个值丢入神经网络中，生成一张图片x，我们希望求出 $\theta$ 对应出的density function，即 $p(x; \theta)$ 。
- 从一维的情况来看，我们设随机变量Z有密度函数 $p_Z(z)$ ，用一个神经网络 $g$ 对其进行变换之后 $X = g(Z)$ 即是所生成的图，我们认为X有密度函数 $P_X(x)$ 。我们假设 $g$ 函数是可逆函数，即一维情况下单调递增的，且 $g$ 与 $g^{-1}$ 都可导。我们把

$$P(X < x) = P(Z < g^{-1}(x)) = \int_{-\infty}^{g^{-1}(x)} P_Z(z) dz$$

对上式中的两侧同时对x求导有：

- $P_X(x) = P_Z(z)[g^{-1}(x)]'$
- 已知 $g(g^{-1}(x)) = x$ 是成立的，因此对其两边同时对x求导有： $g'(z)[g^{-1}(x)]' = 1$ ，代入上式有：

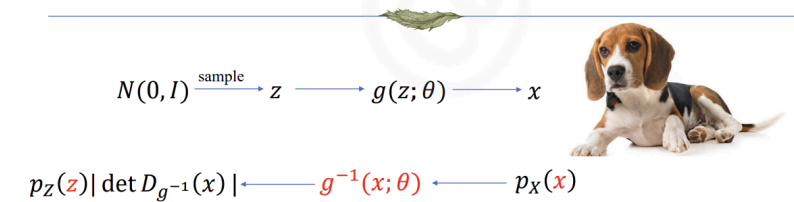
$$P_X(x) = P_Z(z)[g'(z)]^{-1}$$

- $Z \in R^d$  is a random vector with probability density  $p_Z(z)$
- $X = g(Z) \in R^d$  is a random vector with probability density  $p_X(x)$
- $x = g(z) = (g_1(z_1, \dots, z_d), \dots, g_d(z_1, \dots, z_d))$       Function mapping

$$p_X(x) = p_Z(z)[g^{-1}(x)]' \quad \longrightarrow \quad p_X(x) = p_Z(z)|\det D_{g^{-1}}(x)|$$

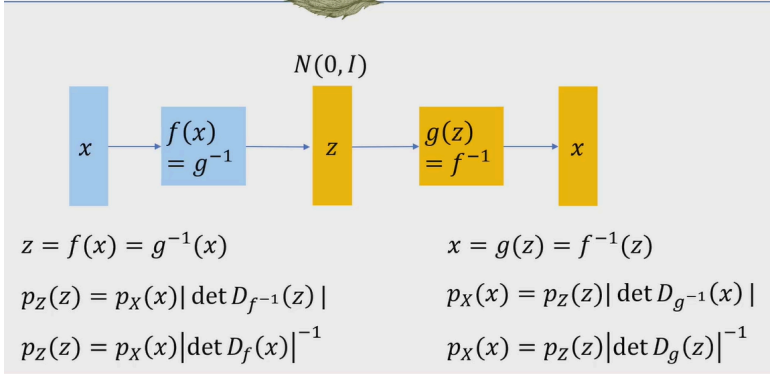
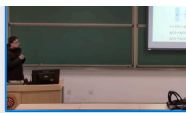
- 对于高维的情况而言，我们可以定义**微分同胚(Diffeomorphism)**，其从 $P_X(x)$ 的两部分出发，一部分是 $P_Z(z)$ ，即目标映射，另一部分是 $\det(D_g^{-1}(x))$ ，即Jacobian矩阵的行列式，故结果变成 $P_X(x) = P_Z(z)|\det D_g^{-1}(x)|$ 。

### High-dimensional case



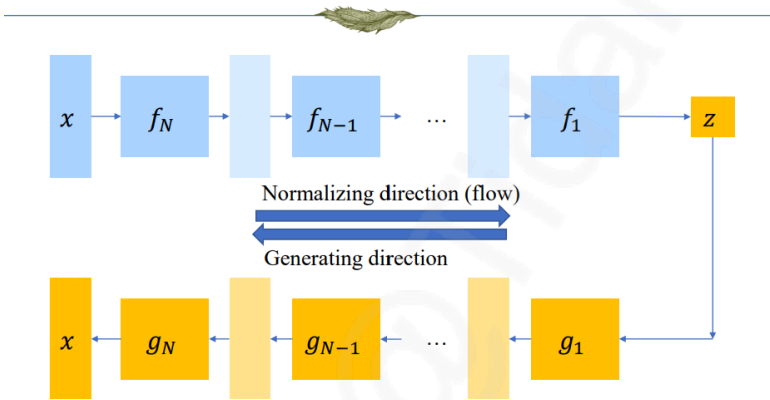
- The inverse function  $f = g^{-1}$  moves (**flows**) in the opposite: **normalizing** a complex distribution  $p_X$  to a more regular form of simple distribution  $p_Z$ .
- 我们将采样的概率 $z$ 称之为base density，将生成的概率 $x$ 称之为complex density normalizing flow的含义就是把一个复杂的概率分布flow为一个简单的概率分布。
- Flow的意思是将复杂的分布简单化，Normalizing的意思是将其归一化，从机理上 Normalizing Flow与VAE的相似度比较高，但是VAE的**encoder和decoder没有相关的约束限制（可导性的约束）**，因此其极其容易过拟合，导致VAE的深度不会过高。Flow对encoder和decoder的参数有较强的限制，即要求函数是可逆的。这种方式对函数的限制极大，因此会极大的缓解过拟合。

# Discussion



- 以上的这几个公式都是可以互相推理的，推理方式就像之前讲过的一样。
- 上图展示了Normalizing Flow的结构图，即对于一个输入，先normalizing，之后希望其可以恢复为原图。在实际情况中我们可以选择参数化f或者参数化g来解决问题。比如说参数化f的话，其后就需要通过Jacobian信息计算出每点的g结果是什么，进而获得函数的近似。
- 在不同的实践情景中我们可能需要参数化不同的函数。第一类应用是**density estimation**，即 给一个x之后将其传入模型，获得模型输出P(x)的值，这个应用一般会用于“找到最优图片”等地方，在这种应用的情况下一般参数化f，因为目的是求 $P_X(x)$ ，根据上面的知识我们知道 $P_X(x) = P_Z(f(x)) |det D_f(x)|$ ， $P_X(x) = P_Z(g^{-1}(x)) |det D_g(z)|^{-1}$ ，一定注意，根据题设，我们不知道Z的概率分布，因此不能直接套用公式。从描述x密度函数的两个方程来看，如果我们考虑参数化f的话，只需要手动计算Jacobian矩阵即可，但是如果我们考虑参数化g的话，我们还需要再计算其反函数，造成极大的时间消耗和浪费。因此这时我们一般选择参数化f。
- 第二类应用是**sampling**，即给模型传入一个噪音z，希望生成一个复杂的分布p(x)。此时我们 一般参数化g，训练的时候逆函数也需要求出，但是在训练结束之后直接使用的时候直接采用 $x = g(z)$ 即可，不用再计算反函数。如果参数化f的话最后使用的时候还需要使用逆函数。
- 不难发现，**Normalizing Flow**有很多问题，即神经网络里存在的大量的非线性变换和卷积操作等行为，这些行为组合之后如何**确保反函数是能够求出的**？甚至反函数的存在性都存疑。科学家的做法是寻找function class，满足这些function class里的函数是**容易计算的，容易求出可逆值的，而且能够计算Jacobian 行列式的**。

## Composition of layers



- 由于直接寻找到一个可逆的层将x直接变换为z是极其困难的，我们希望**寻找一系列**的层，只 需要确保这些层都是可逆的，通过叠加这些层可以最终生成z。
- 我们定义 $g(z) = g_N \dots g_1(z)$ ，代表函数的组合，定义 $x_n = g_n \dots g_1(z)$ ，即经过了n次函数组合的结果。因此根据前面讲到的公式， $p_X(x) = p_Z(z) |det D_{g^{-1}}(x)| = p_Z(z) |det D_{g_N^{-1}}(x_N)| \dots |det D_{g_1^{-1}}(x_1)|$ ，因此只需要每一层的函数都是可逆的，我们就可以依此信息计算出概率密度函数。

- **Elementwise Flow:**
- 即在normalizing flow的基础上加以限制，即每一个维度都不与其余的维度相关联，这时Jacobian Matrix是一个**对角矩阵**，计算detergent的时候极为简单。 $g(z) = (h(z_1), h(z_2), \dots, h(z_d))$ 其中如果h是可逆的，那么g(z)函数本身是可逆的。我们于是想到使用合适的激活函数，比如说Relu函数来充当h(x)函数，这样整个函数都是可逆的。这个方式计算很高效，但是capacity是不足的。
- Capacity: ---; Efficiency: +++
- **Linear Flow:**
- $g(z) = Az + b$ ，其中的A是一个 $d * d$  Matrix，而b是bias term。如果A矩阵是可逆矩阵，g(z)也是可逆的，即 $g^{-1}(x) = A^{-1}(x - b)$ 。此时Jacobian Matrix是容易计算的，即 $det D_g(z) = |A|$ ,  $det D_{g^{-1}}(x) = |A^{-1}|$ 。但是这个方式的问题是矩阵计算的时间复杂度太大，对于一个常规矩阵而言，其计算行列式的时间复杂度是 $O(n^3)$ ，Capacity: +; Efficiency: --。
- 根据矩阵的一些特性，我们可以使用特殊的矩阵来约束结果，对于**上三角和下三角矩阵**而言，其逆矩阵的计算是极为容易的，只需要确保对角线上的元素均不为0即可得知此矩阵是可逆的。而行列式只需要将diagonal上的

数相乘即可，即 $\det D_g(z) = \prod \text{diagonal}(A)$ 。计算行列式的时间复杂度是 $O(n)$ ，计算逆矩阵的时间复杂度是 $O(n^2)$ 。

- 如果换一种方式约束矩阵，将矩阵设计为**正交矩阵**，其可逆性是易得的，即 $A^{-1} = A^T$ 。 $g(z) = Az + b$ 中可以得出 $D_g(z) = A$ ， $D_{g^{-1}}(x) = A^T$ ， $\det D_g(z) = \det D_{g^{-1}}(x) = 1$ 。这个结果极其出色，即计算逆矩阵和行列式的时间复杂度都无限小。可是我们应该如何限制A矩阵才能确保其是正交矩阵？我们一般使用Household Transform。
- 如果从结构视角看，MLP 也可以看作是由可逆流（linear + elementwise）交替组成的一个映射，这里面stack的是linear，而elementwise就是充当可逆的映射部分。
- 注意：**在训练的时候normalizing flow的目标是传入一个真实图片，希望编码出的z尽量接近标准正态分布；在推理的时候则是生成过程，传入标准正态分布的噪音，希望能够生成接近原图的样态。

**Residual flow:**

- 我们认为 $g(z) = z + h(z)$ ，将输入的z分成两部分，即 $z = (z_1, z_2)$ ，其中 $z_1 \in R^{d_1}, z_2 \in R^{d-d_1}$ ，我们定义 $h_1 : R^{d-d_1} \rightarrow R^{d_1}$ ， $h_2 : R^{d_1} \rightarrow R^{d-d_1}$ 。即将z拆分为两部分，每一部分都计算之后得出两部分的x，把两部分的x叠加起来之后得出最终的结果x。这时候反向求解：

$z_2 = x_2 - h_2(x_1) \quad z_1 = x_1 - h_1(z_2) = x_1 - h_1(x_2 - h_2(x_1))$

- $g(z) = z + h(z)$
- Additive couple function:
  - $x_1 = z_1 + h_1(z_2)$
  - $x_2 = z_2 + h_2(x_1)$
  - $x = (x_1, x_2) = g(z)$
- Invertible with any  $h_2, h_1$ 
  - $z_2 = x_2 - h_2(x_1)$
  - $z_1 = x_1 - h_1(z_2) = x_1 - h_1(x_2 - h_2(x_1))$

- Capacity ++++
- Efficiency ----
- Practical +++

```
graph TD
    z1[z1] --> plus1((+))
    z2[z2] --> h1[h1]
    plus1 --> x1[x1]
    x1 --> plus2((+))
    z2 --> plus2
    plus2 --> x2[x2]
    x2 --> h2[h2]
    x1 --> h2
    h2 --> plus3((+))
    plus3 --> z2_out[z2]
    z2_out --> h1_inv[h1]
    x1 --> h1_inv
    h1_inv --> plus4((+))
    plus4 --> z1_out[z1]
```

**Residual Flow Ver.2:**

- $g(z) = z + h(z)$ 的计算方式下，如果满足Lipchitz constraint有 $Lip(h) < 1$ ，即可以证明 $g(z)$ 是可逆的，但是其Efficiency极低，因为虽然理论上 $g(z)$ 是可逆的，其可逆解只能通过approx逼近手段求出，因此效率极低。

- $g(z) = z + h(z)$
- Lipchitz constraint:
  - $Lip(h) < 1$
- $g(z)$  is invertible

- If  $z_1 \neq z_2$  and  $z_1 + h(z_1) = z_2 + h(z_2)$
- We have  $\frac{h(z_2) - h(z_1)}{z_2 - z_1} = 1$
- Contradict with  $Lip(h) < 1$

※ If  $z_1 \neq z_2, z_1 + h(z_1) \neq z_2 + h(z_2)$

- Capacity ++
- Efficiency -----
- Practical ++

**Continuous Flow:**

- 理解需要常微分方程(ODE)的知识，一个一阶的常微分方程可以被定义为 $\frac{dx}{dt} = f(x, t)$ ，我们希望找到一个函数 $x^*(t)$ ，满足这个微分方程，而且满足柯西解（初值条件）。在大多数时刻除了微分方程本身和初值条件，我们还需要目标点的信息才能计算出analytical solution。通过**皮卡存在性条件**我们可以得知解的唯一性和存在性条件。

**Picard's Existence and Uniqueness Theorem**

Suppose  $f(x, t)$  and  $\frac{\partial f(x, t)}{\partial x}$  are continuous functions in some open rectangle  $R = \{(x, t): a < x < b, c < t < d\}$  that contains  $(x, t) = (z, t_0)$ , then there exists  $h$  such that the ODE has unique solution in  $[t_0 - h, t_0 + h]$

Connection to normalizing flow

- Given a noise  $z$
- Given the parameter  $\theta$
- Solve the ODE:
$$\frac{dx}{dt} = f(x, \theta)$$
$$x(0) = z$$
- Obtain  $x(1)$  and output (e.g., an image)

This is a normalizing flow

- The solution  $x(1) = \Phi(z, \theta)$  is invertible
  - Intuition
  - Given  $z_1 \neq z_2$ ,
  - If  $x_1(1) = x_2(1) = y \quad \frac{dx}{dt} = f(x, \theta)$ 
$$x(1) = y$$

- 传统的ODE告诉我们 $\frac{dx}{dt} = f(x, t)$   $x(t_0) = z$ ，这个式子需要改为 $\frac{dx}{dt} = f(x, t, \theta)$   $x(0) = z_0$ 。将时间的维度取消，我们得到了time-independent ODE。 $\frac{dx}{dt} = f(x, \theta)$   $x(0) = z_0$ 。但是这个也与normalizing flow不太一致。
- 我们可以认为给一个noise z和参数 $\theta$ 之后，我们解ODE，解完之后的 $x(1)$ 就是我们需要的图片输出。可以**证明ODE是有可逆性的**，即使用反证法，如果 $z_1 \neq z_2$ 且 $x_1(1) = x_2(1) = y$ ，此时满足的是同一个ODE，因此 $z_1 = z_2$ ，故矛盾，因此我们可以证明其可逆性。因此ODE也是一种Normalizing Flow。
- 阅读Neural Ordinary Differential Equations(Best Paper in NIPS 2019)

Diffusion Model

- **扩散过程的insight:** 不管一开始的时候扩散多复杂，最终都会达到稳态的 $N(0, 1)$ 。我们希望从一个复杂的分布出发，一直变换到 $N(0, 1)$ ，之后再变换回原本的复杂的分布。
- $x \xrightarrow{\quad} z \sim N(0, 1) \xrightarrow{\quad} x$ 。（后部分使用神经网络来拟合）
- **扩散模型的三个挑战:**
- 如何模拟对数据集的扩散过程，如何反向进行这个过程，如何训练模型？

• **前向传播（Forward Process）:**

- 即对数据加噪音的过程。我们将 $x_0$ 代表为image， $q(x_0)$ 中的q代表与前向传播有关的分布， $N(x; \mu, \sigma^2 I)$ 代表一个高斯分布，其中 $x$ 代表随机变量，即高斯分布是与x的分布相关的。 $x = \mu + \epsilon \sigma$  则代表采样过程，从分布中采样出一个结果。

- $$q_1(x_1|x_0) = N(x_1; \sqrt{1 - \beta_1} * x_0, \beta_1 I), \beta_1 \in (0, 1)$$
- $$x_1 = \sqrt{1 - \beta_1} * x_0 + \sqrt{\beta_1} * \epsilon_1, \epsilon_1 \in N(0, I)$$
- 这个过程可以理解为向其中加噪音（ $\sqrt{\beta_1}$ ），之后再移除其中的语义部分（ $\sqrt{1 - \beta_1}$ ），形成下一张图，这一系列的过程可以用串联顺序的形式表示，用一系列的图的变化来体现这个过程，进而希望能找到描述整个过程的表达式。

Forward process

Data

$x_0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad \dots \quad x_T$

$q_1(x_1|x_0) = N(x_1; \sqrt{1 - \beta_1} * x_0, \beta_1 I), \beta_1 \in (0, 1)$

$x_1 = \sqrt{1 - \beta_1} * x_0 + \sqrt{\beta_1} * \epsilon_1, \epsilon_1 \sim N(0, I)$

Add noise to the image (by  $\sqrt{\beta_1}$ ) and remove the contextual information in x (by  $\sqrt{1 - \beta_1}$ )

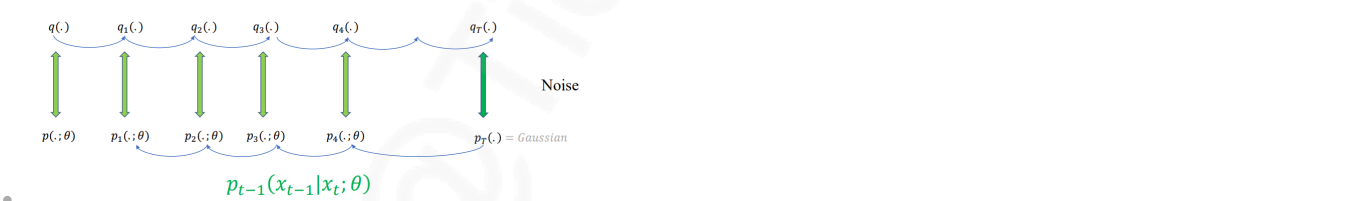
$q_{t+1}(x_{t+1}|x_t) = N(x_{t+1}; \sqrt{1 - \beta_{t+1}} * x_t, \beta_{t+1} I)$

$x_{t+1} = \sqrt{1 - \beta_{t+1}} * x_t + \sqrt{\beta_{t+1}} * \epsilon_{t+1}, \epsilon_{t+1} \sim N(0, I)$

- 我们把式子中的关系一层一层代入，最终可以得到 $x_{t+1} = \sqrt{\prod(1 - \beta_i)}x_0 + \sqrt{1 - \prod(1 - \beta_i)}\epsilon$ ，化简之后用正态分布的形式表示则是 $x_{t+1} = \sqrt{\alpha_{t+1}} * x_0 + \sqrt{1 - \alpha_{t+1}}\epsilon, \epsilon \sim N(0, I)$ ，因此当 $T$ 趋向于 $\infty$ 的时候， $\alpha_t$ 趋于0，因此这个分布一定会逐渐趋向于 $N(0, I)$ ，从单点的分布来看， $q_1(x_1) = \sum q(x_0)q_1(x_1|x_0)$ ，即每一个像素点都与原图的每一个点都有关联。

• **反向传播（backward process）**

- 即希望从一个高斯分布之中不断经过改变形成最初的照片信息，这个过程很复杂，需要使用神经网络来实现。前向传播的最后阶段和反向传播的第一阶段，两者的分布信息是一致的，因此我们可以假设**前向传播的每一个过程和反向过程的每一个过程是一一对应的**，如果每一步都严格一致的话最终的生成结果也一定是合理的，进而证明diffusion model的合理性。



Summary

- Forward process (noising)
  - Given any image  $x_0 \sim q_0(x_0)$
  - $x_t \sim q_t(x_t|x_{t-1}) = N(x_t; \sqrt{1 - \beta_t} * x_{t-1}, \beta_t I)$
  - $x_T \sim N(0, I)$
- Backward process (denoising)
  - Given any noise  $x_T \sim N(0, I)$
  - $x_{t-1} \sim p_{t-1}(x_{t-1}|x_t; \theta) = N(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$
  - We hope  $p_t(.) \approx q_t(.)$

• The forward process is fixed.  $\beta_t, \alpha_t = (1 - \beta_t) \dots (1 - \beta_1), \sigma_t$  are not learnable parameters

- 用形式化的语言来描述，给定任何的噪音 $x_T \sim N(0, I)$ ， $x_{t-1} \sim p_{t-1}(x_{t-1}|x_t; \theta)$ ，我们希望对于每一个t都有 $p_t = q_t$ ，其中 $p_{t-1}(x_{t-1}|x_t; \theta)$ 是一个神经网络（给定 $x_t$ 的条件下，通过神经网络 $\theta$ 求出 $x_{t-1}$ 的概率分布），我们希望这个神经网络可以尽可能保证每一个时刻的forward process和backward process是一致的。
- **DDPM:**
- 一切生成模型的核心都是优化MLE，即最大似然估计，我们希望参数下 $q_0(x_0)$ 的期望最大，就是参数下第一步的最大似然最大。之所以不设计对其余步的loss损失，是因为其余步都是我们制造出的有干扰的图，去拟合有干扰的图不具有统计学意义。
- MLE的含义是最大似然，我们认为样本是iid的，因此可以连乘代表联合概率，对其取对数变成加法，再用样本总数N归一化就是下面的期望形式。



- $$\text{minimize } E_{q_0(x_0)}(-\log p_0(x_0; \theta))$$

是我们希望优化的目标。

- 首先我们先理清上面期望式子的含义： $q_0(x_0)$ 表示真实数据分布， $p_0(x_0; \theta)$ 是模型希望学到的分布，因此这个期望表示在真实数据分布 $q_0$ 下，模型给出的概率 $p_0(x_0)$ 平均有多大。

假设有一个随机变量  $X$ ，它服从分布  $p(x)$ 。

如果我们要求某个函数  $f(X)$  的平均值（期望），记作：

$$E_{p(x)}[f(X)] = \int f(x) p(x) dx$$

意思是：

在  $p(x)$  这个分布下，随机采样很多个  $x$ ，对  $f(x)$  求平均。

- 但是 $p_0$ 是难以计算的，其是一个神经网络，并不是一个简单的概率分布。因此我们需要对其进行一些调整，首先加入额外条件的独立采样并不会影响期望的值，我们约定 $z = x_1 : x_t$ 再在等式上下同乘即可得到：

- $$\mathbb{E}_{q_0(x_0) q_{1,...,T}(z|x_0)} \left[ -\log \frac{q_{1,...,T}(z|x_0)}{q_{1,...,T}(z|x_0)} p_0(x_0; \theta) \right]$$

- 对这个公式的 $p_0(x_0; \theta)$ 用Bayesian展开则是

- $$\mathbb{E}_{q_0(x_0) q_{1,...,T}(z|x_0)} \left[ -\log \frac{q_{1,...,T}(z|x_0)}{q_{1,...,T}(z|x_0)} \frac{p_0(x_0, z; \theta)}{p_0(z|x_0; \theta)} \right]$$

- 再重组之后黄色部分是待求值，紫色部分是-KL散度，其值一定为非正数，其值是无法求导的，因此黄色的值是ELBO。

$$\underbrace{E_{q_0(x_0) q_{1,...,T}(z|x_0)} \left[ -\log \frac{p_0(x_0, z; \theta)}{q_{1,...,T}(z|x_0)} \right]}_{\text{ELBO}} + \underbrace{E_{q_0(x_0) q_{1,...,T}(z|x_0)} \left[ -\log \frac{q_{1,...,T}(z|x_0)}{p_0(z|x_0; \theta)} \right]}_{\geq 0}$$

- 注：紫色部分有错误，应该是小于等于0！
- 注：KL散度后面项无法求导的原因是： $\nabla_{\theta} K(\theta) = -E_{q_0(x_0) q(z|x_0)}(\nabla_{\theta} \log p_{\theta}(z|x_0))$ ，拆成 $\log p_{\theta}(z|x_0) = \log p_{\theta}(x_0, z) - \log p_{\theta}(x_0)$ ，其中后一项是无法计算偏导的，我们无法得知已知神经网络参数的情况下生成真实分布的概率。这是一个高斯路径积分，需要验证所有的先验。
- 我们只对黄色部分继续优化，经过拆分和马尔可夫性我们可以推导出最终的training loss为：

- $$\text{minimize } E_{q_0, ..., T(x_0:x_T)} [-\log p_0(x_0|x_1; \theta) + \sum_{t=2}^T KL(q_{t-1}(x_{t-1}|x_t, x_0) || p_{t-1}(x_{t-1}|x_t; \theta))]$$

- 为了最小化这个期望，与P相关的分布是可以计算的，因为其只与上一步有关系，Q相关的分布也是可以计算的，因为其都是高斯分布相关的。 $p_{t-1}(x_{t-1}|x_t; \theta) \sim N(x_{t-1}; \mu_{\theta}(x_t, t), \sigma_t^2 I)$ ，同理也可以定义 $q_{t-1}(x_{t-1}|x_t, x_0) \sim N(x_{t-1}; \hat{\mu}_t(x_t, x_0), \hat{\beta}_t I)$ 。
- 最后通过两个高斯分布之间的KL散度的公式，第一项和最后一项都与 $\theta$ 无关。根据定义，可以理解为两个高斯分布之间的距离，定义为下图的样子，这个即是一种回归的假设，最终的目的就是minimize两个高斯分布之间的距离。

- $$\frac{1}{2\sigma_t^2} ||\hat{\mu}_t(x_t, x_0) - \mu_{\theta}(x_t, t)||_2^2$$

是最终的优化目标，我们希望minimize这个值。

### Reparameterization

$$\text{minimize } E_{q_{0:T}(x_0:x_T)} [-\log p_0(x_0|x_1; \theta) + \sum_{t=2}^T KL(q_{t-1}(x_{t-1}|x_t, x_0) || p_{t-1}(x_{t-1}|x_t; \theta))]$$

$$\hat{\mu}_t(x_t, x_0) = \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t} x_0 + \frac{\sqrt{1-\beta_t}(1-\alpha_{t-1})}{1-\alpha_t} x_t$$

$$x_t = \sqrt{\alpha_t} \cdot x_0 + \sqrt{1-\alpha_t} \epsilon$$

$$\hat{\mu}_t(x_t, x_0) = \frac{1}{\sqrt{1-\beta_t}} (x_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} \epsilon)$$

$$\frac{1}{2\sigma_t^2} ||\hat{\mu}_t(x_t, x_0) - \mu_{\theta}(x_t, t)||_2^2$$

- If we parameterize  $\mu_{\theta}$  as

$$\mu_{\theta}(x_t, t) = \frac{1}{\sqrt{1-\beta_t}} (x_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} \epsilon_{\theta}(x_t, t))$$

- 上面的公式还是不太优雅，我们可以进一步化简，因为我们已知 $\hat{\mu}_t(x_t, x_0)$ 的表达式，我们可以把式子中的 $x_0$ 消去， $\hat{\mu}_t(x_t, x_0) = \frac{1}{\sqrt{1-\beta_t}} (x_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} \epsilon)$ ，同理我们可以人为设定的参数化 $\mu_{\theta}$ 的值也为这个格式，唯一不同的是用 $\epsilon_{\theta}(x_t, t)$ 。因此上述的二范数公式可以改写为：

- $$\text{minimize } \frac{\beta_t^2}{2\sigma_t^2(1-\beta)(1-\alpha_t)} ||\epsilon - \epsilon_{\theta}(x_t, t)||_2^2$$

- $$\text{minimize} \quad \|\epsilon - \epsilon_\theta(\sqrt{\alpha_t}x_0 + \sqrt{1 - \alpha_t} * \epsilon)\|_2^2$$
- Diffusion模型经常使用U-Net的结构外加上残差块和注意力层来实现，Diffusion也有可能使用Transformer的架构加以实现。为了告知模型当前的时间步，进而使得模型知道当前的噪音等级，一般会使用余弦编码对位置进行编码加入到U-Net之中。
- 在DDPM的实现中，Ho.et al在2020年的论文里将 $\sigma_t^2$ 设定为 $\beta_t$ ，即加噪程度。之后有一些研究认为可以将其值改为可学习的参数。
- DDPM & Score Function:**
- 从其他角度理解一下DDPM，下图推导之后得到的结果是

$$\nabla_{x_t} \log q_t(x_t|x_0) = -\frac{1}{\sqrt{1 - \alpha_t}} \epsilon$$

，而我们称对一个概率密度函数的一个点求导，这个值就是Score Function。

- Score matching的意思不是让神经网络去match真实未加噪的图的Score Function，而是希望神经网络去match前向过程中逐步加噪之后产生的图的Score Function, Score Function和分布基本是一一对应的。

Recall

$$q_t(x_t|x_0) \leftarrow N(x_t; \sqrt{\alpha_t} \cdot x_0, (1 - \alpha_t)I), \alpha_t = (1 - \beta_1) \dots (1 - \beta_t)$$

$$q_t(x_t|x_0) = \frac{1}{\sqrt{(2\pi)^d(1 - \alpha_t)}} e^{-\frac{1}{2(1 - \alpha_t)}(x_t - \sqrt{\alpha_t}x_0)^T(x_t - \sqrt{\alpha_t}x_0)}$$

$$\log q_t(x_t|x_0) = \log \frac{1}{\sqrt{(2\pi)^d(1 - \alpha_t)}} - \frac{1}{2(1 - \alpha_t)}(x_t - \sqrt{\alpha_t}x_0)^T(x_t - \sqrt{\alpha_t}x_0)$$

$$\nabla_{x_t} \log q_t(x_t|x_0) = -\frac{1}{(1 - \alpha_t)}(x_t - \sqrt{\alpha_t}x_0)^T$$

$x_t = \sqrt{\alpha_t} \cdot x_0 + \sqrt{1 - \alpha_t} \epsilon, \epsilon \sim N(0, I)$

- DDPM & Stochastic Differential Equation:**
- 数学系物理系的人会用微分方程的角度描述DDPM，应用映射把stepwise的过程映射为(0, 1)实数上的稠密过程。T是一个确定值，是一个极大的值，t是自变量， $\frac{t}{T}$ 代表出T和t的位置关系，如果较为接近1的话代表步数较多，较为接近0的话代表步数较少。

Consider the limit of many small steps

$$x_{t+1} = \sqrt{1 - \beta_{t+1}} \cdot x_t + \sqrt{\beta_{t+1}} \epsilon$$

Let  $\beta\left(\frac{t}{T}\right) = T\beta_t$ , where  $\beta(\cdot)$  is define on  $[0, 1]$

$$x\left(\frac{t+1}{T}\right) = \sqrt{1 - \frac{\beta\left(\frac{t+1}{T}\right)}{T}} \cdot x\left(\frac{t}{T}\right) + \sqrt{\frac{\beta\left(\frac{t+1}{T}\right)}{T}} \epsilon$$

Let  $x\left(\frac{t}{T}\right) := x_t$

See "Score-based Generative Modeling Through Stochastic Differential Equations" for details

Consider the limit of many small steps

$$x(t + \Delta t) = \sqrt{1 - \beta(t + \Delta t)\Delta t} \cdot x(t) + \sqrt{\beta(t + \Delta t)\Delta t} \epsilon$$

$$x(t + \Delta t) \approx x(t) - \frac{1}{2}\beta(t)\Delta t x(t) + \sqrt{\beta(t)\Delta t} \epsilon$$

Taylor expansion

$$dx(t) \approx \underbrace{-\frac{1}{2}\beta(t)x(t)dt}_{\text{Drifting term}} + \underbrace{\sqrt{\beta(t)}dw_t}_{\text{diffusion term}}$$

Called forward diffusion stochastic differential equation

Continuous-time Markov process

- 把 $\frac{t}{T}$ 写成t， $\frac{1}{T}$ 写成 $\Delta(t)$ ，之后再进行泰勒展开，可以找到一种递推关系，化简用连续时间的马氏过程来化简。这就关联起了forward process与SDEs的关系。对于ODE，正向过程和反向过程是一致的，但是对于SDE，正向过程和反向过程是不一致的，科学家们推导出forward process是直接计算的，backward process则需要多加一项score function项，这个项是难以计算的，因此还是需要神经网络来估计score function。这说明了数学上的SDE和推导出的diffusion model是大一统的。

$$dx(t) \approx \underbrace{-\frac{1}{2}\beta(t)x(t)dt}_{\text{Drifting term}} + \underbrace{\sqrt{\beta(t)}dw_t}_{\text{diffusion term}}$$

Forward diffusion SDE

$$dx(t) = \underbrace{\left[-\frac{1}{2}\beta(t)x(t) - \beta(t)\nabla_{x(t)} \log q_t(x_t)\right]dt}_{\text{Drifting term}} + \underbrace{\sqrt{\beta(t)}dw_t}_{\text{diffusion term}}$$

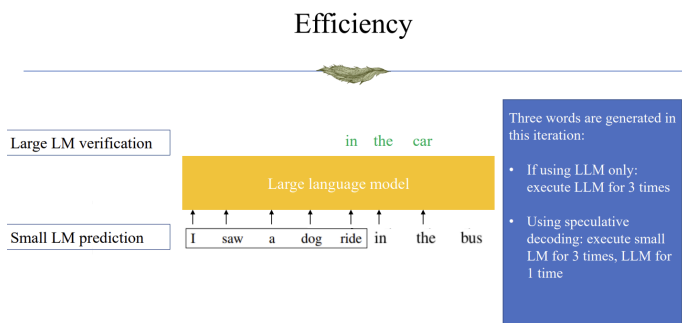
Reverse diffusion SDE

- 推荐阅读：Denoising Diffusion Probabilistic Models, DPM-Solver, Score-Based Generative Modeling Through Stochastic Differential Equations

## 25.11.14 Acceleration

- High-level idea: 用小的语言模型去生成，用大的语言模型来校正
- Draft Construction:** 利用small LM针对之前的输入生成一系列的sequence， $(x_{t+1}, \dots, x_{t+m})$ ，小语言模型针对prompt生成的信息；**Draft Verification:** 将整个生成的语段，包括prompt和draft传给LLM，通过一次前转让LLM找到第一个问题的所在位置；**Draft Correction:** 将第一个问题的所在位置换成LLM给的指导词，将正确的部分 $(x_{t+1}, \dots, x_{t+m'})$ 给用户，并且利用small LM继续建造一个draft，循环往复。

- 这个做法可以显著加快LLM的推理速度，原本生成三个词需要三次LLM的decode，但是修改之后不需要这么多次数的decode。



- 但是我们需要验证正确性，即speculative decoding equals to sampling from LLM，即给定两个分布 $p(x)$ 和 $q(x)$ ，我们能不能从 $p(x)$ 中采样数据，使得采样到的结果是符合 $q(x)$ 分布的？其中 $p(x)$ 是small LM的分布， $q(x)$ 是LLM的分布

#### Algorithm 1 Token-level maximal coupling

Input: Distributions  $p, q$ , Draft sample  $X \sim p$ .

1: Compute the residual distribution  $p^{res}$  where

$$\forall x \in \Omega, p^{res}(x) = \frac{q(x) - \min\{p(x), q(x)\}}{1 - \sum_{x'} \min\{p(x'), q(x')\}}.$$

2: Sample  $\eta \sim U(0, 1)$ .

3: if  $\eta \leq \min\left(1, \frac{q(X)}{p(X)}\right)$  then

4:   Return  $Y = X$ . {Accept the draft token.}

5: end if

6: Return  $Y \sim p^{res}$ . {Sample a corrected token from the residual distribution.}

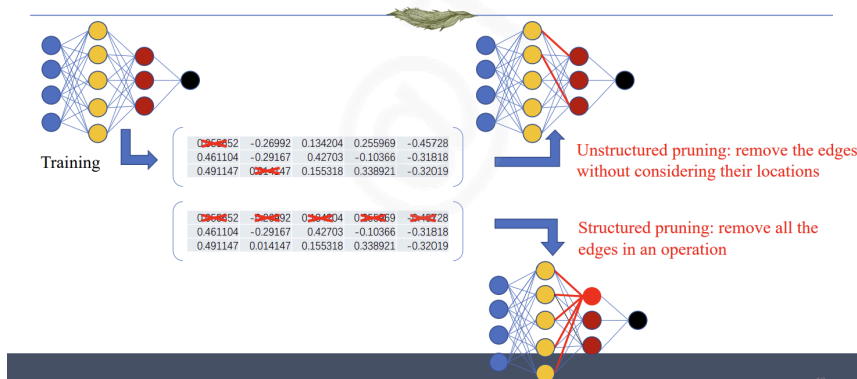
If  $q(x) < p(x)$

- $p^{res}(x)$ 是大模型在小模型中没被选中的部分，还想要补采样的分布，分母的目的是为了确保概率归一化，随机采样 $\eta$ ，如果 $q(X) > p(X)$ ，即大模型认为生成的概率比小模型更高的时候，认为这是一个合理的采样，因此选择小模型生成的分布 $X$ 为输出，即 $Y = X$ ；如果大模型认为生成的分布概率更高，即 $q(X) < p(X)$ ，则认为这不是一个合理的采样，因此需要采样 $p^{res}$ 为最终的结果，即采样大模型中有，但是小模型中没有被选中的部分。

#### Pruning

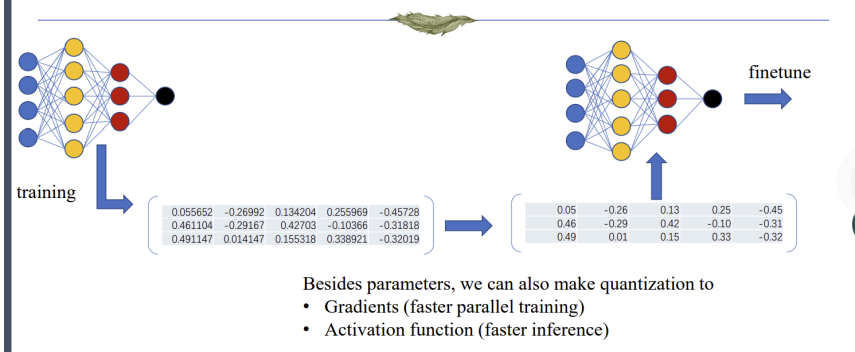
- 在训练的过程中，如果将weight\_matrix中接近于0的值设置为0，即代表将图中的两条边剪枝掉，去除两条边。继续训练，持将weight\_matrix中接近于0的值设置为0，即代表将图中的两条边剪枝掉，继续训练。
- 这种持续不断的剪枝可以提高运算效率
- Unstructured Pruning：不考虑节点的位置，只依靠weight\_matrix的数值大小进行剪枝。这样看似可以提高运算效率，但是GPU中对稀疏矩阵的计算很迟缓，矩阵改成稀疏之后效果不明显。
- Structured Pruning：考虑节点的位置，删除节点，而不是单单去除边，在weight\_matrix表示为删除一整个矩阵的列。这样可以实打实地提高运行效率。

#### Structure pruning and unstructured pruning



- Quantization（常用）
- 数字的表示方式有诸多种，有单精度、双精度、半精度的表示方法。即训练的时候使用高精度进行训练，推理使用的时候使用低精度的方式进行推理，这样可以节约memory consumption并且fast computation，并且尽量确保表现降低不多。
- 训练的时候采用高精度的方式进行，训练结束之后对weight\_matrix round降低精度，之后采用finetune的方式再微调。当然quantization还可以对activation function和gradient进行，提高推理效率和实现更快的并行训练。
- 有一些团队尝试过把gradient和weight matrix clip映射到离散的值，进一步减少内存和消耗。但是Quantization的方法在工程上极其困难，pytorch不支持低精度的写法，如果改为低精度的类似3-bit写法，需要自己手写CUDA算法。同时低精度舍弃了太多信息之后对于最终结果的影响也很显著。

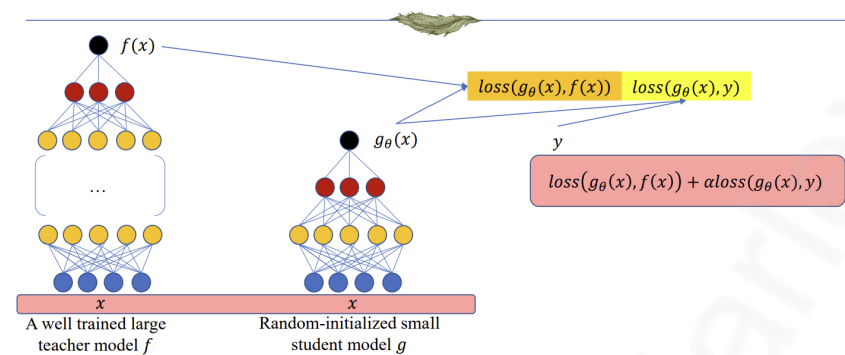
## Where to make quantization



### Knowledge Distillation

Student model and Teacher model，希望学生模型能够预测出教师模型的输出结果。学生模型的输出架构要与教师模型一致，有两个优化目标，第一个是希望小模型和大模型的输出一致，第二个是希望小模型的输出和最终的输出任务要求尽量接近，即 $loss(g_{\theta}(x), f(x)) + \alpha loss(g_{\theta}(x), y)$ 。这种loss比直接对着任务进行学习的效果要更好。但是对此没有过多的解释，人们将其认为一种好用的策略进行。

## The standard KD algorithm



也有研究员把KD的方法用到Diffusion Model里，即一开始逐步推理，之后融合起来Distill，逐步增多Distill的步数。