

0.搭建简单的神经网络

1.1一个简单的神经网络例子: FFN层(前馈网络层)

简单神经网络例子：前向传播与反向传播

假设我们有一个非常简单的两层神经网络，正要用一个样本(x, y)完成一次训练：

- **输入层** $x = (x_1, x_2)$, 其真实标签为 $y = 1$.
- **第一层**：有两个神经元，权重为 $W_1 = (w_{11} \quad w_{12} \quad w_{21} \quad w_{22})$, 偏置为 $b_1 = (b_{11}, b_{12})$
- **第二层(输出层)**：有一个神经元，权重为 $W_2 = (w_{31}, w_{32})$, 偏置为 b_2

损失函数 L 是均方误差 (MSE) , 假设真实标签是 y , 我们要最小化损失 $L = \frac{1}{2}(\hat{y} - y)^2$, 其中 \hat{y} 是网络的预测值。

1. 前向传播: 每一层的计算就是先 **线性变换** 得到加权输入(预激活值), 再来一次**非线性激活** 得到输出(激活值).

1.1 输入层到第一层的计算：

假设输入是 $x = (x_1, x_2) = (1, 2)$, 权重为 :

$$W_1 = (0.5 \quad -0.5 \quad 0.3 \quad 0.8), \quad b_1 = (0.1, -0.2)$$

第一层加权输入(预激活值) z_1 为:

$$\begin{aligned} z_1 &= W_1 \cdot x + b_1 = (0.5 \quad -0.5 \quad 0.3 \quad 0.8) \cdot (1 \ 2) + (0.1 \ -0.2) \\ z_1 &= (0.5 \cdot 1 + (-0.5) \cdot 2 + 0.1 \cdot 0.3 \cdot 1 + 0.8 \cdot 2 - 0.2) = (-0.9 \ 1.7) \end{aligned}$$

然后应用激活函数 (例如 ReLU等. 这里用最简陋的MAX.) 得到 **第一层的输出(激活值) a_1** :

$$a_1 = \text{ReLU}(z_1) = (\max(0, -0.9) \ \max(0, 1.7)) = (0 \ 1.7)$$

1.2 第一层到输出层的计算：

第二层加权输入(预激活值) z_2 为:

$$\begin{aligned} z_2 &= W_2 \cdot a_1 + b_2 = (0.2, -0.4) \cdot (0 \ 1.7) + 0.5 \\ z_2 &= (0.2 \cdot 0 + (-0.4) \cdot 1.7) + 0.5 = -0.68 + 0.5 = -0.18 \end{aligned}$$

然后应用激活函数 (假设没有激活函数, 即线性输出) 得到第二层(最终)输出:

$$\hat{y} = -0.18$$

损失函数计算：

真实标签 $y = 1$, 所以损失是 :

$$L = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(-0.18 - 1)^2 = \frac{1}{2}(1.18)^2 = 0.697$$

2. 反向传播：

现在我们需要计算损失函数 L 对每个权重参数（如 w_{11}, w_{12}, w_{31} 等）的偏导数。

步骤1：计算第二层(输出层)参数 { W_2 , b_2 }的梯度. $L = L(W_2, b_2)$. a_1 是第一层输出到第二层的, 此时看为常数/

首先, 计算损失函数 L 对输出 \hat{y} 的导数 :

$$\frac{\partial L(\hat{y})}{\partial \hat{y}} = \hat{y} - y = -0.18 - 1 = -1.18$$

然后, 计算损失函数 L 对 z_2 的导数. 注意其中第二层激活函数 $f_2(x) = x$, 是恒等变换.

$$\frac{\partial L(f_2(z_2))}{\partial z_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} = -1.18 \cdot 1 = -1.18$$

然后, 计算损失函数 L 对 权重 W_2 的梯度: (L 是 W_2, b_2 的函数)

\$\$ \begin{aligned} & \frac{\partial L(f_2(z_2))}{\partial W_2} \\ &= \frac{\partial L(z_2)}{\partial z_2} \cdot \frac{\partial z_2}{\partial W_2} \\ &= \frac{\partial L(z_2)}{\partial z_2} \cdot \frac{\partial (W_2 \cdot a_1 + b_2)}{\partial W_2} \\ &= \frac{\partial L(z_2)}{\partial z_2} \cdot a_1^T \quad (\text{注意矩阵求导后要转置, 最终结果必须和被求导的矩阵 } (W_2: 2 \times 1) \text{ 形状相同!}) \\ &= -1.18 \cdot a_1^T \end{aligned}

$$(0 \ 1.7)$$

\

&=

$$(0 \ -2.006)$$

\end{aligned} \$\$

计算损失对偏置 b_2 的梯度 :

$$\frac{\partial L}{\partial b_2} = \frac{\partial L(z_2)}{\partial z_2} \cdot \frac{\partial z_2(b_2)}{\partial b_2} = \frac{\partial L(z_2)}{\partial z_2} \cdot \frac{\partial (W_2 \cdot a_1 + b_2)}{\partial b_2} = \frac{\partial L}{\partial z_2} \cdot 1 = -1.18$$

步骤2：计算第一层参数(W_1 , b_1)的梯度. 此时 $L = L(W_1, b_1)$. a_1 也是 W_1, b_1 的函数.

首先, 计算 $L(a_1(W_1, b_1))$ 对 a_1 的导数.

\$\$ \begin{aligned} & \frac{\partial L(a_1(W_1, b_1))}{\partial a_1} \\ &= \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial a_1} \end{aligned}

$$\frac{\partial L}{\partial a_1} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} = \frac{\partial L}{\partial z_2} \cdot (W_2 \cdot a_1 + b_2) \cdot \frac{\partial a_1}{\partial a_1} = \frac{\partial L}{\partial z_2} \cdot W_2 \cdot 1 =$$

$$(0.2 - 0.4)$$

$$\cdot (-1.18) =$$

$$(-0.236 \ 0.472)$$

$$\end{aligned} \\$$

然后计算 L 对 z_1 的导数 ($a_1 = \text{ReLU}(z_1)$)

$$\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1}$$

因为我们使用的是 ReLU 激活函数, $\frac{\partial a_1}{\partial z_1}$ 会是 1 或 0。对于 $a_1 = (0 \ 1.7)$, 我们有 :

$$\frac{\partial L}{\partial z_1} = (0 \ 0.472)$$

计算 L 对权重 W_1 的梯度 :

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial z_1} \cdot x^T$$

假设输入 $x = (1 \ 2)$, 那么 :

$$\frac{\partial L}{\partial W_1} = (0 \ 0.472) \cdot (1 \ 2) = (0 \ 0 \ 0.472 \ 0.944)$$

计算 L 对偏置 b_1 的梯度 :

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_1} = (0 \ 0.472)$$

3. 更新参数 :

根据计算的梯度, 可以使用梯度下降法来更新参数 { W_1 , W_2 , b_1 , b_2 } :

$$W_1 \leftarrow W_1 - \alpha \cdot \frac{\partial L}{\partial W_1}, \quad W_2 \leftarrow W_2 - \alpha \cdot \frac{\partial L}{\partial W_2}, \quad b_1 \leftarrow b_1 - \alpha \cdot \frac{\partial L}{\partial b_1}, \quad b_2 \leftarrow b_2 - \alpha \cdot \frac{\partial L}{\partial b_2}$$

1.2 归一化层

1. 什么是 batch ?

在神经网络训练中, 我们通常不会一次性输入整个数据集, 而是**分批 (batch) 输入**, 这样可以提高计算效率, 并利用梯度下降优化参数。

举个例子

假设你有一个 10000 张图片的数据集 :

- **全数据（全部样本）训练**：把 10000 张图片一次性输入神经网络进行训练，这会导致内存消耗过大。
- **小批量（mini-batch）训练**：把数据集拆分成多个小批次，比如每次输入 32 张图片（batch size = 32）。
- **单样本（stochastic）训练**：每次只输入 1 张图片（batch size = 1）。

batch 就是一次输入神经网络的一组样本，例如：

- 如果 batch size = 32，表示**每次训练输入 32 张图片**。
- 如果 batch size = 64，表示**每次训练输入 64 张图片**。

batch 主要用于：

1. **减少内存占用**，避免一次性处理整个数据集。
2. **加速梯度计算**，因为 mini-batch 计算均值和梯度更加稳定。
3. **提高优化器的效果**，像 SGD（随机梯度下降）在 mini-batch 里计算梯度时更加高效。

2. 什么是归一化层？（Normalization Layer）

归一化（Normalization）就是对输入数据进行标准化，使其数值分布更加均匀，从而**加速收敛，提高泛化能力**。

2.1 为什么要归一化？

神经网络训练时，输入的数据如果数值范围相差太大，网络容易出现以下问题：

- **梯度消失/梯度爆炸**：如果输入数据过大或过小，经过多个层传递后，梯度可能会变得很小（学习变慢）或很大（不稳定）。
- **训练不稳定**：不同层之间的数值分布变化过大，导致训练时权重更新不均衡。

归一化层的作用是：

1. **调整数据分布**，让输入数据均值接近 0，方差接近 1，避免梯度问题。
2. **加速收敛**，让训练更快、更稳定。
3. **减少对初始化的敏感性**，让神经网络更容易训练。

3. 归一化层的几种类型

归一化层的核心思想是**对输入数据进行标准化**，常见的归一化方法有：

1. **Batch Normalization (BN)** （按 batch 归一化）
2. **Instance Normalization (IN)** （按单个样本归一化）
3. **Layer Normalization (LN)** （按特征层归一化）
4. **Group Normalization (GN)** （按多个通道归一化）

3.1 Batch Normalization (BN, 批量归一化)

归一化范围：在一个 batch 里的**相同通道上**计算均值和标准差。

假设 batch size = 32，每个样本是 3 通道（RGB）的 32×32 图片：

- 计算整个 batch 里的每个通道的均值和标准差。
- 所有样本在同一个通道上的数值会一起归一化。

数学公式

给定某个 batch 的输入 $x_{n,c,h,w}$, 它表示 :

- 第 n 个样本 (batch size 内)
- 第 c 个通道 (如 RGB 的 R 通道)
- 高度 h , 宽度 w

BN 计算公式 :

1. 计算均值和标准差 (在 batch 维度上计算)

$$\mu_c = \frac{1}{NHW} \sum_{n=1}^N \sum_{h=1}^H \sum_{w=1}^W x_{n,c,h,w}$$

$$\sigma_c^2 = \frac{1}{NHW} \sum_{n=1}^N \sum_{h=1}^H \sum_{w=1}^W (x_{n,c,h,w} - \mu_c)^2$$

- 计算整个 batch 的某个通道的均值和方差, 让同一通道的所有样本归一化。

2. 归一化

$$\hat{x}_{n,c,h,w} = \frac{x_{n,c,h,w} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$$

3. 加入可训练参数

$$y_{n,c,h,w} = \gamma_c \hat{x}_{n,c,h,w} + \beta_c$$

- γ_c 和 β_c 是可训练参数, 用于调整归一化后的数据。

特点 :

- 适用于大 batch 训练 (batch size 越小, 均值和标准差计算越不稳定)。
- 用于 CNN 任务, 如分类、检测等。

3.2 Instance Normalization (IN, 实例归一化)

归一化范围 : 对单个样本的每个通道计算均值和标准差, 而不考虑 batch 之间的关系。

数学公式

1. 计算均值和标准差 (在单个样本内的单个通道计算)

$$\mu_{n,c} = \frac{1}{HW} \sum_{h=1}^H \sum_{w=1}^W x_{n,c,h,w}$$

$$\sigma_{n,c}^2 = \frac{1}{HW} \sum_{h=1}^H \sum_{w=1}^W (x_{n,c,h,w} - \mu_{n,c})^2$$

2. 归一化

$$\hat{x}_{n,c,h,w} = \frac{x_{n,c,h,w} - \mu_{n,c}}{\sqrt{\sigma_{n,c}^2 + \epsilon}}$$

3. 加入可训练参数

$$y_{n,c,h,w} = \gamma_c \hat{x}_{n,c,h,w} + \beta_c$$

特点：

- 适用于风格迁移、医学图像等任务。
- 不受 batch size 影响。

1. 一些术语

学习率策略:

喵呜~对的！你猜得没错，这个是 **余弦退火学习率调度器** (Cosine Annealing Learning Rate Scheduler)！让我们详细解析一下它在代码中的作用~

1. CosineAnnealingLR 的作用:

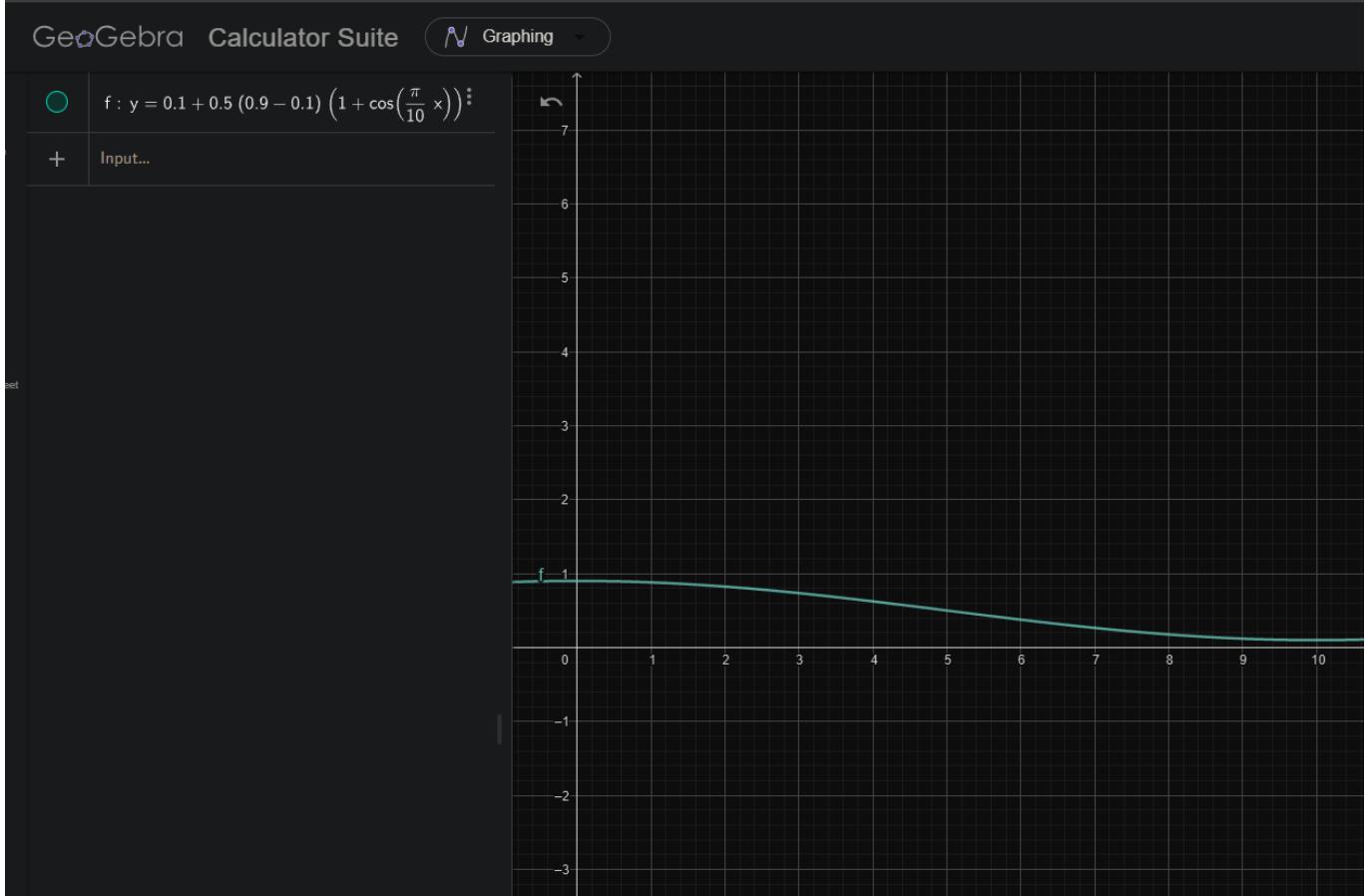
- **背景:** 学习率 (learning rate, lr) 是训练神经网络时非常重要的超参数。通常我们会通过某些策略来动态调整学习率，使得模型在训练过程中能够更好地收敛。**余弦退火学习率调度器**就是这样一种策略。
- **功能:** `CosineAnnealingLR` 调度器基于余弦函数来调整学习率。具体来说，学习率会从一个初始值开始，逐渐减小，并且按照余弦函数的形式在训练过程中逐渐降低，直到训练结束时趋近于一个最低值 (`eta_min`)。

数学上，余弦退火学习率的变化通常是这样计算的：

$$lr(t) = \eta_{\min} + \frac{1}{2} (\eta_0 - \eta_{\min}) \left(1 + \cos \left(\frac{t\pi}{T_{\max}} \right) \right)$$

- η_0 是初始学习率。
- η_{\min} 是最低学习率。
- T_{\max} 是一个周期 (通常是总训练步数或者训练的 epoch 数量)。
- t 是当前的训练进度 (epoch 或者 step)。

随着训练的进行，学习率会呈现出一个周期性地从高到低的变化，最后趋于最低值。



2. 神经网络的各种层

上采样 upsampling

指的是把低分辨率的数据转换为高分辨率的数据，在计算机视觉中主要用于恢复图像细节、提高分辨率。上采样通常在解码器（Decoder）或者生成模型（如 GAN）中使用，以便从小的特征图生成大尺寸的输出。事实上上采样和下采样都可以归为图像的重采样resample。

常见的上采样方法：

- **插值interpolation**

插值层一般没有可学习的参数. 是固定的生成.

- **最近邻插值 (Nearest Neighbor Interpolation)** : 复制最近的像素值。
- **双线性/双三次插值 (Bilinear/Bicubic Interpolation)** : 使用加权平均法插值。

- **转置卷积 (Transpose Convolution, 或反卷积)** : 学习一个**可训练的卷积核**, 用于恢复空间尺寸。

反卷积数学实现：

(1) 普通 3D 卷积的数学计算

标准的3D 卷积可以表示为：

$$Y(i, j, k) = \sum_{m=0}^{K_d-1} \sum_{n=0}^{K_h-1} \sum_{p=0}^{K_w-1} X(i+m, j+n, k+p) \cdot W(m, n, p)$$

其中：

- $X(i, j, k)$ 是输入特征图，大小为 $D_{in} \times H_{in} \times W_{in}$ 。
- $W(m, n, p)$ 是卷积核，大小为 $K_d \times K_h \times K_w$ 。
- $Y(i, j, k)$ 是输出特征图，大小为 $D_{out} \times H_{out} \times W_{out}$ 。

这个过程会降低分辨率 (如果步长 $S > 1$)。

(2) 3D 逆卷积 (Transpose Convolution) 的数学计算

在转置卷积中，我们希望执行卷积的逆操作，从 (Y) 反推 (X)。

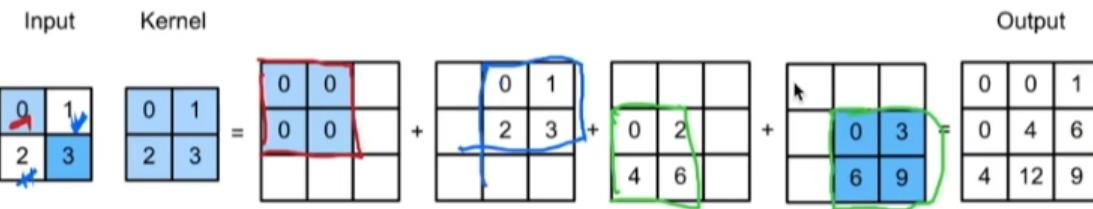
(2) 3D 逆卷积 (Transpose Convolution) 的数学计算

在转置卷积中，我们希望执行卷积的逆操作，从 Y 反推 X 。

计算方法：

$$X(i, j, k) = \sum_{m=0}^{K_d-1} \sum_{n=0}^{K_h-1} \sum_{p=0}^{K_w-1} Y(i//S_d, j//S_h, k//S_w) \cdot W(m, n, p)$$

这里 // 表示整除， S_d, S_h, S_w 是步长。



$$Y[i : i + h, j : j + w] += X[i, j] \cdot K$$



其实就是一个 2×2 卷积核，它对(比如说 2×2 大小的)input每个像素进行逆卷积(也就是卷积核每个元素都和这个input元素相乘，得到一个 2×2 的逆卷积结果。然后这些 2×2 的逆卷积结果拼起来得到 3×3 的输出。) 我们可以看到这个过程反过来就是卷积。

3.时间序列分析

3.1自回归过程 AutoRegressive Process, ARP

"假设当前数据点用过去数据点的线性组合估计未来."

定义：

一个 p 阶 AR 过程 (AR(p)) 的表达式：

$$Y_t = \mu + \sum_{i=1}^p \phi_i Y_{t-i} + u_t$$

- μ : 常数项 (Constant)
 - Y_{t-i} : 过去时间的数据点
 - ϕ_i : 自回归系数
 - u_t : 噪声项, 通常假设服从 **零均值正态分布** $\sim N(0, \sigma^2)$
-

例子：天气预测

假设过去几天的气温为 (Y_t), 如果我们使用 **二阶 AR 模型 (AR(2))**, 则：

$$Y_t = \mu + \sum_{i=1}^2 \phi_i Y_{t-i} + U_t$$

训练过程：

1. 每天多次拟合一次 (或者使用过往数据)。
2. 可以选择简单的损失函数 最小化均方误差 (MSE) :

$$\sum (\hat{Y} - Y)^2$$

4.概率建模

4.1 概率建模是一种生成模型.

机器学习中, 任务大致分为:

- **判别模型(discriminator)**
 - 任务是接收输入x, 给出一个判别输出y.
- **生成模型(generator)**
 - 任务是自动地(或者接收一些参数输入后)生成**优质**的输出x(比如图像).
- 其他的. 比如AE(autoencoder)自编码器. 它算是一个**重建任务**.

我们常见的各种用损失函数L的判别/分类模型就是判别器; GAN模型就是一个判别器D一个生成器G.

概率生成器/概率建模任务, 它是**生成模型**的一种.

概率建模大致分为两类:

- 1. 显式概率建模

- 从样本空间 X 中, 试图学习一个概率分布 $p_\varphi(x)$, 例如**高斯混合模型GMM**.
- 使它和真正的 $p(x)$ 非常接近, 我们可以从训练好的概率分布 p 中采样.
- 最终能: 得到模仿整个样本空间的输出.
- 显式概率建模的**目标函数**一般为数据的**最大化对数似然**, **MLE**:

- $$MLE = \max_{\theta} \sum_i^N \log p_{\theta}(x_i)$$

- 其思路容易理解: MLE越大, 可以近似认为:

- $$\sum_i^N p_{\theta}(x_i)$$

- 即把所有输入变量填入我们训练的概率分布, 概率和越大. 显然, 只有

- 2. 隐式概率建模(更常见)

- 样本空间 X 存在一个能描述 x 的特征的隐变量空间 Z . 比如, 看似多样的小动物图片集 X , 每个图片都属于具体的小动物类别. 我们试图学习一个**带隐变量**的分布 $p_\varphi(x|z)$. 例如**VAE, GMM, DDPM**.
- 使它和真正的 $p_\varphi(x|z)$ 非常接近. 给出我们期望的隐变量 $z = z_0$, 从 $p_\varphi(x|z)$ 中采样得到期望的隐变量为 $z = z_0$ 时的输出 x .
- 最终能: 模型在人为指定隐变量 z_0 (比如输入提示词 dog)后, 生成模仿整个样本空间中那些满足 $z(x_0)=z_0$ 的 x_0

(其中 φ 是可以训练的参数. 模型训练好开始用的时候, φ 是常数定值.)

例如:

图片生成模型就是一个隐变量概率建模: (输入 dog , 生成小狗图像, 输入 cat , 生成小猫图像...)可以看成一个隐变量概率建模问题: 所有的小狗图像为样本空间 X , 所有的提示词为隐变量空间 Z . 给出 $z = z_0$ 后, 模型 $p_\varphi(x|z)$ 会从 $p_\varphi(x|z_0)$ 这个分布中采样, 给你返回采样的结果图片.

4.2 显式概率建模

4.2.1 对显式概率建模: 使用MLE为目标函数

我们已经预设了一个确定形式的含参数的概率分布函数 $p_\theta(x)$.

我们可以取概率建模的目标函数为**最大对数似然MLE**. 核心思想: 找到最可能生成训练数据的分布 p .

提醒一下, 一般的显式概率建模, 比如最简单的, x 服从一个未知 μ, σ 的高斯分布. 本质上是一个统计建模任务, 是基础的机器学习, 没有神经网络. 我们所要做的就是:

- 取一个样本集 $\{x_i\}$,
- 然后求此时的 $L(\theta)$ 的最大值,
- 得到对应的 $\hat{\theta}$, 就是我们的极大似然估计参数. 于是整个模型结束了. 求 $L(\theta)$ 极大值, 显然就是求"选一个 θ , 让我再抽一个相同容量样本, 结果和这个训练集样本完全一样的概率最大". 这样显然意味着此时的分布

模型 p_θ 可以最好代表我们的训练集样本.

4.2.* (补)maximum likelihood estimator, 极大似然估计, MLE

likelihood function, 似然函数

设总体为X. 先验地认为 $X \sim p_\theta(x)$. 取出一个容量为n的样本 x_i .

则, 这个样本被取出的概率(换句话说, 我们再抽一个容量为n的样本, 和这个一样的概率为):

$$L(\theta) = \prod_i^n p(x_i | \theta)$$

称其为**似然函数**.

极大似然估计的思想就是想让这个 $L(\theta)$ 最大.

意味着最能表现我们到目前为止的抽样.

$L(\theta)$ 这个连乘很难算所以我们一般取个对数, 于是也就成了对数极大似然估计.

4.3 隐变量概率建模-原理: 变分推导

4.3.* 概率probability和似然likelihood的区别:

概率是 $P(x|\theta)$, 给定参数 θ 后样本x发生的概率 似然是 $L(\theta|x)$, 给定样本x, 求**参数 θ 让这个x出现**的概率.

4.3.1 隐变量概率建模的目标函数

在隐变量情境下, 描述整个样本空间的结构, 其实是x和z的联合分布 $p(x,z)$.

(描述整个图片集的结构, 既有每个图片x向量, 又有每个图片属于哪个小动物的onehot向量z.)

在隐变量概率建模中, 我们如果也取**目标函数为对数似然 (log marginal likelihood)**: (N 为训练集样本容量)

$$\max_{\theta} \sum_i^N \log p_{\theta}(x_i)$$

会发现此时这个公式通常**难以直接计算**, 因为 :

在隐变量建模中, $p(x)$

$$p(x) = \int p(x, z) dz = \int p(x|z)p(z) dz$$

这里的积分在高维空间上可能很难求解, 因此我们**无法直接优化** ($\log p(x)$).

于是, 我们引入一个**近似分布** ($q_{\phi}(z|x)$) 来逼近真实的后验分布 ($p(z|x)$), 并利用**变分推导**来优化它。

4.3.1.*变分下界 ELBO, 隐变量概率建模中重要的目标函数

我们看到上面的好多个模型, VAE,

4.4.2. ELBO 的推导

我们从对数边缘似然开始：

$$\log p(x) = \log \int p(x, z) dz$$

在这个公式中，我们加上一个辅助的近似后验分布 ($q_{\phi}(z|x)$) 并重写：

$$\log p(x) = \log \int q_{\phi}(z|x) \frac{p(x, z)}{q_{\phi}(z|x)} dz$$

利用 Jensen's Inequality (Jensen 不等式)，我们得到： $\log p(x) \geq \mathbb{E}_{q(\phi)(z|x)} [\log p(x, z) - \log q(\phi)(z|x)]$ 这个不等式右侧的部分就是**ELBO (变分下界)**： $\mathbb{E}_{q(\phi)(z|x)} [\log p(x|z)] - D_{KL}(q(\phi)(z|x) || p(z))$

4.4.2. ELBO 直观理解

ELBO 由两项组成：

1. **重构误差**： $\mathbb{E}_{q(\phi)(z|x)} [\log p(x|z)]$

- 这项衡量 (z) 生成的 (x) 有多接近原始数据，类似 VAE 的重构损失。

2. **KL 散度**：

$$D_{KL}(q_{\phi}(z|x) || p(z))$$

- 这项让近似后验 ($q(\phi)(z|x)$) 逼近先验 ($p(z)$)，防止隐变量 (z) 变得太自由（避免 overfitting）。

最终，优化 ELBO 既可以让 VAE 生成的数据更逼近真实数据，又能保持 (z) 具有良好的先验分布。

4.4 隐变量概率推理-几种模型

首先**概率建模**是大框架, **变分推理**是**概率建模**的其中一个思想:

首先概率建模(如VAE, GMM, DDPM)干的事: 建模输入数据的概率分布 $p(x)$ 或者带隐变量

这是从物理学中的**变分**概念得到的. 即让一个函数去逼近另一个函数, 在数学上是同一件事: 是自变量为函数的求最小值问题(泛函问题). 我们一点一点解释...

5.1 隐变量图模型:

首先解释一下隐变量(隐变量图模型). 隐变量就是(计算机)无法直接观测的变量. 比如如下场景: 有很多小动物图片向量 x . 它们所属于的动物物种看作onehot特征向量 z , 则是隐变量: 必须训练一个好的分类器来得到 $x \rightarrow z$.

x 为输入图片(被观测变量); z 为表示分类的onehot向量(类别为猫). 它是隐变量(无法直接**观测**到, 需要从被观测变量 x 中推理). 于是 z 和 x 组成**隐变量图模型**.

又或者: 假设你是一个视频网站的用户, 平台希望给你推荐新电影。 观测变量 x : 你看过的电影、评分。 隐变量 z : 你的兴趣类别, 比如你更喜欢科幻片还是喜剧片?

一般, 隐变量的分布 $p(z)$ 没有办法直很容易直接得到.

更加复杂的现实情况:

5.2 高斯混合模型 (GMM)

把这K个高斯分布换成其他复杂的由几个参数决定的分布的线性组合, 便称为更普遍的混合分布模型.

我们有很多数据点(简单起见, 一维数值.), 假设它们是从好几个高斯分布

$N_1(\mu_1, \sigma_1^2), N_2(\mu_2, \sigma_2^2), N_3(\mu_3, \sigma_3^2), \dots, N_K(\mu_K, \sigma_K^2)$ 中抽取出来的, 但是我们不知道具体 x_i 是哪个分布出来的. 或者说, 不知道 $p(z)$. 即任意一个点是从 $z=i$ 个高斯分布中来的概率. 于是我们可以认为每个点来自的高斯分布序号是个 **隐变量** z , $z=1, 2, 3, \dots$

于是, 由全概率公式,

$$p(x) = \sum_{z=1}^K p(x | z)p(z)$$

令 $z=i$ 的概率 $p(i) = \pi_i$, 则有:

$$p(x) = \sum_{z=1}^K p(x | z)\pi_i$$

π_i 是第 i 个高斯分布 N_i 的权重系数. 于是 $p(x)$ 被表示为多个高斯分布的线性组合.

开始训练!

现在只需要选个损失函数开始训练, 学习 π_i 这些权重系数, 最后得到一组 π_i , 即得到了 $p(z)$ 这个隐变量分布.

事实上**GMM**不是典型的隐变量概率模型问题, 因为虽然说是隐变量, 但是 $p(x)$ 已经可以解析地表达成含参数的具体分布了, 隐变量 z 的地位已经退化为了 $p_\theta(x)$ 的参数. 我们对**GMM**不需要变分原理, 直接把 z 看成参数, 求 z 的极大似然分布即可.

↓↓**隐变量模型中, 几种概率解释:**

后验概率: $p(z|x) = \frac{p(x,z)}{p(x)} = \frac{p(x|z)p(z)}{p(x)}$, 是 x 的函数.

即"得到特定图像 x 后, 它属于哪种类别小动物的概率".

$p(z|x)$ 的具体形式

这正是图像处理中我们最终想要的东西.

先验概率: 即 $p(z)$.

这个分布是在观察数据 x 之前就假设的, 所以称为 **先验(Prior)**

$p(z)$ 即为"各种类别小动物在整个图片库中的分布", 也是我们要学习的一组参数.

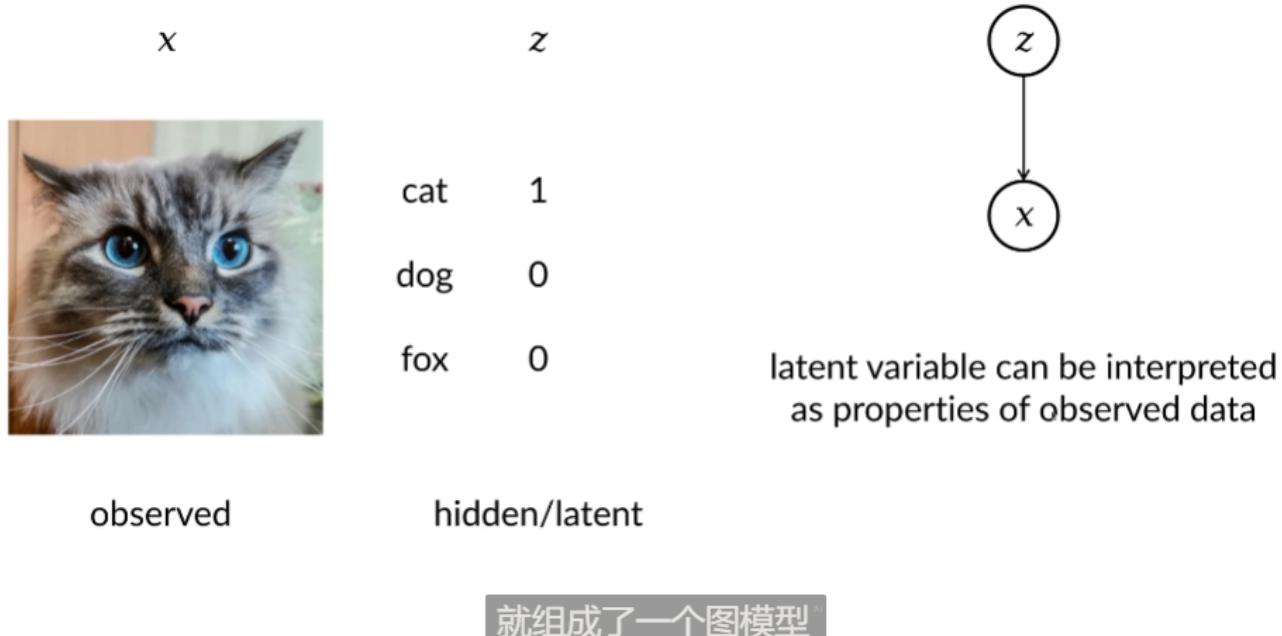
联合分布: 即 $p(x, z)$.

边缘概率: 即 $p(x)$. 概率论中从 $p(x,y)$ 积分(或全概率加起来)掉其中一个变量后就称为 x 或 y 的边缘概率:

- $p(x) = \int p(x, y) dy$
- $p(y) = \int p(x, y) dx$

Latent Graphical Model

bilibili



5.3. VAE的引子: 自编码器, AE, autoencoder

AE的数学结构

它主要包含两个部分: **编码器 encoder** 将输入数据 x (图片向量)映射到一个隐空间 Z (图片的种类onehot空间). 即: $z = f_{enc}(x), Z -> X$ **解码器 decoder** 从隐空间 Z 中恢复出一个输入数据 x' . 即: 即: $x = f_{dec}(z), X -> Z$

损失函数一般选择**最小化重构误差**: $L = ||x - x'||^2$

AE能干啥?

从数学结构上, AE干的事是从输入数据中学到一个确定的隐变量向量 z , 然后可以用这个固定的 z 还原出一个"输出图片". 这个过程只是个**压缩重建模型**, 属于**重建任务**, 一般不称为**生成模型**. 毕竟它没有进行概率建模, 生成的数据是固定的. 如果设计 z 向量很长, 比如是输入图形 x 的一半, AE此时就是一个**图片压缩-解压器**.

5.4. 变分自编码器VAE, variational autoencoder

普通的AE的隐变量 z 没有**概率分布的约束**.

VAE:

- 强制隐变量 z 的后验分布 $p(z|x)$ 服从一个分布 $q_\phi(z|x)$, 而不是简单的定值. (一般是标准正态分布)
- 然后用变分推理来学习 z 的概率分布, 而不是直接学习一个固定的 $z(x)$.

即: 学习那个分布 $q_\phi(z|x)$ 的参数 ϕ .

比如可以让 $q_\phi(z|x) = N(\mu(\phi), \sigma^2(\phi))$, 其中 $\mu(\varphi)$ 可简单取: $\mu(\varphi) = \varphi$,

于是 $q_\phi(z|x)$ 就是一个包含参数 φ 的分布了.

于是:

VAE的encoder输出不再是一个 z , 而是一个 z 的概率分布 $q(z|x)$. 然后从这个分布中随机采样一个 z , 放入解码器:

$$x' = f_{dec}(z)$$

VAE的损失函数称为**变分下界 Evidence Lower Bound**:

4. ELBO 在扩散模型中的作用

虽然扩散模型不像 VAE 那样显式地引入隐变量 (z), 但它也用到了类似的 ELBO 公式 : $\mathcal{L} = \mathbb{E}_{q(x_t | x_0)} [\log p(x_0 | x_t)] - D_{\text{KL}}(q(x_T) \| p(x_T))$ 这里 :

- ($q(x_t | x_0)$) 是加噪过程, 相当于变分推理中的近似分布 ($q(\phi)(z | x)$)。
- ($p(x_0 | x_t)$) 是去噪网络的估计, 相当于 VAE 中的解码器 ($p(\theta)(x | z)$)。
- KL 散度项保证了模型学习的逆扩散过程符合数据分布。

所以, **ELBO 在 VAE 和扩散模型中都扮演了优化角色, 只是方式不同。** $\mathcal{L} = 1/2 \|\varepsilon_\theta(x_t, t) - \varepsilon\|^2$

x_0

$x_{t-1} \rightarrow x_t$

5.6 自回归 AR, autoregressive

自回归在机器学习中, 指当前输出依赖于先前输出.

比如马尔可夫过程显然是一种自回归.

- 马尔可夫过程定义为这样的过程: 第 $t+1$ 步状态只和第 t 步状态有关, 和过去状态无关. 概率论上就是一组随机变量序列 $s_{t=0}^\infty$, 它满足: 每个随机变量的条件概率只会包含上一个.
即: $P(s_{t+1} | s_t, s_{t-1}, \dots, s_0) = P(s_{t+1} | s_t)$

6 DDPM, 去噪扩散概率模型, 扩散模型.

数学过程:

1.前向加噪

取 x_0 为输入图像. T 为总步长, 一般设置为 $T=1000$.

我们进行 T 步加噪. 每一步采样一个正态噪声 ε , 然后按照系数 β_t 累加到上一步给出的图像:

$$x_t = \sqrt{\beta_t} \varepsilon_t + \sqrt{1 - \beta_t} x_{t-1}$$

$\$$

其中 β_t 的选择一般是 $0 < \beta_1 < \beta_2 < \dots < \beta_T < 1$. 即扩散速度越来越快了!

上述递推公式不断展开, 得到:

- 注意到两个独立的随机变量相加, 可以卷积起来成为一个新的随机变量. 两个独立的正态随机变量有:
 $N(\mu_1, \sigma_1^2) + N(\mu_2, \sigma_2^2) = N(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$

\$\$

\begin{aligned}

$$x_t &= \sqrt{\beta_t} \varepsilon_t + \sqrt{1-\beta_t} x_{t-1} \\ &= \sqrt{\beta_t} \varepsilon_t + \sqrt{1-\beta_t} (\sqrt{\beta_{t-1}} \varepsilon_{t-1} + \sqrt{1-\beta_{t-1}} x_{t-2}) \\ &\dots = \sqrt{1-\bar{\alpha}_t} \varepsilon + \sqrt{\bar{\alpha}_t} x_0$$

\end{aligned}

\[10pt]

其中 $\alpha_t \equiv \beta_t$, $\bar{\alpha}_t \equiv \prod_i^t \alpha_i$

$$x_t = \sqrt{1 - \alpha_t} \times \epsilon_t + \sqrt{\alpha_t} \times x_{t-1}$$



$$x_t = \sqrt{1 - \bar{\alpha}_t} \times \epsilon + \sqrt{\bar{\alpha}_t} \times x_0$$

$$\bar{\alpha}_t = \alpha_t \alpha_{t-1} \alpha_{t-2} \alpha_{t-3} \dots \alpha_2 \alpha_1$$

2. 反向去噪



$$x_t = \sqrt{1 - \alpha_t} \times \epsilon_t + \sqrt{\alpha_t} \times x_{t-1}$$

$$x_t = \sqrt{1 - \bar{\alpha}_t} \times \epsilon + \sqrt{\bar{\alpha}_t} \times x_0$$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$$P(x_{t-1}|x_t) = \frac{P(x_t|x_{t-1})P(x_{t-1})}{P(x_t)}$$

上式为了严谨，加上条件 x_0 (x_0 相同的时候，可以忽视 x_0)

带入具体概率密度函数：

$$P(x_t|x_{t-1}, x_0) = \frac{1}{\sqrt{2\pi}\sqrt{1-a_t}} e^{-\frac{1}{2} \frac{(x_t - \sqrt{a_t}x_{t-1})^2}{1-a_t}} \leftarrow N(\sqrt{a_t}x_{t-1}, 1-a_t)$$

$$P(x_t|x_0) = \frac{1}{\sqrt{2\pi}\sqrt{1-\bar{a}_t}} e^{-\frac{1}{2} \frac{(x_t - \sqrt{\bar{a}_t}x_0)^2}{1-\bar{a}_t}} \leftarrow N(\sqrt{\bar{a}_t}x_0, 1-\bar{a}_t)$$

$$P(x_{t-1}|x_0) = \frac{1}{\sqrt{2\pi}\sqrt{1-\bar{a}_{t-1}}} e^{-\frac{1}{2} \frac{(x_{t-1} - \sqrt{\bar{a}_{t-1}}x_0)^2}{1-\bar{a}_{t-1}}} \leftarrow N(\sqrt{\bar{a}_{t-1}}x_0, 1-\bar{a}_{t-1})$$

$$P(x_{t-1}|x_t, x_0) = \frac{P(x_t|x_{t-1}, x_0)P(x_{t-1}|x_0)}{P(x_t|x_0)}$$



大白话AI

就可以把它们写成正态分布的概率密度函数形式

$$P(x_{t-1}|x_t, x_0) = \frac{\frac{1}{\sqrt{2\pi}\sqrt{1-a_t}} e^{-\frac{1}{2} \frac{(x_t - \sqrt{a_t}x_{t-1})^2}{1-a_t}} \frac{1}{\sqrt{2\pi}\sqrt{1-\bar{a}_{t-1}}} e^{-\frac{1}{2} \frac{(x_{t-1} - \sqrt{\bar{a}_{t-1}}x_0)^2}{1-\bar{a}_{t-1}}}}{\frac{1}{\sqrt{2\pi}\sqrt{1-\bar{a}_t}} e^{-\frac{1}{2} \frac{(x_t - \sqrt{\bar{a}_t}x_0)^2}{1-\bar{a}_t}}}$$

$$\frac{1}{\sqrt{2\pi} \left(\frac{\sqrt{1-a_t} \sqrt{1-\bar{a}_{t-1}}}{\sqrt{1-\bar{a}_t}} \right)} e^{-\frac{\left(x_{t-1} - \left(\frac{\sqrt{a_t}(1-\bar{a}_{t-1})}{1-a_t} x_t + \frac{\sqrt{\bar{a}_{t-1}}(1-a_t)}{1-a_t} x_0 \right) \right)^2}{2 \left(\frac{\sqrt{1-a_t} \sqrt{1-\bar{a}_{t-1}}}{\sqrt{1-\bar{a}_t}} \right)^2}}$$

$$\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_{t-1}-\mu)^2}{2\sigma^2}}$$

P(x_{t-1}|x_t, x₀) = $\frac{\frac{1}{\sqrt{2\pi}\sqrt{1-a_t}} e^{-\frac{1}{2} \frac{(x_t - \sqrt{a_t}x_{t-1})^2}{1-a_t}} \frac{1}{\sqrt{2\pi}\sqrt{1-\bar{a}_{t-1}}} e^{-\frac{1}{2} \frac{(x_{t-1} - \sqrt{\bar{a}_{t-1}}x_0)^2}{1-\bar{a}_{t-1}}}}{\frac{1}{\sqrt{2\pi}\sqrt{1-\bar{a}_t}} e^{-\frac{1}{2} \frac{(x_t - \sqrt{\bar{a}_t}x_0)^2}{1-\bar{a}_t}}}$

于是我们得到了给定x_t时,x_{t-1}的概率分布

$$P(x_{t-1}|x_t, x_0) = \frac{P(x_t|x_{t-1}, x_0)P(x_{t-1}|x_0)}{P(x_t|x_0)}$$

大白话AI

表示在相同的x₀(条件)下, 实际上可以忽视它们

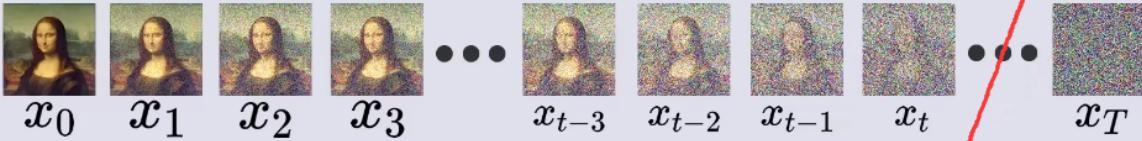
$$\frac{1}{\sqrt{2\pi} \left(\frac{\sqrt{1-\bar{a}_t} \sqrt{1-\bar{a}_{t-1}}}{\sqrt{1-\bar{a}_t}} \right)} e^{\left[-\frac{\left(x_{t-1} - \left(\frac{\sqrt{\bar{a}_t}(1-\bar{a}_{t-1})}{1-\bar{a}_t} x_t + \frac{\sqrt{\bar{a}_{t-1}}(1-a_t)}{1-\bar{a}_t} x_0 \right) \right)^2}{2 \left(\frac{\sqrt{1-\bar{a}_t} \sqrt{1-\bar{a}_{t-1}}}{\sqrt{1-\bar{a}_t}} \right)^2} \right]}$$

↙

$$P(x_{t-1}|x_t, x_0) \sim N \left(\frac{\sqrt{a_t}(1-\bar{a}_{t-1})}{1-\bar{a}_t} x_t + \frac{\sqrt{\bar{a}_{t-1}}(1-a_t)}{1-\bar{a}_t} x_0, \left(\frac{\sqrt{1-a_t} \sqrt{1-\bar{a}_{t-1}}}{\sqrt{1-\bar{a}_t}} \right)^2 \right)$$


将 x_0 用 x_t 和一个正态分布随机变量 ϵ 替换:

$$x_t = \sqrt{1-\bar{a}_t} \times \epsilon + \sqrt{\bar{a}_t} \times x_0$$

$$x_0 = \frac{x_t - \sqrt{1-\bar{a}_t} \times \epsilon}{\sqrt{\bar{a}_t}}$$


$$P(x_{t-1}|x_t, x_0) \sim N \left(\frac{\sqrt{a_t}(1-\bar{a}_{t-1})}{1-\bar{a}_t} x_t + \frac{\sqrt{\bar{a}_{t-1}}(1-a_t)}{1-\bar{a}_t} \times \frac{x_t - \sqrt{1-\bar{a}_t} \times \epsilon}{\sqrt{\bar{a}_t}}, \left(\sqrt{\frac{\beta_t(1-\bar{a}_{t-1})}{1-\bar{a}_t}} \right)^2 \right)$$

$$P(x_{t-1}|x_t, x_0) \sim N \left(\frac{\sqrt{a_t}(1-\bar{a}_{t-1})}{1-\bar{a}_t} x_t + \frac{\sqrt{\bar{a}_{t-1}}(1-a_t)}{1-\bar{a}_t} x_0, \left(\frac{\sqrt{1-a_t} \sqrt{1-\bar{a}_{t-1}}}{\sqrt{1-\bar{a}_t}} \right)^2 \right)$$


最终得到不含 x_0 的, 已知 x_t 时 x_{t-1} 的概率分布.

```
## P\bigl(x_{t-1} \mid x_t, x_0\bigr) \sim \mathcal{N}!\Biggl( \sqrt{\frac{\alpha_t}{1-\bar{\alpha}_t}}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\frac{\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}}(1-\alpha_t)\frac{x_t - \sqrt{1-\bar{\alpha}_t} \times \epsilon}{\sqrt{\bar{\alpha}_t}}, \left( \sqrt{\frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}} \right)^2 \Biggr)
```

有了上式, 我们得到了从已知 x_t 推测 x_{t-1} 的概率分布.

实际上, 有了 x_t 后, 只要我们知晓当初从 x_0 添加噪声变成 x_t 的那个噪声 ϵ , 就可以确定出此时 x_{t-1} 的分布(注意还不能唯一确定).

- 后向去噪中, 从 x_T 抽样到 x_0 的过程中, 这个正态分布:

```
## P\bigl(x_{t-1} \mid x_t, x_0\bigr) \sim \mathcal{N}!\Biggl( \sqrt{\frac{\alpha_t}{1-\bar{\alpha}_t}}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\frac{\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}}(1-\alpha_t)\frac{x_0}{\sqrt{1-\bar{\alpha}_t}}, \left( \sqrt{\frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}} \right)^2 \Biggr)
```

的方差会越来越小, 直到接近0; 期望中心则从原点0渐渐偏移到 x_0 .

3. 构建神经网络

现在有了：

- 前向加噪：用一个正态分布抽样噪声 ε 即可从 x_0 变为 x_t ；

$$x_t = \sqrt{1 - \bar{\alpha}_t} \varepsilon + \sqrt{\bar{\alpha}_t} x_0$$

- 后向去噪：只要知道 x_t 和让它从 x_0 变成 x_t 的噪声 ε ，就可以得到 x_{t-1} 的概率分布；

$$\begin{aligned} & \$\$ P(x_{t-1} | x_t, x_0) \sim \mathcal{N}(\sqrt{\frac{\bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}} x_t + \sqrt{\frac{1 - \bar{\alpha}_{t-1}}{\bar{\alpha}_t}} \varepsilon, \epsilon^2) \\ & \$\$ \end{aligned}$$

我们可以训练一个神经网络(一般选用UNET.)

- 这个网络的输入输出：

- 输入 x_t
- 输出预测噪声 ε_t
- (内部拟合结构暂时黑箱)

- 这个网络的训练过程：**

- 设定 $T=1000$ ，确定常数 α_t 。

- 数据准备：**

- 取一个训练样本 x_0 （如一张小猫图片）。
- 从均匀分布中随机采样 $t \in \{1, 2, \dots, T\}$ 。
- 采样一个标准高斯噪声 $\varepsilon_t \sim N(0, I)$ ，然后利用 **前向扩散公式**：

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \varepsilon$$

生成 x_t （即 x_0 的部分加噪版本）。

- 模型前向传播：**

- 输入 x_t 和 t 到模型 $\varepsilon_\theta(x_t, t)$ 。
- 模型的目标是 **预测噪声** ε_t ，即：

$$\varepsilon_\theta(x_t, t) \approx \varepsilon_t$$

- 计算损失函数：**

- 采用 **均方误差 (MSE)** 损失：

$$L = \mathbb{E}_{x_0, t, \varepsilon} [|\varepsilon - \varepsilon_\theta(x_t, t)|^2]$$

- 反向传播更新模型中的参数。

- 一轮训练结束。

- **这个网络的使用过程**

- 采样一个高斯噪声当作 x_T .
- 开始对 x_T 逐步去噪: 将其作为输入给到模型. 模型输出 ε_T . 利用公式
$$\mathcal{P}(\varepsilon_t | x_0) \sim \mathcal{N}(\sqrt{\frac{\alpha_t}{\bar{\alpha}_t}}(x_0 - \bar{x}_t), \sqrt{\frac{\alpha_t}{\bar{\alpha}_t}}\varepsilon_t)$$
 得到 $P(\varepsilon_{T-1} | x_0) \sim \mathcal{N}(\sqrt{\frac{\alpha_{T-1}}{\bar{\alpha}_{T-1}}}(x_0 - \bar{x}_{T-1}), \sqrt{\frac{\alpha_{T-1}}{\bar{\alpha}_{T-1}}}\varepsilon_{T-1})$, 对其抽样得到一个 x_{T-1} .
- 然后把得到的 x_{T-1} 输入给模型, 模型输出 ε_{T-1} .
- 继续操作...直到抽样得到一个 x_0 .
- x_0 就是生成的图片了.

现在我们来看模型UNET具体的结构.

4.UNET模型

5. free-guidance的有条件的扩散模型

- **这个网络的训练过程:**

- 设定 $T=1000$, 确定常数 α_t 。(同DDPM)
- **数据准备 :**
 - 取一个训练样本 x_0 (如一张小猫/小狗/小猪图片), 以及其对应标签 y . 比如 $y=1$ 代表小猫, $y=2$ 代表小狗.
 - p_{drop}
 - 从均匀分布中随机采样 $t \in \{1, 2, \dots, T\}$.
 - 采样一个标准高斯噪声 $\varepsilon_t \sim N(0, I)$, 然后利用 **前向扩散公式** :

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \varepsilon$$

生成 x_t (即 x_0 的部分加噪版本)。

- **模型前向传播 :**

- 输入 x_t 和 t 到模型 $\varepsilon_\theta(x_t, t)$ 。
- 模型的目标是 **预测噪声** ε_t , 即 :

$$\varepsilon_\theta(x_t, t) \approx \varepsilon_t$$

- **计算损失函数 :**

- 采用 **均方误差 (MSE)** 损失 :

$$L = \mathbb{E}_{x_0, t, \varepsilon} \left[|\varepsilon - \varepsilon_\theta(x_t, t)|^2 \right]$$

- **反向传播更新模型中的参数.**

- **一轮训练结束。**

- 这个网络的使用过程

- 采样一个高斯噪声当作 x_T .
- 开始对 x_T 逐步去噪: 将其作为输入给到模型. 模型输出 ε_T . 利用公式 $P(\bar{x}_{t-1} | x_t, x_0) \sim \mathcal{N}(\sqrt{\frac{\alpha_t}{\alpha_{t-1}}(1 - \bar{\alpha}_{t-1})} x_t + \sqrt{\frac{\bar{\alpha}_t}{\alpha_{t-1}}\bar{\alpha}_{t-1}}\varepsilon_T, \sigma^2)$ 得到 $P(x_{t-1} | x_t, x_0)$, 对其抽样得到一个 x_{t-1} .

等等使用这个网络生成图片的时候(采样环节)我们没有 x_0 呀! 我们是想得到一个 x_0 呀. 此时我们只能用前向扩散公式给出 x_0 的估计值:

根据前向扩散:

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon \Rightarrow x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon)$$

于是我们可以估计:

$$\hat{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon_\theta(x_t, t))$$

然后把这个估计值 x_0 带入到上式分布, 然后采样.

- 然后把得到的 x_{T-1} 输入给模型, 模型输出 ε_{T-1} .
- 继续操作...直到抽样得到一个 x_0 .
- x_0 就是生成的图片了.

扩散模型 (Denoising Diffusion Probabilistic Model, DDPM) 属于**隐变量概率建模**的一种。它的核心思想是引入一个**隐变量** (z_t) (可以看作是噪声注入的中间状态), 并通过逐步去噪的方式恢复数据 (x)。

DDPM 如何符合隐变量概率建模框架?

隐变量概率建模的核心是**学习一个联合分布** $p_\phi(x, z)$, 然后通过**条件概率** $p_\phi(x | z)$ 生成样本。

在 DDPM 里, 我们可以把整个**前向过程 (加噪)** 和**逆向过程 (去噪)** 看作是在一个**隐变量空间** $z_{t=0}^T$ 中进行变换:

1. 前向扩散过程 (加噪) :

- 这是一个**马尔可夫过程**.
- 给定真实数据 (x_0), 我们逐步加入噪声, 使其变成一个标准正态分布的随机变量 (z_T).
- 这可以理解为**定义了一个隐变量模型**, 其中 (x_0) 是最终目标, z_T 是最强噪声状态: $q(z_t | x_0) = \mathcal{N}(z_t; \sqrt{\bar{\alpha}_t}x_0, (1 - \bar{\alpha}_t)^{-1})$

- 其中 α_t 是一个控制噪声水平的参数。

1. 逆向去噪过程（解码）：

- 我们训练一个神经网络来学习去噪过程，即估计每一步的条件概率 ($p_{\phi}(z_{t-1} | z_t)$)：

$$p_{\phi}(z_{t-1} | z_t) = \text{mathcal}{N}(z_{t-1}; \mu_{\phi}(z_t, t), \Sigma_{\phi}(t))$$
- 这个过程可以理解为，我们在尝试重建 (x_0)，类似于 VAE 的解码过程。

最终，我们可以从一个随机噪声 ($z_T \sim \mathcal{N}(0, I)$) 开始，按照 ($p_{\phi}(z_{t-1} | z_t)$) 逐步去噪，最终生成一个样本 (x_0)。

DDPM vs. VAE

	VAE	DDPM
隐变量	低维 (z) (通常是 64D-256D)	高维 (z_t) (与数据维度相同)
隐变量分布	($p(z) = \mathcal{N}(0, I)$)	($p(z_T) = \mathcal{N}(0, I)$)
生成方式	直接从 ($p(x z)$) 采样	通过逐步去噪 ($p(z_{t-1} z_t)$) 生成
优化目标	变分下界 ELBO	变分下界 ELBO + 逐步去噪的误差

结论

是的！DDPM 符合隐变量概率建模的定义，它的隐变量 (z_t) 代表不同噪声水平的数据状态，最终目标是从一个标准正态分布 ($z_T \sim \mathcal{N}(0, I)$) 逐步还原出目标数据 (x_0)。

不同于 VAE 直接在低维隐变量 (z) 上建模，DDPM 在数据的高维空间里直接建模一个马尔可夫链的隐变量过程，这也是它生成质量更高的原因之一。

10. 还没归纳的草稿：

问题一，隐变量一般是啥，为啥 $p(z)$ 是先验的，一开始都认为规定它的分布？

因为不这样就没法进行下去了...

现实中的图像生成模型，最终感兴趣的确实是 $p(x | z)p(x|z)p(x | z)$ ，因为我们希望根据用户提供的 zzz 来生成对应的图片。

但你的直觉——“输入 dog 然后生成狗的图片”——其实不是直接使用 dog 这个标签，而是模型隐式地把某些潜在变量的方向学成了‘狗’的概念。

1. 现实中的生成模型如何使用 $p(x | z)p(x|z)p(x | z)$

我们关心的其实是：

$p(x | z) = \text{"给定隐变量 } z, \text{ 生成图像 } x \text{ 的概率"} p(x | z) = \text{text("给定隐变量 } (z), \text{ 生成图像 } ((x)) \text{ 的概率")}$
 $p(x | z) = \text{"给定隐变量 } z, \text{ 生成图像 } x \text{ 的概率"}$

这意味着：

- 如果我们能控制 z_{zz} , 就能控制生成的图片 x_{xx} 。
 - 但 z_{zz} 不是直接的文本标签, 而是高维的潜在变量 !
 - 现代模型让 z_{zz} 自动学习如何表示“dog”这个概念, 而不是手动规定。
-

2. 为什么你会觉得“提示词 = z ”？

你的直觉很合理！因为：

- 现实中的图像生成模型（如 DALL-E、Stable Diffusion）确实支持“输入文本 -> 生成图片”
- 但它们不是简单地把“dog”当作 z_{zz} , 而是用了一个文本编码器（如 CLIP）, 把“dog”变成一个高维向量 z_{zz}
- 这个 z_{zz} 进入 $p(x|z)p(x|z)p(x|z)$, 然后生成对应的图片

所以在 文本到图像的模型（如 Stable Diffusion）中：

- 用户输入的文本 dog 并不是直接的 z_{zz} , 而是通过文本编码器得到一个 $z_{text} \backslash \text{text}\{text\} z_{text}$
- 这个 $z_{text} \backslash \text{text}\{text\} z_{text}$ 是连续的, 不是离散的类别变量
- 生成模型最终学习到 “某些方向的 $z_{text} \backslash \text{text}\{text\} z_{text}$ 代表‘狗’”

换句话说, 你输入 dog, 其实是 把它转换成一个“狗相关的潜在变量 z_{zz} ”, 然后用 $p(x|z)p(x|z)p(x|z)$ 采样图片。

3. 现实中的 $p(x|z)p(x|z)p(x|z)$ 是怎么工作的？

不同模型实现 $p(x|z)p(x|z)p(x|z)$ 的方式不同：

生成模型

隐变量 z_{zz}

解释

VAE

z_{zz} 是一个高斯分布的连续向量

编码 x_{xx} 为 z_{zz} , 再从 ($p(x$

GAN

z_{zz} 是一个随机高斯向量

直接用 z_{zz} 作为输入, 生成 x_{xx}

DDPM

z_{zz} 是噪声

通过去噪迭代生成 x_{xx}

Stable Diffusion

zzz 是“文本编码” + 视觉噪声

结合文本信息，引导去噪，生成 xxx

所以： 我们最终关心的是 $p(x|z)p(x|z)p(x|z)$ ，因为这决定了“如何根据 zzz 生成 xxx”

zzz 不是离散类别标签，而是高维潜变量（可以是风格、内容、构图信息等）

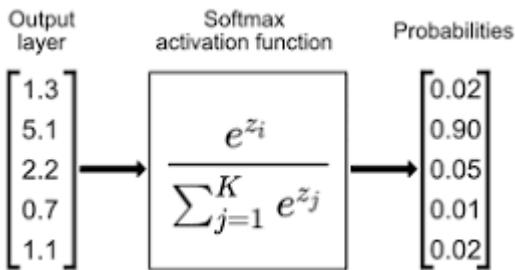
在文本到图像的任务中，文本会被转换成一个潜变量 $z_{text} \sim p(z|x)$ ，然后用于控制 $p(x|z)p(x|z)p(x|z)$

4. 总结

- 你的思路“输入 dog，然后生成狗的图片”是对的，但 dog 本身不会直接成为隐变量 zzz
- 现实中，dog 会被文本编码器转换成一个高维向量 $z_{text} \sim p(z|x)$ ，然后进入 $p(x|z)p(x|z)p(x|z)$
- 我们感兴趣的是 $p(x|z)p(x|z)p(x|z)$ ，因为它定义了“如何根据 zzz 生成 xxx”
- 现代生成模型学到的是：“zzz 的某个方向 ≈ 狗的概念”，而不是手工设定“狗 = zzz 的某个值”

5. 常用函数&概念

softmax(): 归一化指数函数，把任意实数向量映射到(0,1)区间，且和为1的概率分布。是常用的激活函数。在分类器神经网络中很常见。



输入一个向量x后，输出向量softmax(x)可以看成一个保证每个元素的相对大小地位，整体和为1的向量，可以当作概率分布用(如果x的某个分量很大意味着更强更容易被选择的话)。

9. transformer: CODEC(COmpressor-DECompressor, 编解码器) Assisted Matrix Condensing, 编解码器辅助的矩阵压缩。这是专门让transformer模型更小更快的技术。

8. transformer

核心机制是自注意力机制。

和上述的FNN例子模型, transformer模型:

输入是一句话 -> 一个矩阵 $X \in \mathbb{R}^{n \times d}$, 其中 n 是句子的长度 (Token 数量), d 是每个词的特征维度.

transformer的每一层有三个矩阵:

查询向量矩阵 W^Q (*query*), 键向量矩阵 W^K (*key*) $\in \mathbb{R}^{d \times d_k}$, 通常 $d_k = d$

值向量矩阵 W^V (*value*) $\in \mathbb{R}^{d \times d_V}$, 通常 $d_V = d$

它们分别作用在 X 上, 得到三个矩阵 **Q, K, V**:

- $Q = XW^Q$ (查询: 我要找什么?)
- $K = XW^K$ (键: 我这里有什么?)
- $V = XW^V$ (值: 我实际的内容是什么?)

可以把三个矩阵 **Q, K, V** 看作是 **把原始信息 X 投射到了三个不同的空间里**.

然后, 还要计算出 **S, P, Z** 三个矩阵:

- $S = QK^T$. S 为 Scores, 相关性矩阵. 它表示词与词之间的两两关系.
- $P = \text{softmax}\left(\frac{S}{\sqrt{d_k}}\right)$. P 是 S 的归一化. 其中 d_k 是 Dimension of Keys, 在最基础的 transformer 中, 我们只有一个注意力头, $d_k = d$.
- 加权求和: $Z = P \cdot V$. 最终的输出 Z 是用这些权重对内容 V 进行加权平均. Z 就是最后的输出矩阵.

上述式子写成一步就是:

$$\text{输出 } Z = \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

这就是 transformer 的一层注意力层. 事实上 transformer 有 encoder+decoder 两部分, 它们都包含多层上述的注意力层.

transformer 可以做预测/分类, 也可以做生成.

- GPT(生成式): 根据前面的词预测下一个词.
- BERT(理解式): 根据前面的词和后面的词, 预测中间被遮挡的词.

transformer 的损失通常是交叉熵.

transformer 的典型样本:

是一个 Pair, 由由“源序列”和“目标序列”组成。

- **源序列 (Source Sequence)** : 你想让模型理解的内容。
- **目标序列 (Target Sequence)** : 你希望模型生成的内容。

现在我们来看数学过程的意义.

首先注意力就是为了理解当前的词, 我应该对句子里的哪些词分配多少权重.

训练之前一开始, W^Q, W_K, W_V 都是随机数. 通过标签训练的反向传播更新后, 这些矩阵会学到 **如何根据当前词的含义, 找到句子里相关的词**.

encoder层:

就是多个注意力子层.

decoder层:

decoder的每个块包含两个不同的注意力层(掩码注意力层+交叉注意力层)+一个FFN层(前馈网络层).

掩码注意力层:

它是一个上述注意力层, 但做出改动:

当接收上一层的输入 X , 运算得到 Q, K, V 后,

继续运算得到 $S = QK^T$.

然后, 对这个 S 做一个**掩码操作(masking)**:

把未来词的分数设为 $-\infty$, 然后再做softmax归一化. 这样就保证了**当前词只能关注前面的词, 不能看到未来的词**. 这是生成模型必须的约束条件.

假设我们的目标序列有 3 个词 : $[t_1, t_2, t_3]$ (对应 "I", "love", "cats") 。

得到的 S 矩阵是一个 3×3 的矩阵, 代表词与词之间的相关性 :

	K1 (I)	K2 (love)	K3 (cats)
Q1 (I)	S11	S12	S13
Q2 (love)	S21	S22	S23
Q3 (cats)	S31	S32	S33

然后是掩码操作: 在进行 softmax 之前, 我们会给这个矩阵叠加一个掩码矩阵。我们会把右上角 (即当前词之后的词) 的得分全部替换成 $-\infty$ (负无穷) :

	K1	K2	K3
Q1	S11	$-\infty$	$-\infty$
Q2	S21	S22	$-\infty$
Q3	S31	S32	S33

做 softmax 时, $e^{-\infty}$ 会趋近于 0。

交叉注意力层

一个实际训练例子讲解:

我们希望一个初始transformer模型学会翻译中文到英文.

一个样本:

- 原序列: 我 爱 猫
- 目标序列: <sos> I love cats 其中 <sos> 为序列开始符号.

把它们 embedding(词嵌入), 每个 token 都变成为长度为 d 的向量.

- $X_{enc} = [0.1, -0.2, 0.5, \dots]$ (长度为 $3d$, 即 $3d$ 矩阵.)
- $X_{dec} = [<\text{sos}>, \text{I}, \text{love}, \text{cats}]$ (长度为 $4d$, 即 $4d$ 矩阵.)

对于 encoder 部分, 输入为 X_{enc} . 最终输出为 Z_{enc} , 一个 $3 \times d$ 的矩阵. 也被称为特征向量. 这个向量包含了对输入序列的理解.

对于 decoder-掩码自注意力层部分, 输入为目标序列 X_{dec} . 这一层会对矩阵 S 使用掩码. 输出为 Z_{masked} , 一个 $4 \times d$ 的矩阵.

对于 decoder-交叉注意力层部分,

- 本层的 Q 矩阵: $Q = Q_{decoder} = Z_{masked}$. (比如: "我已经翻译出 I 了, 接下来我想找动词").
- 本层的 K 矩阵: 来自 Encoder 的输出 $K = K_{encoder} = Z_{enc}$ (原句的全部信息) .
- **计算:** $S = Q_{decoder} \times K_{encoder}^T$
- **效果:** 模型拿着"接下来要找动词"的意图, 在"我爱猫"的 Key 里找到了"爱"这个词, 匹配度极高 !
- **输出:** 得到融合了中文信息的特征矩阵 Z_{cross} .

Z_{cross} 经过前馈网络和输出层, 变成对下一个词的概率预测.

- 这里的前馈网络: 巨大的线性层, 把维度为 d 的 Z 投影到维度为 V (词表大小, 比如 30,000) 的空间.
- 这里的输出层: 一般是 Softmax 层, 把投影后的数值转换成概率分布.

最后是损失函数计算.

- 通过 softmax 后得到的就是一个概率分布向量. 例如, 结果是模型觉得下一个词是 "love" 的概率是 90%.
 - 计算 L : 拿这个概率和真实标签 y ("love") 算交叉熵. 通过反向传播, 更新所有 W^Q, W^K, W^V 参数.
-
-

如果要从**生成器**改成**分类器**(最终输出类别标签),

需要:

- 把 decoder 部分去掉, 只保留 encoder 部分.
- 输出层

Video Transformer

在视频 Transformer (ViT) 中, 我们通常不会把整张图片当成一个单元, 而是把它们切碎. 这里的逻辑是这样的:

- 切块 (Patching): 我们把每一帧图片切成很多个小方块 (Patches).
- 排列: 假设一帧切成 16 块, 100 帧就有 $100 \times 16 = 1600$ 个小块.
- 向量化 (Embedding): 每个小块会被转换成一个长度为 d 的向量.

此时, 你的 3D 视频矩阵, 在 Transformer 眼里就变成了一个形状为 $1600 \times d$ 的巨大二维矩阵.

