DL深度学习

手撕API

Pytorch

TensorFlow

Transformer

部分指标

准确率（Accuracy）

所有分类正确的样本数量占总样本数量的比例

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

精确率（Precision）

所有被预测为正类的样本中，实际为正类的比例

$$Precision = \frac{TP}{TP + FP}$$

召回率（Recall）

所有实际为正类的样本中，被正确预测为正类的比例

$$Recall = \frac{TP}{TP + FN}$$

数据预处理

1. **数据清洗：**去除噪声数据、处理缺失值、处理异常值
2. **特征缩放**：
   - **归一化（Normalization）**：将数据缩放到 [0, 1] 区间
   - **标准化（Standardization）**：将数据转换为均值为0，标准差为1的分布
   - **鲁棒缩放（Robust Scaling）**
3. **数据增强**：
   - **图像**
     - 基本：旋转、缩放、裁剪

    - 特别：颜色变换、颜色扰动、遮掩/坏块、合成新图
- **文本**
    - 同义词替换、随机插入、删除、交换等方式增加样本

4. **编码转换**：
    - 独热编码、标签编码
5. **特征提取**：
    - 从原始数据中提取更有意义的特征，例如从文本中提取TF-IDF特征
6. **时间序列处理**：
    - 可能需要进行时间对齐、滑动窗口特征提取等
7. **文本预处理**：
    - 分词、去除停用词、词干提取、词形还原等
8. **图像预处理**：
    - 调整图像大小以适应模型输入
    - 应用滤波器以减少噪声
    - 转换为灰度图像
9. **音频预处理**：
    - 采样率转换、静音切除、频谱图转换等
10. **序列填充或截断**：
    - 对于序列数据，可能需要将序列填充到相同长度或截断到固定长度
11. **批处理、随机打乱**

有监督、自监督、半监督、无监督学习

	数据需求	典型任务	代表模型
有监督学习	标注数据（X, Y）	分类、回归	LR、CNN
自监督学习	无标签数据（X）	预训练表示学习	BERT
半监督学习	少量（X, Y）+ 大量X	分类（增强泛化）	GCN
无监督学习	无标签数据（X）	聚类、降维	K-Means

反向传播

从输出层开始，逐层计算各层的误差，利用微积分中的链式法则，将损失函数对权重的梯度分解为各层激活值的局部梯度乘积。除了问原理，偶尔还以以下方式考查：

- 手搓反向传播，稍微常见一点
- 某特定层：卷积层、池化层或激活函数的反向传播过程，这种需要结合对特定层的理解，建议结合自己的简历自行问AI

过拟合、欠拟合

问题	训练集表现	测试集表现	模型复杂度	解决方法
过拟合	很好	很差	高	简化模型、增加数据、正则化
欠拟合	很差	很差	低	增加复杂度、改进特征继续训练

梯度消失，怎么解决

梯度消失是指在深度神经网络训练中，随着反向传播的进行，网络较深层的权重更新所使用的梯度变得非常小，甚至趋近于零，导致深层网络无法有效地学习和更新参数的问题。梯度消失通常发生在深度神经网络中，特别是在使用一些饱和度较高的激活函数（如Sigmoid、Tanh）或者网络结构较深时

解决梯度消失问题的方法：

- **使用适当的激活函数**：选择一些饱和度较低的激活函数，如ReLU、Leaky ReLU等，可以减少梯度消失的风险，这些激活函数在一定范围内不会饱和
- **标准化或归一化**
- **残差连接**：像ResNet这样的网络结构引入了残差连接，使得网络可以直接学习残差（即跳过连接的部分），从而减少了梯度消失的影响，有助于训练更深的网络
- **梯度裁剪**：梯度裁剪不仅可以用来解决梯度爆炸问题，也可以在一定程度上减轻梯度消失问题
- **合适的权重初始化**

梯度爆炸，怎么解决

梯度爆炸是指在深度神经网络训练过程中，梯度值变得非常大，甚至超出了计算机的数值表示范围，导致权重参数更新异常剧烈，训练过程失效的现象。梯度爆炸通常发生在网络层数较深、激活函数梯度较大或者学习率设置过高的情况下。

解决梯度爆炸问题的方法：

- **梯度裁剪**：通过设置一个阈值，当梯度的范数超过该阈值时，对梯度进行缩放
- **使用合适的激活函数**：使用一些饱和度较低的激活函数，如ReLU、Leaky ReLU等
- **初始化权重**：合适的权重初始化方法也可以帮助减少梯度爆炸的发生
- **减小学习率**：适当减小学习率可以减缓梯度的增长速度，降低梯度爆炸的风险
- **归一化**：可以一定程度上减少梯度爆炸的问题
- **使用梯度消失较小的网络结构**：一些网络结构设计上避免梯度爆炸的风险，如ResNet中的残差连接可以帮助梯度更好地传播。

Dropout

一种常用的正则化技术，主要用于防止神经网络过拟合。核心思想是在训练过程中随机“丢弃”一部分神经元，从而减少神经元之间的复杂共适应性，增强模型的泛化能力

- 在每次训练迭代中，随机将神经网络中的一部分神经元“丢弃”（即将其输出设为 0），同时将剩余神经元的输出按比例放大
- 在测试阶段，不使用Dropout，而是将所有神经元的输出乘以Dropout的概率（以保持输出的期望值不变）

残差连接的意义

防止梯度消失，提升模型表达能力，稳定深层网络训练

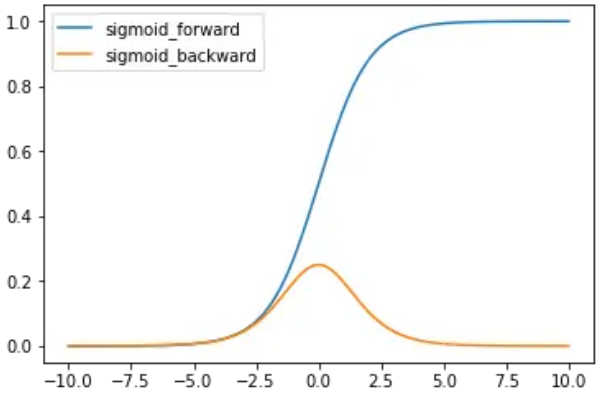
激活函数

激活函数的主要目的是将非线性特性引入模型，决定了某个神经元是否被激活，这个神经元接受到的信息是否是有用的，是否该留下或者是该抛弃

Sigmoid函数

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}$$

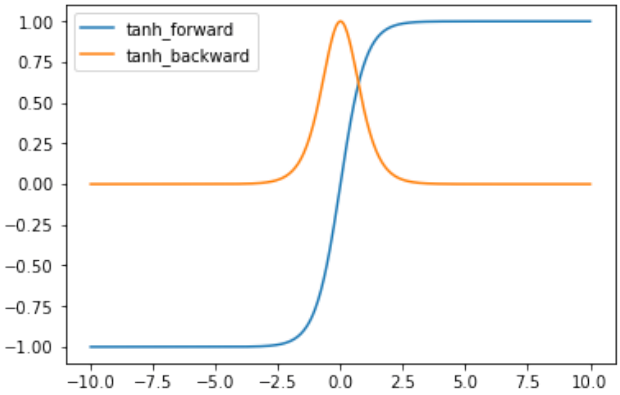
- 范围在 (0, 1) 之间
- 容易出现梯度消失的问题
- 在输出层用于二分类问题



Tanh函数

$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

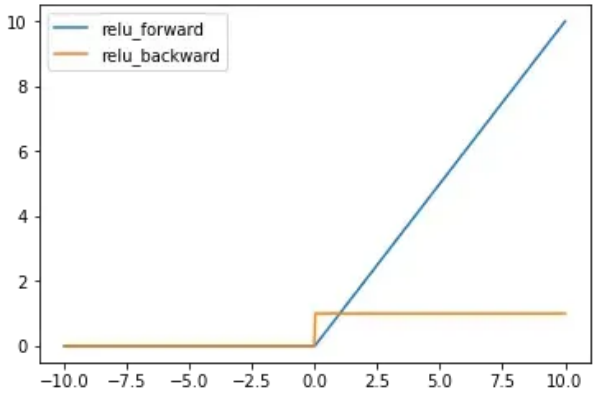
- 范围在 (-1, 1) 之间
- 比Sigmoid函数的输出范围更广



ReLU 函数（Rectified Linear Unit）

$$ReLU(x) = max(0, x)$$

- 简单且计算高效
- 解决了梯度消失问题
- 在深度学习中应用广泛

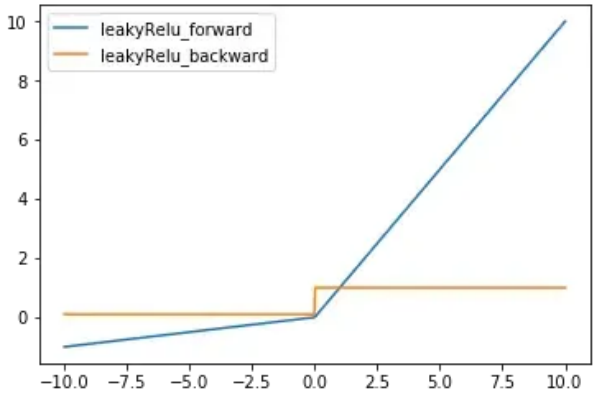


原始的transformer网络使用的是ReLU

Leaky ReLU 函数

$$LeakyReLU(x) = max(\alpha x, x)$$

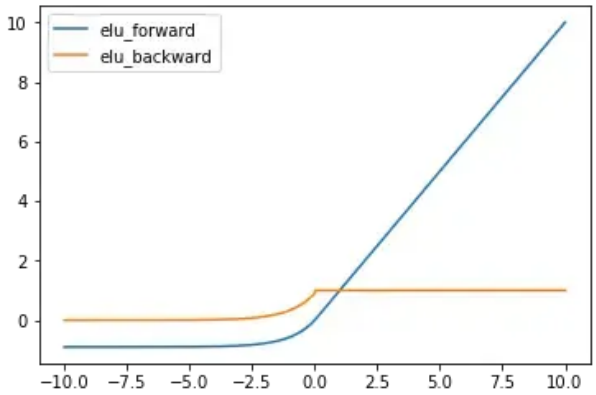
- 允许小于零时有一个小的斜率 α，通常取很小的值如 0.01
- 解决了ReLU函数中负值区域梯度为零的问题



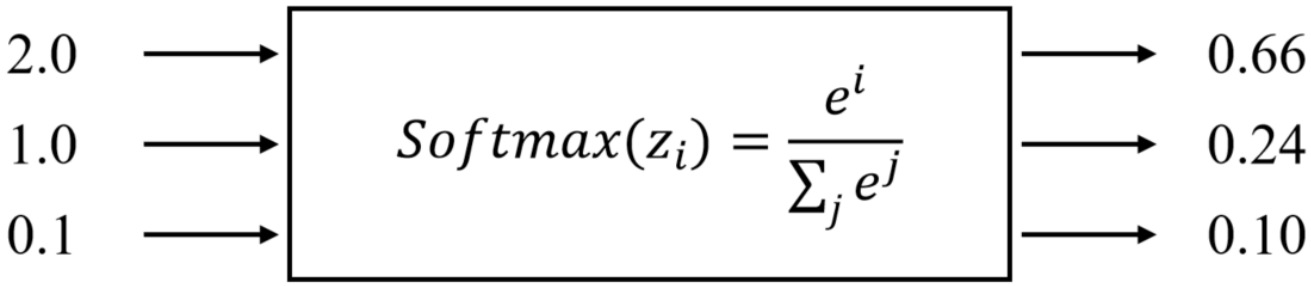
ELU（Exponential Linear Unit）

$$ELU(x) = max(0, x) + min(0, \alpha(e^x - 1))$$

- 具有平滑的负值区域
- 可以帮助加速收敛



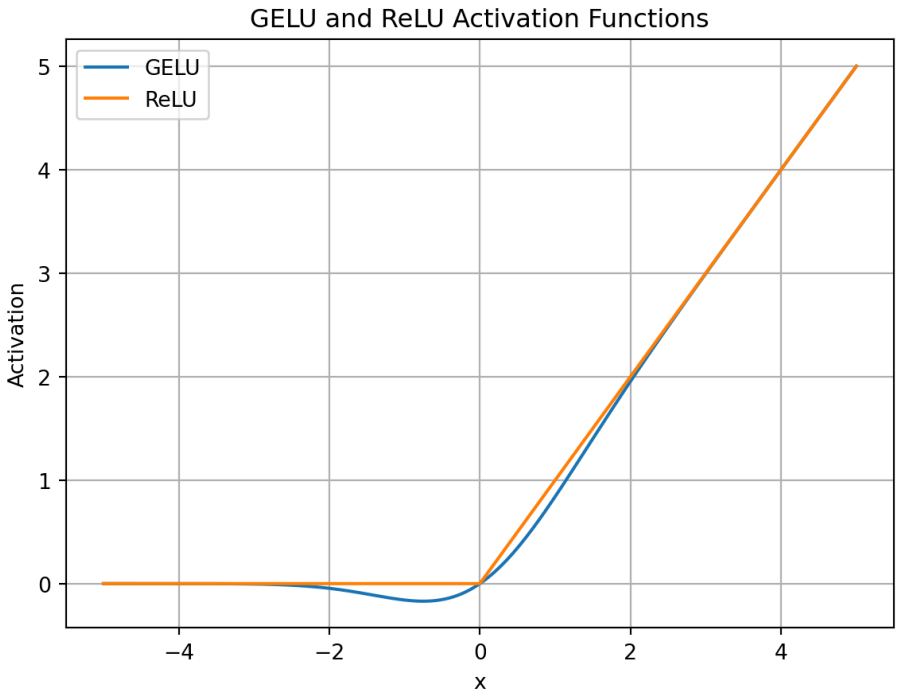
Softmax



- 用于多分类问题，将输出转换为概率分布
- 在零点不可微
- 如果某个神经元的输入长期为负值，其梯度会接近于零，权重无法有效更新

GELU

$$GELU(X) = 0.5 \times x(1 + tanh[\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)])$$



BERT、RoBERTa、GPT2、GPT3使用了激活函数GELU

GLU

引入了两个不同的线性层，其中一个首先经过sigmoid函数，其结果将和另一个线性层的输出进行逐元素相乘作为最终的输出（很多 transformer 模型的FFN都去掉了 bias ）

$$GLU(x, W, V, b, c) = \sigma(xW + b) \bigodot (xV + c)$$

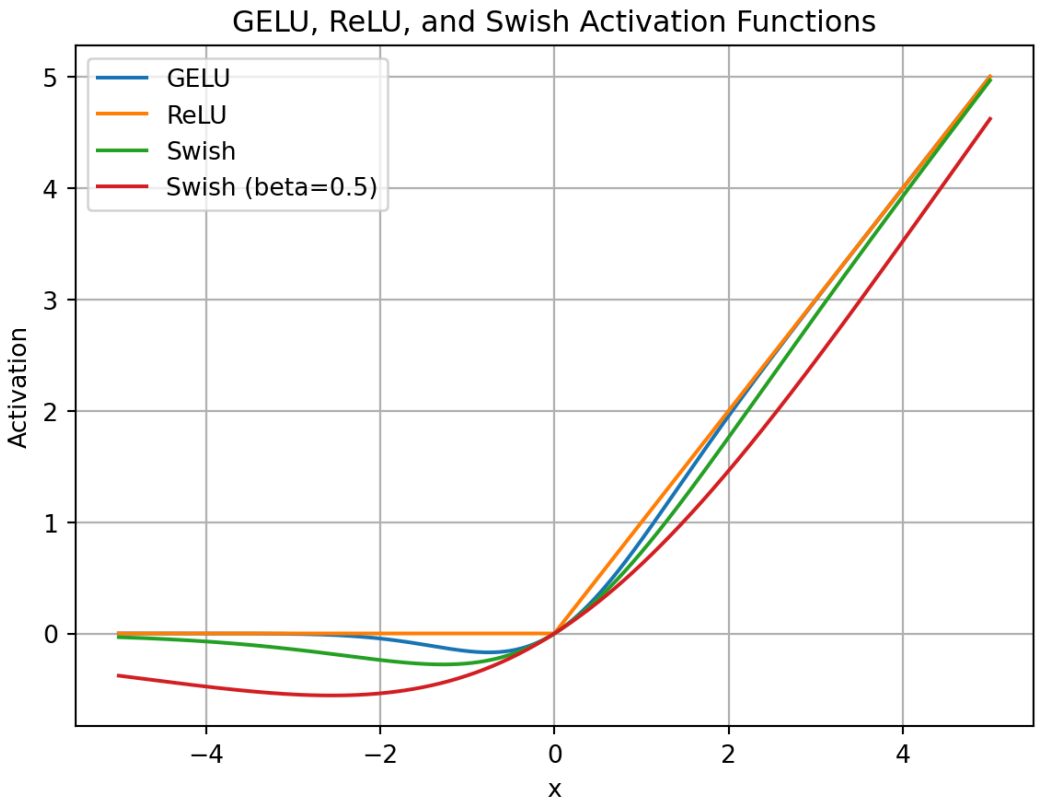
这里W,V以及b,c分别是这两个线性层的参数；$\sigma(xW + b)$作为门控，控制另一个线性层的输出

Swish

由Google团队在2017年提出，被证明在更深的模型上表现出比ReLU更好的性能

$$Swish_\beta(x) = x\sigma(\beta x)$$

其中σ(x)是Sigmoid函数，β为1时，Swish就变成了SiLU (Sigmoid Linear Unit)，大多数框架的默认实现（如PyTorch、TensorFlow 的 nn.SiLU() ）使用的是β=1的固定版本



SwiGLU

SwiGLU就是把门控函数替换成了swish，并且去除掉了bias部分（很多transformer模型的FFN都去掉了bias）

$$SwiGLU(x, W, V, b, c) = Swish_1(xW + b) \bigodot (xV + c)$$

现在主流的LLM比如LLaMA2和Qwen2.5使用的激活函数是SwiGLU

loss函数

损失函数是用来估量模型的输出y^与真实值y之间的差距，给模型的优化指引方向

MAE

主要用于回归任务，也叫L1 loss，平均绝对误差损失 Mean Absolute Error Loss

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

MAE对异常值不敏感

MSE

主要用于回归任务，也叫L2 loss，均方差损失Mean Squared Error Loss

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

对异常值敏感，因为当异常值与正常值差距较大时，误差会大于1，取平方值以后会进一步增大数值

Smooth L1 Loss

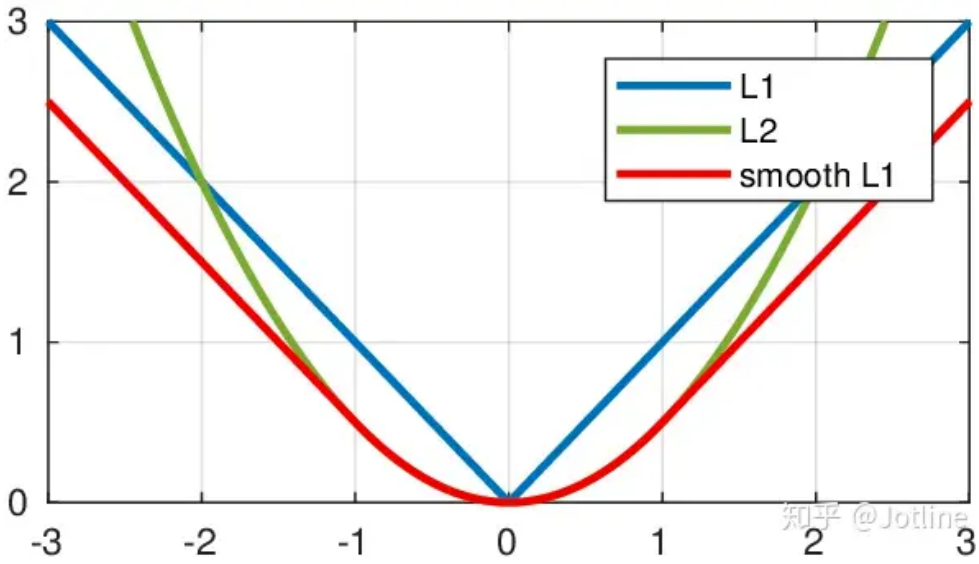
也叫Huber loss。**SoothL1Loss其实是L2Loss和L1Loss的结合**，它同时拥有L2 Loss和L1 Loss的部分优点。

当预测值和ground truth差别较小的时候（绝对值差小于1），其实使用的是L2 Loss；而当差别大的时候，是L1 Loss的平移。

1. 当预测值和ground truth差别较小的时候（绝对值差小于1），梯度不至于太大。（损失函数相较L1 Loss比较圆滑）

2. 当差别大的时候，梯度值足够小（较稳定，不容易梯度爆炸）

$$loss(x, y) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}0.5 * (y_i - \hat{y}_i)^2, & if\ |y_i - \hat{y}_i| < 1\\ |y_i - \hat{y}_i| - 0.5, & otherwise\end{cases}$$



RMSE

对MSE开方

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

CrossEntropy

二分类（BCE Loss，Binary Cross-Entropy Loss）

$$BCELoss(y,\hat{y}) = -\left[y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})\right]$$

- y：真实标签（0或1）
- $\hat{y}$：模型预测的概率

BCEloss平等地对待正负样本，当正负样本极度不平衡时（例如正样本很少），模型可能会倾向于预测负样本，导致正样本的分类效果较差

为什么BCEloss可能会导致偏差

$$CrossEntropy = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i)) + (1-y_i)\log(1-\hat{y}_i))\right]$$

如上图，因为真用起来的时候要加起来，如果负样本多了loss就主要由负样本主导了

Focal Loss

Focal Loss是对BCELoss的改进，引入了调制因子，用于降低易分类样本的损失权重，增加难分类样本的损失权重，旨在解决类别不平衡问题

$$FocalLoss(y,\hat{y}) = -\left[y \cdot (1-\hat{y})^{\gamma} \cdot \log(\hat{y}) + (1-y) \cdot \hat{y}^{\gamma} \cdot \log(1-\hat{y})\right]$$

其中，γ是一个可调参数（通常设为 2）

BCEWithLogitsLoss

结合了sigmoid激活函数和BCE Loss，以提高数值稳定性和计算效率。该损失函数接受模型的原始输出，即未经过激活函数的logits（通常是线性层的输出），并在内部应用sigmoid来计算预测概率

多分类

真实标签*y*是一个**one-hot向量**，例如*y*=[0,0,1,0]

模型预测的概率分布*y*^是一个**概率向量**，例如*y*^=[0.1,0.2,0.6,0.1]

$$CrossEntropy(y,\hat{y}) = -\sum_{i=1}^{C} y_i \cdot \log(\hat{y}_i)$$

- *C*是类别总数
- y_i是真实标签的第*i*个元素（0或1）
- $\hat{y}_i$是模型预测的第*i*个类别的概率

由于*y*是one-hot向量，只有一个元素为1，其余为0，因此公式可以简化为

$$CrossEntropy(y,\hat{y}) = -\log(\hat{y}_k)$$

```
# 示例数据
logits = np.array([[2.0, 1.0, 0.1, 0.5, 0.3],  # 样本 1
```

```
                    [1.0, 3.0, 0.2, 0.4, 0.5],  # 样本 2
                    [0.5, 0.2, 2.0, 1.0, 0.3]]) # 样本 3
labels = np.array([0, 1, 2])  # 真实标签

# 交叉熵损失函数
def cross_entropy_loss(logits, labels):
    # 计算 Softmax 概率
    probs = softmax(logits)

    # 获取每个样本的真实类别概率
    n_samples = labels.shape[0]
    true_class_probs = probs[np.arange(n_samples), labels]

    # 计算交叉熵损失
    loss = -np.sum(np.log(true_class_probs)) / n_samples
    return loss
```

KL散度

Kullback-Leibler散度，用于衡量两个**概率分布**之间的差异，强调匹配真实分布的概率值

- 如果*Q*完全等于*P*，则KL散度为0，表示没有信息损失
- 如果*Q*与*P*差异较大，则KL散度较大，表示信息损失较多

定义

KL散度衡量的是**用一个分布*Q*来近似真实分布*P***时的信息损失

$$D_{KL}(P \parallel Q) = \sum_{i} P(i) \log \left(\frac{P(i)}{Q(i)} \right)$$

对于连续分布

$$D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx$$

- *P*是真实分布（目标分布）
- *Q*是近似分布（模型分布）
- *P*(*i*)和*Q*(*i*)分别是分布*P*和*Q*在点*i*处的概率

KL散度与交叉熵的关系

KL 散度可以分解为交叉熵和熵的形式

$$D_{KL}(P \parallel Q) = H(P, Q) - H(P)$$

其中：

- $H(P,Q)$ 是交叉熵，定义为：

$$H(P,Q) = -\sum_{i} P(i) \log Q(i)$$

- $H(P)$ 是分布 P 的熵，定义为

$$H(P) = -\sum_{i} P(i) \log P(i)$$

因此，KL 散度可以理解为**交叉熵减去熵**

one-hot编码

One-hot 编码是一种常用的数据编码方式，通常用于将分类变量转换为机器学习算法可以处理的形式。在One-hot编码中，每个类别被表示为一个向量，其中只有一个元素为1，其余元素为0。这个1表示该样本属于这个类别，而0表示不属于其他类别。

举例：

假设有一个包含三个类别的分类变量：狗、猫、鼠。使用One-hot编码后，这三个类别将被编码为：

- 狗：[1, 0, 0]
- 猫：[0, 1, 0]
- 鼠：[0, 0, 1]

余弦相似度Cosine Similarity

$$\text{余弦相似度} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

计算步骤

1. 计算点积

$$A \cdot B = \sum_{i=1}^{n} a_i \times b_i$$

2. 计算向量的模

$$\|A\| = \sqrt{\sum_{i=1}^{n} a_i^2}, \quad \|B\| = \sqrt{\sum_{i=1}^{n} b_i^2}$$

3. 计算余弦相似度

$$\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

特点

- 范围：[-1, 1]
  - 1表示两个向量完全相同
  - 0表示两个正交（无相关性）
  - -1表示两个向量完全相反

示例

A = [1, 2, 3]; B = [4, 5, 6]

1. 计算点积

A·B = 1×4 + 2×5 + 3×6 = 4 + 10 + 18 = 32

2. 计算模

$$\|A\| = \sqrt{1^2 + 2^2 + 3^2} = \sqrt{1 + 4 + 9} = \sqrt{14}$$

$$\|B\| = \sqrt{4^2 + 5^2 + 6^2} = \sqrt{16 + 25 + 36} = \sqrt{77}$$

3. 计算余弦相似度

$$\cos(\theta) = \frac{32}{\sqrt{14} \times \sqrt{77}} \approx 0.9746$$

结果表明，向量A和B的相似度较高

优化器Optimizer

SGD

stochastic gradient descent，随机梯度下降，非常经典，每次随机选择一个样本计算梯度

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J(\theta)$$

- 计算快，适合大规模数据
- 波动大，可能难以收敛

Mini-batch SGD

每次使用一个小批量（batch）数据计算梯度

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta J(\theta; x_i, y_i)$$

Momentum

$$v_{t+1} = \gamma v_t + \eta \nabla_\theta J(\theta_t)$$
$$\theta_{t+1} = \theta_t - v_{t+1}$$

- v_t：第*t*步的动量项（初始时为0）
- γ：动量系数（通常设为 0.9），控制历史梯度的保留程度
- η：学习率
- $\nabla_\theta J(\theta_t)$：当前步的梯度

vt 的作用

加速收敛

- 在梯度方向一致的维度上，动量项会累积梯度，增大更新步长
- 例如：损失函数在某一方向持续下降时，v_t会逐渐增大，加快收敛

减少震荡

- 在梯度方向频繁变化的维度上（如峡谷形损失函数），正负梯度会部分抵消，抑制震荡

Adagrad

核心思想是**为每个参数自动调整学习率**，使得频繁更新的参数学习率较小，稀疏更新的参数学习率较大

$$G_t = G_{t-1} + (\nabla_\theta J(\theta_t))^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

- G_t：**梯度平方的累积**（对角矩阵，存储每个参数的历史梯度平方和）
- η：初始学习率
- ϵ：极小值防止除零错误
- $\nabla_\theta J(\theta_t)$：当前梯度
- 适合稀疏特征（如 NLP），但学习率会单调下降至消失

RMSProp

改进AdaGrad，引入指数加权平均避免学习率衰减过快

$$G_t = \beta G_{t-1} + (1-\beta)(\nabla_\theta J(\theta_t))^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla_\theta J(\theta_t)$$

- β：通常0.9，控制历史梯度的影响
- 适合非平稳目标（如RNN）

Adam

超常见的一个优化器，结合了动量（Momentum）和自适应学习率的优点。通过计算梯度的一阶矩（均值）和二阶矩（未中心化的方差）来调整每个参数的学习率

- **计算梯度的一阶矩和二阶矩**

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta J(\theta_t)$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta J(\theta_t))^2$$

其中，$\nabla_\theta J(\theta_t)$是当前梯度，*mt*和*vt*分别是一阶矩和二阶矩，*β*1和*β*2是衰减率（通常设为 0.9 和 0.999）

- **修正偏差**

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

- **更新参数**

$$\theta_t = \theta_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

其中，*η*是学习率，*ϵ*用于防止除零错误

AdamW

- 在Adam的基础上，修正了权重衰减的实现方式
- 将权重衰减与梯度更新分离，避免权重衰减与自适应学习率之间的耦合
- **前面两步相同，更新参数的方式不一样**

$$\theta_t = \theta_{t-1} - \eta \cdot \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_{t-1} \right)$$

归一化Normalization

是深度学习和机器学习中常用的技术，目的是将数据或特征的分布调整到一定的范围内，以加速模型训练、提高模型性能或增强模型的稳定性

Batch Normalization (BN)

计算当前批次数据的均值和方差，然后用这些统计量对数据进行归一化

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

- x_i：输入数据
- μ_B：当前数据的均值
- σ_B^2：当前批次的方差

- ϵ：防止除零错误

归一化后，BN还会引入可学习的缩放参数γ和偏移参数b，以恢复模型的表达能力：

$$y_i = \gamma \hat{x_i} + b$$

优点

- 加速模型训练，缓解梯度消失问题
- 有一定的正则化效果，可以减少对Dropout的依赖

缺点

- 对Batch Size敏感，小批量数据下效果较差
- 不适用于RNN等动态网络结构

适用场景

- 主要用于CNN和全连接网络（如ResNet、VGG等）

Layer Normalization (LN)

对于每个样本，LN计算该样本所有特征的均值和方差，然后用这些统计量对数据进行归一化：

$$\hat{x}_i = \frac{x_i - \mu_L}{\sqrt{\sigma_L^2 + \epsilon}}$$

- μ_L：当前样本的均值
- σ_L^2：当前样本的方差

优点

- 不依赖于Batch Size，适合小批量数据或单样本数据
- 适用于RNN、Transformer等动态网络结构

缺点

- 在CNN中效果不如BN

适用场景

- 主要用于RNN、Transformer等序列模型（如 BERT、GPT）

Instance Normalization (IN)

对每个**样本的每个通道**进行归一化

$$\hat{x}_i = \frac{x_i - \mu_I}{\sqrt{\sigma_I^2 + \epsilon}}$$

优点

- 在图像生成任务中效果很好，能够保留图像的风格信息
- 不依赖于Batch Size

缺点

- 在分类任务中效果不如 BN

适用场景

- 主要用于图像生成任务（如 StyleGAN）

Group Normalization (GN)

将每个样本的通道分成若干组，对每组计算均值和方差，然后用这些统计量对数据进行归一化

$$\hat{x}_i = \frac{x_i - \mu_G}{\sqrt{\sigma_G^2 + \epsilon}}$$

优点

- 不依赖于Batch Size，适合小批量数据
- 在CNN中效果优于LN和IN

缺点

- 需要手动设置分组数。

适用场景

- 主要用于小批量数据的CNN（如目标检测任务中的 Faster R-CNN）

RMS Normalization （RMSNorm）

通过计算输入数据的均方根值，并将输入数据除以此值，从而实现归一化

给定输入向量$x \in \mathbb{R}^d$，RMSNorm的计算公式如下：

$$RMS(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}$$

$$\hat{x}_i = \frac{x_i}{RMS(x)}$$

- x_i：输入向量的第i个元素
- $RMS(x)$：输入向量的均方根值
- $\hat{x}_i$：归一化后的输出向量的第i个元素

优点

- 计算简单：只需计算输入数据的均方根值，计算复杂度较低
- 提高模型的训练稳定性
- 适应性：不依赖于批处理数据，适用于小批量或在线学习场景

与其他方法的比较

- Batch Normalization (BN)：BN依赖于批处理数据的均值和方差，因此在batch较小时效果不佳。RMSNorm不依赖于批数据，适用于各种batch大小
- Layer Normalization (LN)：LN对每一层的输入进行归一化，但计算的是均值和方差。RMSNorm只计算均方根值，计算更简单
- Instance Normalization (IN)：IN主要用于图像处理任务，对每个样本的每个通道进行归一化。RMSNorm更通用

RMSNorm可以应用于各种神经网络中，在现在主流的LLM中，其被广泛用于替代LN

模型参数初始化为0有什么问题

对称性问题

如果模型的所有参数（如神经网络的权重）都初始化为相同的值（例如 0），那么在反向传播过程中，所有参数会以相同的方式更新。这会导致神经网络的每一层中的神经元学习到相同的特征，从而失去多样性。

导致模型的表达能力会受到限制，无法捕捉数据中

梯度消失问题

如果参数初始化为0，某些激活函数（如Sigmoid或Tanh）在输入为0时的梯度非常小。在反向传播过程中，梯度会逐层减小，最终导致梯度消失，参数无法有效更新

模型训练速度变慢，甚至完全停止学习

模型无法启动学习

如果参数初始化为0，某些模型（如线性回归、逻辑回归）的输出可能会完全一致，导致损失函数的梯度为0，模型无法启动学习过程

对某些模型的影响

- **神经网络**：神经网络对参数初始化非常敏感，初始化为0会导致上述对称性和梯度消失问题
- **树模型（如决策树、随机森林）**：树模型通常不依赖参数初始化，因此初始化为0不会对其产生影响
- **线性模型（如线性回归、逻辑回归）**：初始化为0可能会导致模型无法启动学习，但可以通过调整学习率或使用随机初始化解决

解决方法

参数初始化：使用随机初始化方法代替全0初始化

模型选择：如果数据稀疏性较高，可以选择适合稀疏数据的模型（如基于树模型的方法或稀疏神经网络）

训练无法收敛可能是什么导致的

- 学习率过大或过小、优化器选择不当、损失函数设计的不对、Batch Size过大或过小

- 数据有误，如未归一化或标准化或者标签有错误
- 模型参数未正确初始化、梯度爆炸或消失
- 模型架构有问题：太简单无法捕获复杂表示、太复杂了过拟合了早期数据、激活函数有问题等

如何判断训练已经收敛

- 训练损失是否稳定（变化幅度 < 阈值）
- 验证指标是否不再提升（如准确率、F1）
- 梯度是否接近零（`max(|grad|) < 1e-5`）
- 早停条件是否触发（如 `patience=10`）
- 训练/验证曲线是否正常（无过拟合或欠拟合）

如何判断对于某个任务，模型的参数是否足够

1. **看训练表现**
   - 训练误差高 → 参数不足（欠拟合）
   - 训练误差接近低但验证误差高 → 参数过多（过拟合）
2. **看验证表现**
   - 参数合适：验证误差随训练同步下降后稳定
   - 参数不足：两条线误差都高
   - 参数过多：训练/验证误差差距大
3. **看数据比例**
   - 参数数量 ≤ 训练样本数/10（小数据集）
   - 大数据集可适当突破
4. **使用辅助工具**
   - 使用`tensorboard`查看各层激活值，若多数神经元始终不激活，可能参数有浪费
   - 使用`神经架构搜索（NAS）`（听过但不会）
5. **尝试和对比**
   - 从简单模型开始尝试，逐步增加层宽/深，观察变化
   - 对比baseline，同类型任务中有哪些模型变现好，参考他们的网络设计

模型量化和剪枝

模型量化

模型量化主要目的是将模型中的浮点数参数转换为整数形式，以减少模型的存储空间和计算复杂度。量化操作涉及将实数映射到整数

模型剪枝

模型剪枝的目标是通过移除模型中不重要的权重或神经元来减少模型的大小和计算量

> 量化和剪枝可以组合使用，以实现更高效的模型压缩

模型蒸馏

一种模型压缩和知识迁移技术，旨在将一个复杂的大型模型（教师模型）所学习到的知识迁移到一个相对简单的小型模型（学生模型）中，使小型模型在保持一定性能的同时，具有更小的计算量和更快的推理速度

基本原理

- **知识提取**

教师模型通常是在大规模数据上经过长时间训练得到的，具有丰富的知识和强大的性能。蒸馏模型通过一种特殊的训练方式，让学生模型从教师模型中提取有用的知识

- **软标签与硬标签**

在传统的模型训练中，使用真实标签（硬标签）进行监督学习。而蒸馏模型除了使用硬标签外，还利用教师模型对数据的预测结果作为软标签。软标签包含了更多的信息，如不同类别之间的概率分布，学生模型通过学习软标签来获取教师模型的知识

工作流程

1. **训练教师模型**

首先在大规模的标注数据集上训练一个复杂的教师模型，使其达到较高的性能水平。例如，在图像识别任务中，使用数百万张图像数据训练一个深度卷积神经网络作为教师模型，使其能够准确地对各种图像进行分类

2. **生成软标签**

将训练数据输入到教师模型中，得到教师模型对每个数据样本的预测结果，即软标签。这些软标签反映了教师模型对数据的理解和判断，包含了比硬标签更丰富的信息

3. **训练学生模型**

使用软标签和硬标签共同训练学生模型。在训练过程中，通过定义一个合适的损失函数，使学生模型的输出既接近硬标签，又接近教师模型的软标签。通过不断调整学生模型的参数，使其逐渐学习到教师模型的知识

关键技术

- **损失函数设计**

蒸馏模型的损失函数通常由两部分组成，一部分是学生模型输出与硬标签之间的交叉熵损失，另一部分是学生模型输出与软标签之间的交叉熵损失。通过调整这两部分损失的权重，可以平衡学生模型对硬标签和软标签的学习

- **温度参数调整**

在计算软标签和学生模型输出的交叉熵损失时，通常会引入一个温度参数。温度参数用于调整软标签的平滑程度，较高的温度会使软标签更加平滑，包含更多的信息。通过调整温度参数，可以控制学生模型对软标签的学习

程度

应用场景

- **模型部署**
- **模型加速**

zero shot、one shot、few shot

方法	训练数据量	核心思想	
Zero-shot	未见过的类别	利用语义关系或先验知识泛化到新类别	CLIP、GPT
One-shot	每个类别1个样本	从单个样本中学习关键特征	人脸识别、签名验证
Few-shot	每个类别少量样本	从少量样本中提取有用信息	医学图像分类、低资源语言翻译