##### Transformer

###### 介绍一下bert结构

BERT使用了Transformer模型中的编码器（Encoder）部分，而没有使用解码器（Decoder）。Transformer编码器由多个层（通常是12层）组成，每层包括自注意力机制和前馈神经网络。

双向编码：与之前的语言模型不同，BERT采用双向训练，即在预训练阶段，每个词的表示同时考虑了它之前和之后的上下文信息

###### Transformer结构与原理

输入处理：

输入序列通过词嵌入（Embedding）层转化为高维向量表示。加入位置编码（Positional Encoding），使得模型能感知序列中各个词的位置。

编码器（Encoder）：

由 N 层相同的编码器层组成，每层包括两个子层：多头自注意力机制（Multi-Head Self-Attention）和前馈神经网络（Feed-Forward Neural Network）。每个子层后都有残差连接（Residual Connection）和层归一化（LayerNormalization）。

多头自注意力机制：将输入映射到查询（Q）、键（K）、值（V）矩阵，通过自注意力计算输出表示。

前馈神经网络：对每个位置的表示独立地进行两次线性变换和激活函数操作。

解码器（Decoder）：

也由 N 层相同的解码器层组成，每层有三个子层：多头自注意力机制（对解码器自身输入）、多头注意力机制（对编码器输出）和前馈神经网络。每个子层后也有残差连接和层归一化。解码器的多头自注意力机制在训练时使用掩码（Masking），以确保当前位置只能关注之前的位置。

输出处理：

解码器的最终输出经过线性变换和 Softmax 层，得到每个位置上可能的词汇分布。

###### transformer和CNN的区别是什么

CNN：依赖于局部连接和权重共享，这意味着每个卷积核只关注输入数据的一个局部区域，适合处理具有强烈局部相关性的数据，如图像。

Transformer：通过自注意力机制处理序列数据，能够捕捉长距离依赖关系，适合处理序列数据，如文本或时间序列。

CNN：卷积核的参数在整个输入上共享，这减少了模型的参数数量，但限制了模型捕获全局依赖的能力。

Transformer：虽然也有参数共享（如自注意力和前馈网络中的权重），但自注意力机制允许模型在序列的每个位置捕获全局信息。

模型结构不同

CNN：在计算机视觉任务中非常流行，如图像分类、目标检测和图像分割。

Transformer：在自然语言处理（NLP）任务中表现出色，如机器翻译、文本摘要、问答系统等。

CNN：卷积层可以被解释为提取图像中的局部特征，如边缘和纹理。

Transformer：自注意力机制提供了一种直观的方式来理解模型是如何在不同位置之间分配注意力权重的。

###### 注意力机制

通过引入注意力机制，神经网络能够自动地学习并选择性地关注输入中的重要信息，提高模型的性能和泛化能力。

###### 自注意力机制

每个元素都可以与序列中的其他元素建立关联，而不仅仅是依赖于相邻位置的元素。它通过计算元素之间的相对重要性来自适应地捕捉元素之间的长程依赖关系。

具体而言，对于序列中的每个元素，自注意力机制计算其与其他元素之间的相似度，查询向量Q和键向量K点积，再经过缩放操作，得到注意⼒分数。并将这些相似度归一化为注意力权重。然后，通过将每个元素与对应的注意力权重进行加权求和，可以得到自注意力机制的输出。

###### 多头注意力机制的多头指的是

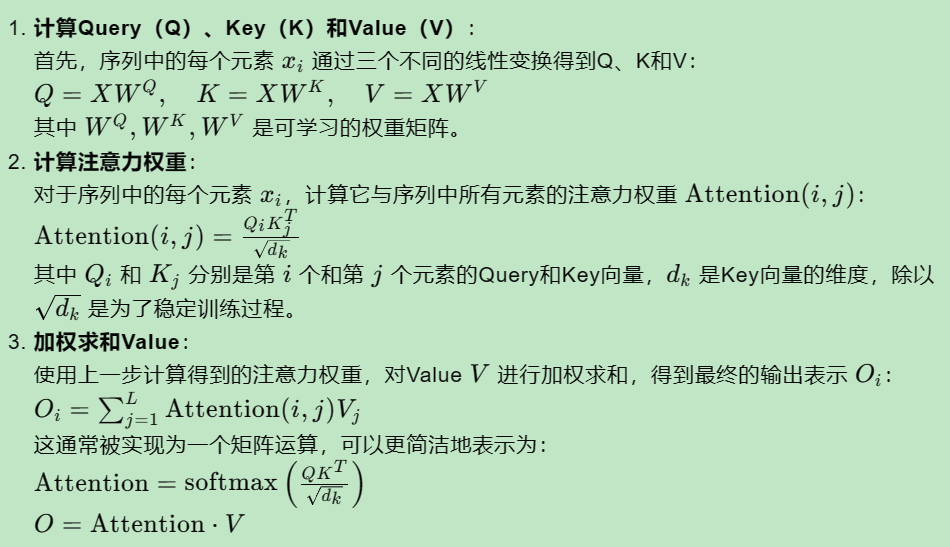
在自注意力机制中，每个单词或者字都仅仅只有一个 查询向量Q、键向量K和数值向量V与其对应，使用多个独立的注意力头，分配多个 QKV，分别计算注意力权重，并将它们的结果进行拼接或加权求和，从而获得更丰富的表示

###### Attention和Transformer的原理

第一步： query 和 key 进行相似度计算，得到权值

第二步：将权值进行归一化，得到直接可用的权重

第三步：将权重和 value 进行加权求和



**时间序列预测的模型架构和指标**

###### 多头注意力机制里的QKV

查询（Q）：表示当前词语的查询向量，用于在序列中查找相关信息。

键（K）：表示序列中所有词语的键向量，用于与查询向量计算相似度。

值（V）：表示序列中所有词语的值向量，是计算注意力加权平均后的输出

为什么基于 Transformer 的架构需要多头注意力机制**？**

并行处理：Transformer架构的一个核心特点是其能够并行处理序列数据，而多头注意力机制正是这种并行性的关键组成部分。

信息多样性：通过多头注意力，模型可以同时从不同的表示子空间中学习信息，这有助于捕获数据中的多种特征和模式。

增强的学习能力泛化能力和鲁棒性：多头注意力机制可以增强模型的学习能力，因为它允许模型在多个层面上同时学习数据的复杂特征。

改善梯度流动：在深层网络中，梯度可能会消失或爆炸，多头注意力机制有助于改善深层Transformer网络中的梯度流动问题。

##### 机器学习

结构风险最小化是指在模型选择中不仅考虑经验风险（）（在训练集上的误差）最小化，还考虑了模型复杂度。

经验风险最小化（Empirical Risk Minimization，ERM）是指在模型选择中在训练集上的误差最小化。这种方法容易导致过拟合（Overfitting），因为模型可能会过于复杂，

###### SVM 之所以能够减少VC维度

它在构建模型时采用了结构风险最小化的原则。通过最大化间隔的方式，SVM 试图找到一个能够在训练数据上分类正确，并且对未见过的数据具有较好泛化能力的超平面。这种约束条件导致了模型的复杂度被限制在一定范围内，从而减少了VC维度，提高了模型的泛化能力。

###### 贝叶斯超参数优化原理

贝叶斯优化是一种近似逼近的方法，用各种代理函数来拟合超参数与模型评价之间的关系，然后选择有希望的超参数组合进行迭代，最后得出效果最好的超参数组合。通过基于目标函数的过去评估结果建立替代函数（概率模型），来找到最小化目标函数的值。贝叶斯方法与随机或网格搜索的不同之处在于，它在尝试下一组超参数时，会参考之前的评估结果，因此可以省去很多无用功。

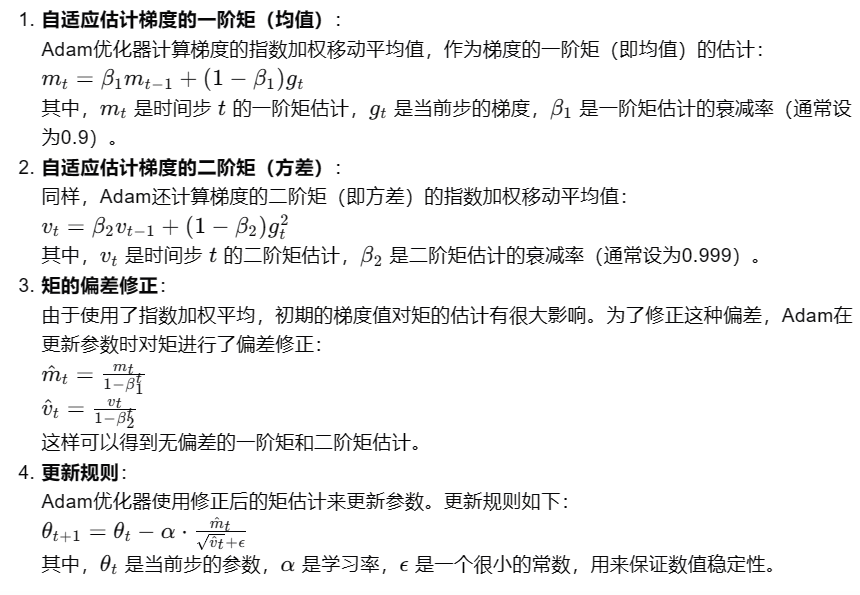
###### 数据不平衡问题处理

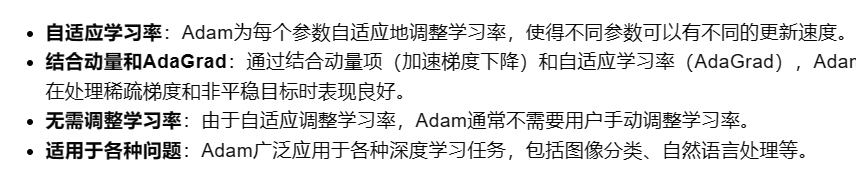
欠采样：从样本较多的类中再抽取，仅保留这些样本点的一部分；过采样：复制少数类中的一些点，以增加其基数；生成合成数据：从少数类创建新的合成点，以增加其基数。添加额外特征：除了重采样外，我们还可以在数据集中添加一个或多个其他特征，使数据集更加丰富，这样我们可能获得更好的准确率结果。

###### LR与决策树的区别

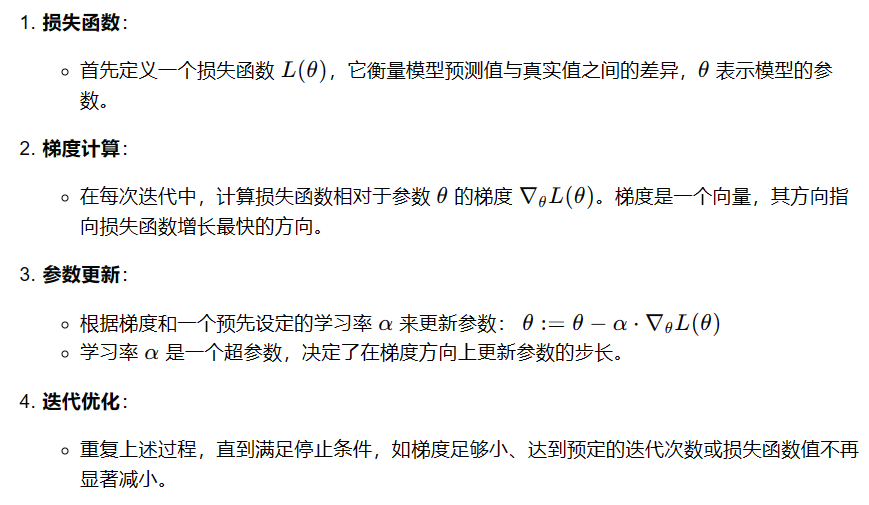
（1）逻辑回归通常用于分类问题，决策树可回归、可分类。（2）逻辑回归是线性函数，决策树是非线性函数。（3）逻辑回归的表达式很简单，回归系数就确定了模型。决策树的形式就复杂了，叶子节点的范围+取值。两个模型在使用中都有很强的解释性，银行较喜欢。（4）逻辑回归可用于高维稀疏数据场景，比如ctr预估；决策树变量连续最好，类别变量的话，稀疏性不能太高。（5）逻辑回归的核心是sigmoid函数，具有无限可导的优点，常作为神经网络的激活函数。（6）在集成模型中，随机森林、GBDT以决策树为基模型

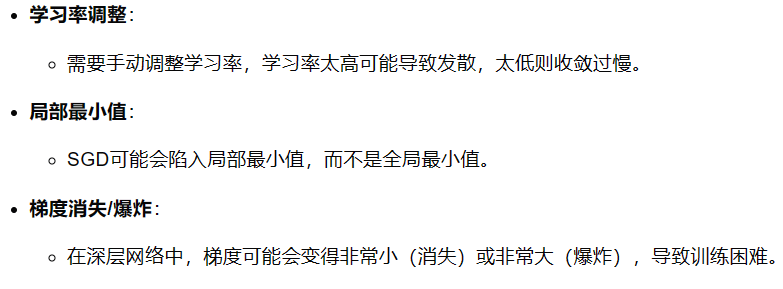
**ADAM原理**

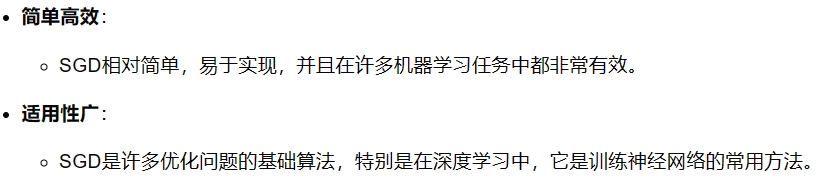




###### SGD







###### 特征向量和特征值

矩阵A当然是一个变换，然后这个变换的特殊之处是当它作用在特征向量 上的时候， 只发生了缩放变换，它的方向并没有改变，并没有旋转。

###### SVM中VC维是什么意思？

对于二分类问题而言，VC维为一个由参数为 定义的函数簇可以将训练数据正确数量称为VC维、+

VC维度衡量了一个假设空间（如分类器或回归模型）能够拟合多复杂的函数的能力。

###### 随机森林的一般运行过程：

初始化：设置森林中决策树的数量（n\_trees），这个参数通常由用户指定。

数据准备：

构建决策树：对于每棵决策树，执行以下步骤： a. 从训练数据中随机选择一个子集（通常使用自助采样，即bootstrap sampling），但不是所有树都使用完全相同的数据。 b. 对于决策树的每个节点：

随机选择一部分特征（而不是所有特征），以寻找最佳分裂点。

基于选定的特征，计算该节点的最佳分裂准则（例如，信息增益、基尼不纯度）。根据最佳分裂准则，将数据分割成子节点。让每棵树尽可能地生长，直到满足停止条件，比如达到最大深度、节点中的样本数少于某个阈值，或者无法进一步减少不纯度。

对于分类问题，每棵树给出一个预测结果，最终的预测是多数树投票的结果。对于回归问题，每棵树给出一个预测值，最终的预测结果是所有树预测值的平均。所有树的预测结果被集成起来，形成一个最终的预测模型。使用测试数据评估随机森林模型的性能，常用的评估指标包括准确率、召回率、F1分数等

###### 深度学习数据清洗流程

对数据的预处理、清洗、填补和修正等步骤，可以去除数据中的噪声、异常值、缺失值和重复值等

###### Q：类别数据怎么做特征工程

对类别特征进行编码，如独热编码（将每一个类别转化为一个二进制向量，向量中只有一个位置为1（类别存在时为1），其余均为0）、标签编码（每一个类别赋予一个唯一的整数标签）

处理大量的类别特征，如特征哈希

基于统计量的特征选择方法，如选择方差大的特征、假设检验、互信息等

###### Q：为什么要用图神经网络，用户的历史交互序列做时序模型不行吗？

相比较于神经网络最基本的网络结构全连接层（MLP），特征矩阵乘以权重矩阵，图神经网络多了一个邻接矩阵。计算形式很简单，三个矩阵相乘再加上一个非线性变换，每个节点都在接收邻居的信息。

处理非结构化数据：图神经网络擅长处理非结构化数据，与时序模型依赖于时间顺序不同，图神经网络能够同时考虑图中的全局信息和局部信息。这使得GNNs能够更好地理解整个系统的状态和结构

###### 怎么样做特征选择

特征是否发散：如果一个特征不发散，例如方差接近于0，也就是说样本在这个特征上基本上没有差异，这个特征对于样本的区分并没有什么用。

特征与目标的相关性：这点比较显见，与目标相关性高的特征，应当优选选择

Filter：过滤法，按照发散性或者相关性对各个特征进行评分，设定阈值或者待选择阈值的个数，选择特征。Wrapper：包装法，根据目标函数（通常是预测效果评分），每次选择若干特征，或者排除若干特征。Embedded：嵌入法，先使用某些机器学习的算法和模型进行训练，得到各个特征的权值系数，根据系数从大到小选择特征。

##### 深度学习

###### 卷积的理解

对卷积这个名词的理解：所谓两个函数的卷积，本质上就是先将一个函数翻转，然后进行滑动叠加。输入一幅图像f(x,y)，经过特定设计的卷积核g(x,y)进行卷积处理以后，卷积处理的结果，其实就是把每个像素周边的，甚至是整个图像的像素都考虑进来，对当前像素进行某种加权处理。

在图像的某个位置上覆盖滤波器；将滤波器中的值与图像中的对应像素的值相乘；把上面的乘积加起来，得到的和是输出图像中目标像素的值；对图像的所有位置重复此操作。在连续情况下，叠加指的是对两个函数的乘积求积分，在离散情况下就是加权求和，为简单起见就统一称为叠加。

而大小为偶数的卷积核即使对称地加padding也不能保证输入feature map尺寸和输出feature map尺寸不变

3d卷积多了一个深度通道，这个深度可能是视频上的连续帧，也可能是立体图像中的不同切

###### 池化层不可导但却要参与反向传播呢？

在反向传播时，梯度是按位传播的，那么，一个解决方法，就是如何构造按位的问题，但一定要遵守传播梯度总和保持不变的原则。对于平均池化，在反向传播时，框架需要将梯度平均分配给每一个神经元再进行反向传播，最大池化在反向传播时，框架仅需要将该区域的梯度直接分配到最大值神经元即可

###### softmax函数在梯度反向传播过程中会出现梯度的钝化，实际过程中该如何解决

使用Warm-up策略:在训练初期，逐渐增加模型的复杂度或学习率，可以让模型参数更加平滑地适应训练数据，减少梯度消失问题。

###### 梯度与导数

方向导数是各个方向上的导数，偏导数连续才有梯度存在。梯度的方向是方向导数中取到最大值的方向，梯度的值是方向导数的最大值

###### 卷积层实现

class torch.nn.MaxPool2d(kernel\_size, stride=None, padding=0, dilation=1, return\_indices=False, ceil\_mode=False)

$$

padding(int or tuple, optional) - 输入的每一条边补充 0 的层数，默认不填充。

dilation(int or tuple, optional) – 卷积核元素之间的间距，默认取值 1 。

bias(bool, optional) - 如果 bias=True，添加偏置

###### 2.3，理解边界效应与填充 padding

如果希望输出特征图的空间维度Keras/TensorFlow 设置卷积层的过程中可以设置 padding 参数值为 “valid” 或 “same”。“valid” 代表只进行有效的卷积，对边界数据不处理。“same” 代表 TensorFlow 会自动对原图像进行补零（表示卷积核可以停留在图像边缘），也就是自动设置 padding 值让输出与输入形状相同。

OpenCV 读取图像后返回的矩阵 shape 的格式是 （H, W, C）格式。当 OpenCV 读取的图像为彩色图像时，返回的多通道的 BGR 格式的矩阵（HWC）

###### CNN 权值共享问题

首先权值共享就是滤波器共享，滤波器的参数是固定的，即是用相同的滤波器去扫一遍图像，提取一次特征特征，得到feature map。

###### 如何判断深度学习模型是否过拟合

1. 首先将训练数据划分为训练集和验证集，80% 用于训练集，20% 用于验证集（训练集和验证集一定不能相交）；训练都时候每隔一定 Epoch 比较验证集但指标和训练集是否一致，如果不一致，并且变坏了，那么意味着过拟合。

2. 用学习曲线 learning curve 来判别过拟合

###### 欠拟合怎么解决

underfitting 欠拟合的表现就是模型不收敛，原因有很多种，这里以神经网络拟合能力不足问题给出以下参考解决方法：

寻找最优的权重初始化方案：如 He正态分布初始化 he\_normal，深度学习框架都内置了很多权重初始化方法；使用适当的激活函数：卷积层的输出使用的激活函数一般为 ReLu，循环神经网络中的循环层使用的激活函数一般为 tanh，或者 ReLu；

选择合适的优化器和学习速率：SGD 优化器速度慢但是会达到最优.

###### 如何判断模型是否欠拟合

神级网络欠拟合的特征就是模型训练了足够长但时间后, loss 值依然很大甚至与初始值没有太大区别，且精度很低，测试集亦如此。根据我的总结，原因一般有以下几种：

神经网络的拟合能力不足；网络配置的问题； 数据集配置的问题；训练方法出错

###### 训练过程中模型不收敛，是否说明这个模型无效，导致模型不收敛的原因

不一定。导致模型不收敛的原因有很多种可能，常见的有以下几种：

没有对数据做归一化。忘了做数据预处理。

忘了使用正则化。 Batch Size 设的太大。学习率设的不对。

最后一层的激活函数用的不对。网络存在坏梯度。比如 ReLu 对负值的梯度为 0，反向传播时，0 梯度就是不传播。 参数初始化错误。 网络太深。隐藏层神经元数量错误。

###### 损失函数概述

大多数深度学习算法都会涉及某种形式的优化，所谓优化指的是改变  以最小化或最大化某个函数  的任务，我们通常以最小化  指代大多数最优化问题。

在机器学习中，损失函数是代价函数的一部分，而代价函数是目标函数的一种类型。

- 损失函数（`loss function`）: 用于定义单个训练样本预测值与真实值之间的误差

- 代价函数（`cost function`）: 用于定义单个批次/整个训练集样本预测值与真实值之间的累计误差。

- 目标函数（`objective function`）: 泛指任意可以被优化的函数。

损失函数大致可分为两种：回归损失（针对连续型变量）和分类损失（针对离散型变量），

###### 为什么用交叉熵

在机器学习中，我们需要评估标签值 $y$ 和预测值 $a$ 之间的差距熵（即两个概率分布之间的相似性），使用 KL 散度  即可，但因为样本标签值的分布通常是固定的，即  不变。因此，为了计算方便，在优化过程中，只需要关注交叉熵就可以了。所以，在机器学习中一般直接用交叉熵做损失函数来评估模型。

###### 为什么不能使用均方差做为分类问题的损失函数？

回归问题通常用均方差损失函数，可以保证损失函数是个凸函数，即可以得到最优解。而分类问题如果用均方差的话，损失函数的表现不是凸函数，就很难得到最优解。而交叉熵函数可以保证区间内单调。

分类问题的最后一层网络，需要分类函数，`Sigmoid` 或者 `Softmax`，如果再接均方差函数的话，其求导结果复杂，运算量比较大。用交叉熵函数的话，可以得到比较简单的计算结果，一个简单的减法就可以得到反向误差

使用MSE损失函数会导致对类别之间的差异进行了平方和处理，无界输出，例如预测房价或股票价格。不符合分类问题的特性。且会有梯度消失的问题。

Log Loss：提供了一个概率框架，使得输出可以解释为概率分布。对于二分类问题，输出可以解释为属于某个类别的概率。这使得模型的输出更加直观和易于解释

###### 欧氏距离和余弦距离的区别

欧氏距离 的范围是不确定的 (没有经过归一化)，值越大表示样本之间越不相似，直度量两个向量点之间的绝对距离。 而余弦距离不考虑向量大小，只关注角度关系，取值范围 [-1,1]，当两个向量方向完全一致时取1，完全相反时取-1。 体现方向上的相对差异

###### 交叉熵原理

衡量的是在给定真实标签的条件下，模型预测概率分布与真实的概率分布之间的差异。当预测值与实际标签越接近时，交叉熵损失越小。交叉熵误差用于分类问题：当目标是预测离散的类别标签时

###### 常用的模型微调方法。

在预训练模型的某些层（通常是底层）冻结权重，不进行训练，而只训练模型的顶层或新添加的层。

全连接层替换：替换或修改模型的最后一层（通常是全连接层），以适应新任务的输出需求。

中间层微调：对模型的中间层部分或全部层的权重进行微调，以学习特定任务的特征。

多任务学习：通过共享预训练模型的底层特征，同时训练多个任务，实现知识的迁移。

###### 如何定位梯度消失的问题?

在训练过程中，使用调试工具或手动记录每层的梯度大小。如果发现梯度值非常小，接近于0，这可能是梯度消失的迹象。检查训练过程中权重的更新情况

###### 手撕Conv2D

def conv2d(X, W, b, stride=1, padding=0):

X: 输入图像，维度 (高度, 宽度, 输入通道数)W: 卷积核权重，维度 (卷积核高度, 卷积核宽度, 输入通道数, 输出通道数)b: 偏置项，维度 (输出通道数,)stride: 步长padding: 填充

inp\_height, inp\_width, inp\_channels = X.shape

kernel\_height, kernel\_width, out\_channels = W.shape

# 计算输出尺寸

out\_height = (inp\_height - kernel\_height + 2 \* padding) // stride + 1

out\_width = (inp\_width - kernel\_width + 2 \* padding) // stride + 1

# 初始化输出图像

Y = np.zeros((out\_height, out\_width, out\_channels))

# 填充输入图像

X\_padded = np.zeros((inp\_height + 2 \* padding, inp\_width + 2 \* padding, inp\_channels))

X\_padded[padding:-padding, padding:-padding, :] = X

# 卷积操作

for y in range(0, out\_height):

for x in range(0, out\_width):

for c in range(0, out\_channels):

# 计算卷积

Y[y, x, c] = np.sum(X\_padded[y \* stride:y \* stride + kernel\_height,

x \* stride:x \* stride + kernel\_width,

:] \* W[:, :, :, c]) + b[c]

return Y

###### 预训练和微调任务有什么区别？两者的目的是什么?

数据集大小：预训练使用的数据集通常非常大，而微调使用的数据集相对较小。

训练目标：预训练旨在学习通用的语言特征，减少对大量标注数据的依赖：通过预训练，可以在没有大量标注数据的情况下训练出强大的模型,而微调则专注于学习特定任务的特征。

训练阶段：预训练是模型训练的第一阶段，微调是在预训练模型的基础上进行的第二阶段。

模型参数：在预训练阶段，模型的所有参数都被训练；在微调阶段，可能只有部分参数或新添加的参数被训练。

###### 深**度网络模型中权重能不能都初始化为0**

如果所有权重都初始化为0，那么在前向传播和反向传播的过程中，每一层的所有神经元将接收到相同的输入和梯度更新，导致它们更新后的权重依然相同。这种情况下，无论网络有多少层或多少神经元，每一层的所有神经元都会执行相同的操作，相当于网络没有多个神经元的学习能力。必须用随机值初始化权重，以确保神经网络能够学习到更丰富的特征。

2. 激活函数的非线性：大多数神经网络中使用的激活函数都是非线性的。如果权重初始化为0，那么无论输入数据如何，激活函数的输入总是0，这将使得激活函数的输出在一开始是固定值，从而无法通过梯度下降进行有效的学习。

###### 数据集中正负样本不均衡的情况怎么解决

重采样：过采样少数类：增加少数类的样本数量，可以通过简单地复制现有样本或使用更复杂的技术。

欠采样多数类：减少多数类的样本数量，以减少类别不平衡的影响。

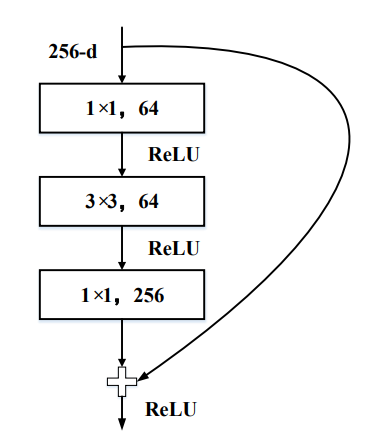
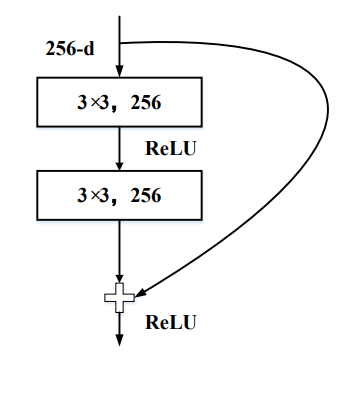
调整权重：在损失函数中为不同类别的样本分配不同的权重，使得模型更关注少数类样本。Focal Loss：专为解决类别不平衡设计的损失函数，它降低了易分类样本的权重，并增加了难分类样本的权重.

Bagging：使用集成方法，如随机森林，可以提高模型对少数类样本的预测能力。Boosting：一些提升方法，如AdaBoost，可以通过在训练过程中增加对少数类样本的关注来解决不平衡问题。

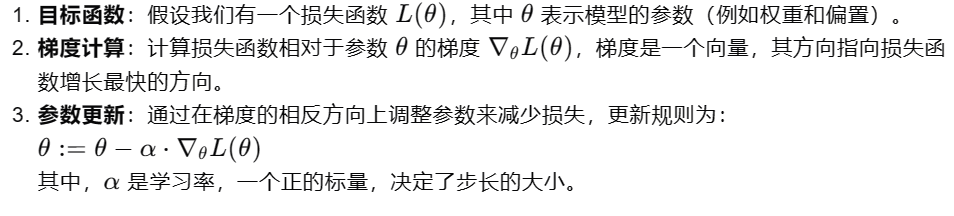
调整分类阈值，使得模型对少数类样本更加敏感。

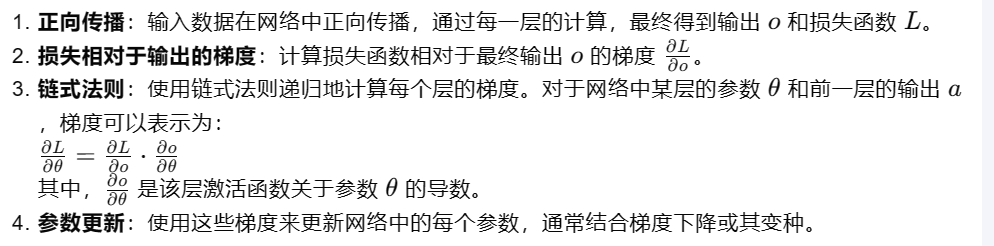
###### resnet

假设神经网络的输入层是x，h(x)是在普通卷积层中学习的映射关系，则残差学习需要学习的目标就是输出和输入的残差函数而非原始函数, 在卷积神经网络卷积层的基础上引入了跳跃连接，绕过一些层的同时产生一个残差层，直接将输入信息绕道传到输出，能够简化复杂的深度网络训练过程，并且也避免了由于网络过深引起的梯度消失和退化. 11卷积作用是改变输入向量的维度，33卷积层可以用来进行学习特征

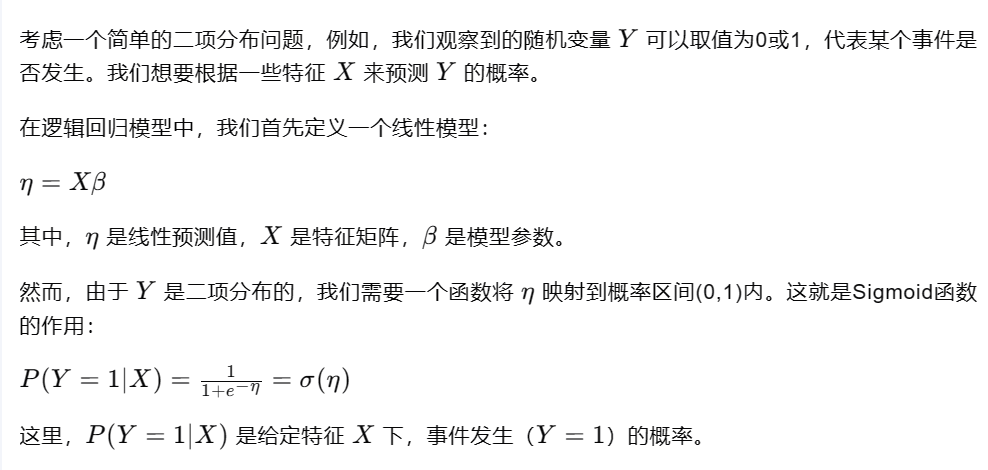
 

###### 从数学角度解释下梯度下降、反向传播





###### 为什么激活函数常使用sigmoid（希望答的是统计学推导）



###### batch normalization的优点与缺点；

BN：Batch Normalization是对一个batch的所有样本的每个特征分别进行归一化，

LN：Layer Normalization是对每个样本的所有特征统一做归一化。

BN的优点：1.可以解决“Internal Covariate Shift”2.解决梯度消失的问题（针对sigmoid），加快收敛速度

BN的缺点：1.batch size小的时候估算的统计值是不合理的 2.不适用于RNN，因为RNN的输入是动态，长度不一致，导致计算的估值不合理.

LN的优点：1.不依赖于batch size。2.适用于RNN

LN的缺点：1.不适用于CNN。2.针对具有多个连续特征的数据，进行特征之间的缩放，可能会导致量纲差异消失。

###### 介绍一下神经网络的初始化方法。

常见的神经网络初始化方法有：随机初始化（权重随机设定，如高斯初始化或均匀分布初始化）、Xavier/Glorot 初始化（权重初始化为均值为0，方差为1/n（n为输入神经元的数量）的正态分布或均匀分布）、He 初始化（类似于Xavier，但方差为2/n，适用于ReLU激活函数）等。

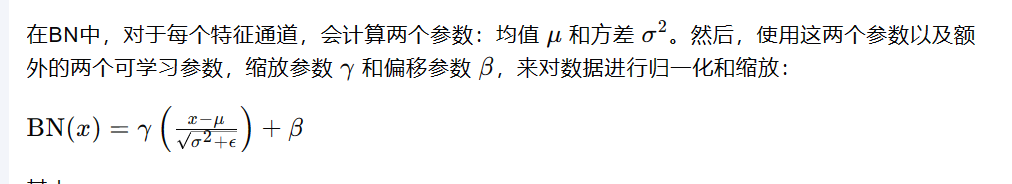
###### 使用1×3和3×1卷积核代替传统的3×3卷积核：

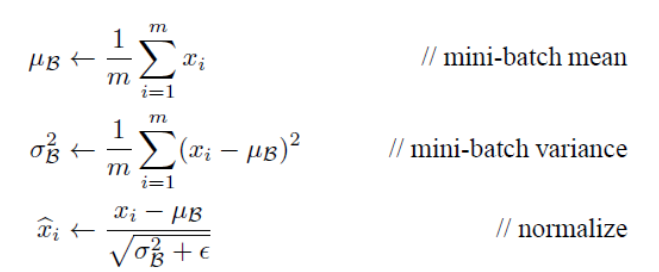
**参数数量**1×3和3×1卷积核的参数数量通常少于3×3卷积核。例如，如果输入特征图的深度为C，那么3×3卷积核有 3×3×𝐶×𝑀3×3×C×M 个参数（其中M是输出通道数），而1×3和3×1卷积核分别有 3×𝐶×𝑀 和 𝐶×𝑀C×M 个参数。计算量也相应减少，这可以提高网络的运行效率。

多尺度特征提取：1×3卷积核可以捕捉水平方向上的特征，而3×1卷积核可以捕捉垂直方向上的特征。这种分离使得网络能够同时学习到不同方向的特征，有助于提取更丰富的特征表示。

网络深度：通过使用1×3和3×1卷积核，可以增加网络的深度而不显著增加参数数量，这有助于学习更复杂的特征表示。减少过拟合。

###### 怎么调整（bn的两个参数） 算法





###### 深度可分离卷积

depthwise卷积和pointwise卷积组合而成，DW卷积操作中，每一个卷积核是只负责一个通道的，这与普通卷积一个卷积核负责几个通道是不一样的

###### 空洞卷积

不增加参数和模型复杂度的条件下，可以指数倍的扩大视觉感受野（每一个输出是由诗句感受野大小的输入决定的）的大小

###### 分布式SGD的主要思想

将训练的过程分散到多个workers中，而server将各worker上传的梯度用于参数更新，再将结果传回各worker

###### 说一下 BN 的归一化操作放在激活前还是激活后，两种有什么区别?

BN通常放在激活函数之前，这是因为它能够更有效地稳定训练过程。由于激活函数可能会产生非线性变换，这可能会使得归一化的效果不如在激活前进行的那么有效。

### C

###### 指针和引⽤的区别

● 指针和引⽤都是⼀种内存地址的概念，但是指针是⼀个实体，可以声明为 void；引⽤只是⼀个别

名，不分配存储空间，不能为 void。

● 引⽤和指针都可以作为参数传递给函数，传递指针的实质是传值，传递的值是指针的地址；传引⽤的

实质是传地址，传递的是变量的地址。

● 指针可以有多级指向，但是引⽤只能有⼀级引⽤。

###### C 和 C 区别

● C 是⾯向过程的，C 是⾯向对象的。因此C语⾔中有类和对象、继承和多态这样的OOP语⾔必

备的内容，此外C还⽀持模板，运算符重载以及STL；

● 在输⼊输出⽅式上，C 是 printf/scanf 库函数，C 是 cout/cin，即 ostream和 istream 类型的

对象；

● 在动态内存管理上，C 语⾔通过 malloc/free 来进⾏堆内存的分配和释放，⽽ C 是通过new/delete 来管理堆内存；

● 在C中，struct 关键字不仅可以⽤来定义结构体，也可以⽤来定义类；

● C不仅⽀持指针，还⽀持更安全的引⽤。不过在汇编代码上，指针和引⽤的操作是⼀样的；

● C⽀持⾃定义命名空间，⽽ C 不⽀持。

###### C 和 Pyhton 区别

● C为编译型语⾔；python为解释型的脚本语⾔。

● C运⾏效率⾼。Python是解释执⾏的，和物理机CPU之间多了解释器这层，⽽C是编译执⾏

的，直接就是机器码，编译的时候编译器⼜可以进⾏⼀些优化。

● 开发效率上，Python要⽐C快很多

###### 重载重定义重写

重载：函数名相同，函数的参数个数、参数类型或参数顺序三者中必须⾄少有⼀种不同。函数返回值的

类型可以相同，也可以不相同。发⽣在⼀个作⽤域内。

：也叫覆盖(override)，⼀般发⽣在⼦类和⽗类继承关系之间。⼦类重写⽗类中有相同名称和参数的

虚函数。

重定义：⼦类重新定义⽗类中有相同名称的⾮虚函数 ( 参数列表可以不同 ) ，派⽣类的函数屏蔽了与其同名的基类函数。

###### 继承

它允许创建一个新的类（称为派生类或子类）来扩展或修改现有类（称为基类或父类）的功能。继承支持代码的复用，并且可以构建出一个类层次结构。

public：派生类继承基类的所有公共成员，并能访问基类的公共和受保护的成员。

protected：派生类继承基类的公共成员为受保护成员，并能访问基类的公共、受保护的成员。

private：派生类继承基类的公共成员为私有成员，并只能通过基类的公共和受保护接口访问基类的成员。

基类保护：基类中的私有成员（private）不能被派生类直接访问，但可以通过基类提供的公共或受保护的接口进行操作。

多态性：继承支持多态性，即派生类对象可以赋值给基类对象，且派生类重写了基类中的虚函数时，可以通过基类指针或引用来实现多态行为。

虚函数和纯虚函数：虚函数允许在基类中声明，并在派生类中提供具体实现，纯虚函数用于创建抽象基类。

###### 构造函数和析构函数在继承中的使用：

构造派生类对象时，首先调用基类的构造函数（如果有的话），然后调用派生类的构造函数。

析构派生类对象时，首先调用派生类的析构函数，然后调用基类的析构函数。

使用virtual关键字在基类中声明函数，使得该函数在派生类中可以被重写。

通过基类的指针或引用调用虚函数时，将根据对象的实际类型来动态确定调用哪个函数

###### fork

将⽗进程复制⼀份给⼦进程，⼦进程从 fork 调⽤处继续执⾏，之后的代码在⽗⼦进程中各⾃执⾏

⼀遍。最终⽗进程的 fork 返回⼦进程的 pid，⼦进程的 fork 返回 0 表示创建成功。所以看起来仿佛

fork 有两个返回值，其实是两个进程的 fork 各⾃的返回值。

exec 函数族可以根据指定的⽂件名或⽬录名找到可执⾏⽂件，并⽤它取代原调⽤进程的数据段、代码

段和堆栈段。在执⾏完后，原调⽤进程除了进程号外，其他全部被新程序的内容替换。

wait 会暂时停⽌当前进程的执⾏，直到有信号或⼦进程结束。如果⼦进程已经结束，则 wait 会⽴即返

回⼦进程结束状态值。

###### 虚函数：/纯虚函数：

虚函数：在基类中冠以关键字 virtual 的成员函数。 它提供了一种接口界面。允许在派生类中对基类的虚函数重新定义。

纯虚函数的作用：在基类中为其派生类保留一个函数的名字，以便派生类根据需要对它进行定义。作为 接口而存在，纯虚函数不具备函数的功能，一般不能直接被调用。

###### 虚函数表（vtable）：

每个含有虚函数的类都会有一个虚函数表，这是一个存储虚函数地址的数组。

动态绑定：当通过基类指针或引用调用虚函数时，程序会在运行时查找对象的虚函数表，并调用相应的函数。

###### 数组、指针区别

1. 数组存放的是数据，是直接访问数据的；指针存放的是变量的地址，是间接访问数据的；

2. 数组通常存储在静态存储区或栈上；指针可以随时地指向任意类型的内存块；

数组指针（指向数组的指针）

1. 数组在内存中的表示：创建⼀个数组就是在内存⾥⾯开辟⼀块连续的空间；

指针数组（存放指针的数组）

###### static关键字

意味着这个变量是类的所有实例共享的，而不是属于某个特定实例的成员变量。类的所有对象共享同一个静态变量，因此，对静态变量的修改会影响到所有对象。

###### 迭代器：

⽤于提供⼀种⽅法顺序访问⼀个聚合对象中各个元素，⽽⼜⽆需暴露该对象的内部表示。

###### 迭代器和指针区别：

迭代器不是指针，是类模板，相当于⼀种智能指针，可以根据不同类型的数据结构来实现递增、递减等操作。迭代器返回的是对象引⽤⽽不是对象的值

482. 堆和栈的区别？

堆存放动态分配的对象——即那些在程序运行时动态分配的对象，其生存期由程序控制，必须手动申请，手动释放。

栈用来保存定义在函数内的非static对象，如局部变量，仅在其定义的程序块运行时才存在，由编译器自 动创建和销毁。

###### strcpy、sprintf 与 memcpy 的区别

三者主要有以下不同之处：

（1） 操作对象不同，strcpy 的两个操作对象均为字符串，sprintf 的操作源对象可以是多种数据类型， 目的操作对象是字符串，memcpy 的两个对象就是两个任意可操作的内存地址，并不限于何种数据类型。

（2） 执行效率不同，memcpy 最高，strcpy 次之，sprintf 的效率最低。

（3） 实现功能不同，strcpy 主要实现字符串变量间的拷贝，sprintf 主要实现其他数据类型格式到字符串的转化，memcpy 主要是内存块间的拷贝。

###### const 是什么静态函数

const 用来定义一个静态函数（Static Functions），只读的变量或对象。主要优点：便于类型检查、方便地进行参数的修改和调整、节省空间， const修饰成员函数不可修改成员变量。

静态函数是与类相关联的函数，而不是与类的某个特定实例相关联。这意味着静态函数可以通过类名直接调用，而不需要创建类的实例。静态函数中不能访问类的非静态成员变量和非静态成员函数。

###### 面向对象的认识

面向对象编程将程序构建为一系列相互作用的对象。每个对象包含数据（属性）和操作这些数据的方法。在面向对象编程中，数据和函数（方法）是结合在一起的，形成一个对象。重点在于创建对象，这些对象是现实世界实体的抽象。通过继承和多态性来实现代码重用。继承允许新类（子类）继承现有类（父类）的特性，而多态性允许子类重写父类的方法。

###### 面向过程编程：

面向过程编程将程序分解为一系列的过程或函数。每个函数完成一个具体的任务。数据和函数分离：在面向过程编程中，数据和函数是分离的。函数操作数据，但数据不是函数的一部分。

重点在于设计一系列的过程来处理数据。通过将通用功能封装在函数中来实现代码重用

面向对象是相当于面向过程而言的，面向过程语言是一种基于功能分析的，以算法为中心的程序设计方

法，而面向对象是一种基于结构分析的，以数据为中心的程序设计思想。

###### 面向对象的三大特征

面向对象的三大特征是封装性、继承性和多态性：

封装：隐藏对象的内部状态和复杂性，只暴露操作该对象的接口。将客观事物抽象成类，

继承：允许创建分层的类结构，子类可以继承父类的属性和方法。

多态性：允许使用统一的接口来对不同类型的对象执行同一个操作。将父类对象设置成为和一个或更多它的子对象相等的技术。用子类对象给父类对象赋值之后，父类对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。

###### 19. 链表和数组有什么区别

（1） 存储形式：数组是一块连续的空间，声明时就要确定长度。链表是一块可不连续的动态空间，长度可变，每个结点要保存相邻结点指针。

（2） 数据查找：数组的线性查找速度快，查找操作直接使用偏移地址。链表需要按顺序检索结点，效 率低。

（3） 数据插入或删除：链表可以快速插入和删除结点，而数组则可能需要大量数据移动。

（4） 越界问题：链表不存在越界问题，数组有越界问题。

数组便于查询，链表便于插入删 除。数组节省空间但是长度固定，链表虽然变长但是占了更多的存储空间。

###### 定义和声明的区别

1. 如果是指变量的声明和定义

从编译原理上来说，声明是仅仅告诉编译器，有个某类型的变量会被使用，但是编译器并不会为它分配 任何内存。而定义就是分配了内存。

###### 构造函数的作

构造函数是类的特殊成员函数，用于在创建对象时初始化对象的状态。它的主要作用包括：初始化成员变量。资源分配：如果对象需要管理资源（如内存、文件句柄等），构造函数可以用来分配这些资源。

析构函数的作用：析构函数是类的另一个特殊成员函数，它在对象生命周期结束时被调用，用于执行清理工作。它的主要作用包括：释放资源：释放对象在生命周期内分配的所有资源。确保对象使用的所有动态分配的内存被正确释放。

###### 析构函数何时被调用：

对于自动对象（局部对象），当它们的作用域结束时，析构函数被调用。对于动态分配的对象（使用new关键字创建的对象），当显式调用delete操作符时，析构函数被调用。异常发生在对象生命周期中，析构函数会在异常处理过程中被调用。对于全局或静态对象，程序结束时，它们的析构函数将被调用。

###### C++中的map实现：

通过红黑树（一种自平衡的二叉搜索树）实现的。红黑树保证了元素的有序性

## 推荐方向

###### 你所理解的推荐系统是什么样的？大致流程？所用模型？

推荐系统是一种通过分析用户的历史行为、个人喜好、兴趣爱好等数据信息，为用户推荐个性化的产品、服务、信息等的技术系统。**推荐系统的目的旨在为 user (用户) 推荐一个（或一系列）未观测的 item (物品，电影，新闻等)。**基本步骤如下：

学习 user 和 item 的向量表示根据 1 中的 user 和 item 向量表示，计算表示 user对 item 的偏好得分，得分函数可以采用內积、DNN等。基于 2 中计算的得分，进行排序推荐。

**流程：**

召回 从海量物品中快速找回一部分重要物品召回,获取粗粒度的用户兴趣，模型要求高准确。

减少计算量：通过从大量候选集中快速筛选出一小部分相关的选项，减少了对排序阶段的计算需求。

提高效率：召回阶段可以使用粗粒度的特征和简单的模型快速过滤，而排序阶段则可以使用更复杂的模型和细粒度的特征对这些选项进行精细化排序。

粗排阶段: 针对这上千/万条item，进行第一遍打分，保证一定精准度并减少物品数量。这个阶段的排序模型一般都比较简单，能够过滤掉一些与用户兴趣明显不相关的。

精排阶段: 得到几百条item后，精排阶段会建立相对精细的模型，根据用户的画像，偏好，上下文，结合业务目标进行排序。一般精排后返回50-100条给到engine侧。

重排阶段: engine 侧拿到精排的50条item。还会做很多的人工干预和产品逻辑，热门，置顶，多种内容之间的位置混合等等。最终会返回5-10条左右的item在线流程还有许多比较细的模块，比如去重服务，避免给用户推荐重复的内容。特征预处理，特征抽取等模块目的改进用户体验，生成最终推荐列表之前对候选推荐项进行优化，以提高推荐的准确性、多样性和用户满意度。

###### 协同过滤

所谓协同过滤算法，基本思想是根据用户的历史行为数据的挖掘发现用户的兴趣爱好，基于不同的兴趣爱好对用户进行划分并推荐兴趣相似的商品给用户。协同过滤算法主要分为两类：

- 基于物品的协同过滤算法：给用户推荐与他之前喜欢的物品相似的物品

计算物品之间的相似度，根据物品的相似度和用户的历史行为给用户生成推荐列表（购买了该商品的用户也经常购买的其他商品）

- 基于用户的协同过滤算法：给用户推荐与他兴趣相似的用户喜欢的物品

- 找到与当前用户A相似的用户B，将相似用户B喜欢的物品而用户A没有见过的物品推荐给用户A。

###### 矩阵分解算法

则是期望为每一个用户和视频生成一个隐向量，将用户和视频定位到隐向量的表示空间上（如图右所示），距离相近的用户和视频表明兴趣特点接近，在推荐过程中，我们就应该把距离相近的视频推荐给目标用户。

###### 推荐系统中的排序算法

Pointwise方法：将排序问题视为回归问题，直接预测每个物品的得分。例如，使用线性回归或逻辑回归模型来预测用户对物品的评分或点击概率 。

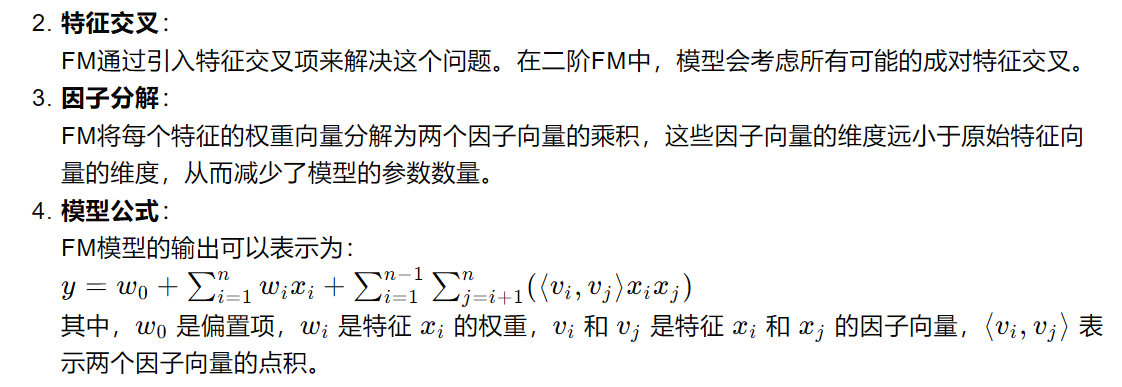
Pairwise方法：将排序问题视为排名问题，通过成对比较来训练模型，使得用户更感兴趣的物品排名高于不感兴趣的物品 。

Listwise方法：将整个候选列表作为输入，模型需要考虑列表中所有物品的相对顺序，学习如何优化整个列表的排序

###### FM和deepFM

FM的核心思想是将高维的特征向量通过因子分解来表示，从而捕获特征间的交互信息。

FFM通常用于预测用户是否会点击某个广告或者是否会购买某个商品。



因子分解机（Factorization Machine, FM）和深度神经网络（Deep Neural Network, DNN）的推荐算法，主要用于处理推荐系统中的点击率（CTR）预估问题。

嵌入层（Embedding Layer）：将稀疏的离散特征通过嵌入层转换为低维稠密向量。不同特征的嵌入向量长度相同，便于后续的计算。因子分解机部分（FM Part）：FM部分负责提取低阶特征，特别是二阶特征交互。FM通过隐向量（latent vector）的点积来捕捉特征间的交互。FM的输出由两部分组成：线性部分和二阶交叉部分。线性部分是特征的线性组合，而二阶交叉部分则是特征向量的点积。深度神经网络部分（DNN Part）：DNN部分负责提取高阶特征交互。它通常由多层全连接层组成，可以捕捉更复

###### 说一下图神经网络。

GNN中最主要的算子是利用图结构进行embedding传播，包括聚合邻居节点的embeddings（fusion操作）以及目标节点的embedding，并通过多个图卷积层来实现graph embeddings的更新。

把user、item当成一个节点如果存在购买行为，则存在边，用邻接矩阵存储交互信息，之后进行邻居消息的传播聚合，可以简单理解为两个大矩阵的矩阵乘法

###### 双塔模型（Two-Tower Model）

在推荐系统中广泛应用于召回阶段，其核心思想是将用户（User）和物品（Item）分别表示为两个独立的塔（或称为编码器），每个塔学习将用户或物品的特征映射到一个低维向量空间中。具体原理如下：

用户塔和物品塔：用户塔学习用户特征的表示，而物品塔学习物品特征的表示。

负样本构造：在召回阶段，正样本是真实的正例，而负样本可以通过采样得到。

线上预测：在线上预测时，通常会将物品塔的输出向量存储起来，用户塔实时计算用户向量，然后通过检索系统找到与用户向量最相似的物品向量。

###### Wide & Deep排序模型的原理

Wide & Deep模型是一种结合了线性模型（Wide部分）和深度神经网络（Deep部分）的混合模型，用于推荐系统中的排序任务。其原理如下：

线性模型部分，它可以捕捉特征之间的交叉和交互，适合记忆性强的特征，即那些在历史数据中频繁出现的特征组合。

Deep部分：深度学习模型部分，使用神经网络（特别是embedding层）来学习特征的低维表示，即那些在历史数据中不常见或需要探索的特征组合。

联合训练：Wide部分和Deep部分在同一模型中联合训练，共享相同的输出层，这样可以互补两者的优势，线性部分提供泛化能力，而深度部分提供记忆能力。

特征表示：Deep部分通过embedding层来表示离散型特征，而Wide部分则可以处理连续型特征和one-hot编码的离散型特征。

###### 召回和排序负样本的区别

在召回阶段，负样本通常是指那些没有被选入候选集的物品，因此大多数物品在这个阶段被视为负样本

负样本是指那些与用户当前兴趣不符，或者相对于其他候选项来说优先级较低的物品。

###### 推荐系统中为什么用GCN？

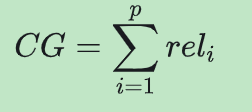
数据一般会采集成多张表的形式，比如用户-item交互表（评价，点击和购买），用户信息表（年龄，性别和收入），item属性表（品牌，种类，价格）。传统的推荐系统不能同时处理多张表的数据，而GNN模型可以用统一的方法来处理这些表数据，并且能学习到users，items和其他特征的高质量的embeddings。

推荐系统的准确度依赖于对用户和items的相似度计算，这些similarity会映射到学习到的embedding空间。如果这些items与其他users间有交互，则这些users和当前使用这些item的user有相似偏好，而GNN能够有效捕获高阶的连接信息。

GNN模型可以将监督信号的处理问题升到半监督信号处理问题。比如带目标标签的购买行为相较于其他无标签的行为（搜索，添加到购物车）来说是相当稀疏的。传统的推荐系统仅对目标行为（监督信号）进行学习，模型性能不好，而GNN模型可以结合多种非目标行为，对图中半监督信号（目标行为带标签，其他行为不带标签）进行编码。

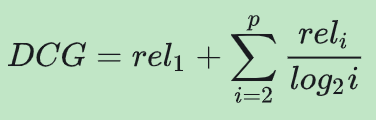
###### 召回评价指标

c**umulative gain（CG）**可翻译为累积增益，只考虑关性而没有考虑位置的影响。其计算方式也比较简单，对搜索相关性进行求和，公式如下:

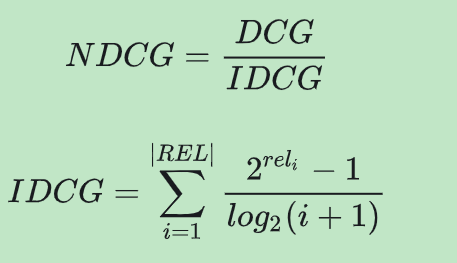


表示第i个位置的item是否是用户喜欢的，喜欢为1，否则为0。CG只能反映总体情况，即反映该页面中推荐的所有item的总分是好是坏。

**discounted CG简称为DCG**m计算方式考虑了位置因素。计算公式如下，第1位不需要衰减，而随着位次不断靠后，其影响力也不断减弱。



**NDCG归一化折损累计增益:**在搜索和推荐任务中，系统常返回一个item列表。衡量这个返回的列表是否优秀,进行不用用户之间的对比，因此需要对其进行标准化



IDCG表示的是召回的集合中的item用户都喜欢的最理想情况。所以IDCG中的分子部分其实都是1。

###### ****推荐系统中的去偏（Debias）****

推荐系统中的偏差可能来自多种因素，包括数据偏差、算法偏差和用户偏差。以下是一些去偏的策略：

1. **数据预处理**：在训练模型之前，对数据进行清洗，移除或修正偏差数据。
2. **多样性和公平性**：优化目标函数，增加多样性和公平性的考量，以减少对特定群体的偏差。
3. 。推荐系统去偏的方法主要有逆权重分数、加入无偏数据、收集随机数据或强化学习等

选择偏差是因为用户可以自由选择要评分的哪些物品，因此观测到的评分不是所有评分的代表性样本。

一致性偏差：指用户趋向于同他一个群体的用户评相似的分数（比如受到朋友的影响），尽管这可能违背了用户自身的判断，这使得这些评分无法反映用户的真实偏好

一致性偏差：指用户趋向于同他一个群体的用户评相似的分数（比如受到朋友的影响），尽管这可能违背了用户自身的判断，这使得这些评分无法反映用户的真实偏好

曝光偏差：使得高相关度的物品更可能被曝光； 4）流行物品更可能被用户看到，因此流行度偏差是曝光偏差另一种形式。

流行度偏差（Popularity Bias）指流行物品被推荐得更频繁

###### 粗排有哪些指标？

召回能力、排序能力和效率指标。召回能力可以衡量粗排和精排的一致性，防止粗排排名靠前的物品无法在精排阶段透出。可以通过如下方法计算：选取一个物品候选集，同时给粗排和精排打分并排序，二者Top-K物品重合个数，除以总物品个数，即为召回率。

排序能力的评估可以采用AUC和GAUC等指标，与精排基本一致，此处不再赘述。

###### AUC怎么求，实际意义

AUC能反映模型的排序能力，他反应的是一个相对性，即item a排在item b之前的能力；

###### 推荐系统的主要类型是什么？

基于内容的推荐系统

基于内容的推荐系统根据用户的历史行为和个人信息，推荐与其喜好和兴趣相关的内容。

协同过滤推荐系统

协同过滤推荐系统根据用户历史行为和兴趣，将具有类似兴趣的用户或物品进行匹配，从全局 user 和 item 的交互数据中学习他们的向量表示，推荐相似用户或物品的内容。

基于社交网络的推荐系统

基于社交网络的推荐系统根据用户的社交关系和交互行为，推荐与其社交网络相关的内容

基于知识图谱的推荐系统

知识图谱可以用来表示实体之间的关系，可以将 item 及其属性信息映射到知识图谱中，以理解 item 之间的相互关系。知识图谱可以扩展一个用户的历史。知识图谱是一种有向信息异构网络，其中节点代表实体，有向边可表示实体间的关系。

###### 图结构与推荐结合

图嵌入模块，主要利用图嵌入的方法学习知识图谱表征向量引入到推荐系统中，扩充用户、物品的语言表征知识图谱中实体和关系的表征；和推荐模块，以建模用户对物品的偏好

利用图谱中实体间的连接方式做推荐，这类方法大多是将包含物品属性的知识图谱与用户-物品的交互矩阵相结合，构建用户-属性-物品图，挖掘用户和物品间的多种连接关系

**同构图**是指每条边只关联两个节点，而且只有一种节点类型和一种边的类型

**异构图**是指每条边只关联两个节点，而且节点类型或者边的类型可以有多种

**超图**是指每条边可以关联多个节点

​ 关系型数据通常用图表表示，但随着非结构化数据的出现（比如图片和文本），它们不显式地包含图结构，因此有必要手动定义节点和边来构造图。比如对于文本，我们可以将单词定义为节点，将IF-IDF（词频-逆文档频率）定义为边。

###### 排序和召回的区别

召回的作用是从大量的候选物品中（例如上百万）筛选出一批用户较可能喜欢的候选集 （一般是几百）。排序又分为粗排和精排，粗排就是选出打分最高的那一部分物品。更进一步，对粗排得到的物品列表，可能需要人工调整，这就是精排。

###### 稀疏性问题：

实际场景中，用户和物品的交互信息往往是非常稀疏的。如电影推荐中，电影往往成千上万部，但是用户打过分的电影往往只有几十部。使用如此少的观测数据来预测大量的未知信息，会极大增加过拟合的风险。解决方法：是引入一些额外信息。比如社交网络，用户\商品的属性信息，多媒体信息，以及一些上下文信息

###### 冷启动问题：

对于新加入的用户或者物品，其没有对应的历史信息，因此难以进行准确的建模和推荐。

###### **推荐系统冷启动问题，怎么解决**？

1. 提供非个性化的推荐

最简单的例子就是热门排行榜，我们可以给用户推荐热门排行榜，然后等到用户数据收集到一定的时候，再切换为个性化推荐。

2. 利用用户注册信息

用户注册时提供包括用户的年龄、性别、职业、民族、学历和居住地等数据，做粗粒度的个性化。有一些网站还会让用户用文字描述他们的兴趣。

3. 利用社交网络信息

引导用户通过社交网络账号登录（需要用户授权），导入用户在社交网站上的好友信息，然后给用户推荐其好友喜欢的物品。

**多目标混排算法有哪些**

基于规则的混排：这种策略通常涉及一些预定义的规则，如广告定坑、特定类目提权等，通过规则来确定不同推荐结果的展示顺序和位置。

基于加权的融合：在多个推荐目标（如点击率、时长、完播率）中，通过加权的方式将不同目标的预估分融合起来，然后进行排序。加权系数可以通过AB测试调整并固定。

共享Embedding多塔结构：底层共享embedding，每个目标有独立的塔（模型部分），通过这种方式来优化不同的目标，并在最后进行融合。

上下文感知的重排：在重排阶段，可以考虑上下文信息，如用户行为、item之间的关系等，通过pairwise或listwise的方法来优化最终的推荐列表。

基于模型的混排：使用机器学习模型来预测不同推荐结果的相关性，并根据预测结果进行混排优化。

**多目标优化：**

在推荐系统中，多目标优化通常指的是同时考虑多个目标（如点击率、转化率、用户满意度等），并使用特定的算法来平衡这些目标，以达到最佳的综合效果。

**GNN用于推荐系统的场景？**

用户-物品协同过滤：GNN可以捕捉用户和物品之间的交互关系，通过图结构数据来增强用户和物品的表示，从而提高推荐的准确性。

序列化推荐：在序列化推荐中，GNN能够捕捉用户行为序列中的模式，例如，用户观看视频或购买商品的顺序，以预测用户可能感兴趣的下一个项目。

社交推荐：利用用户的社交网络信息，GNN可以模拟社交关系对用户偏好的影响，进而提供基于社交关系的推荐。

会话推荐：在会话推荐中，GNN可以处理用户的短期行为模式，如在一次购物会话中的点击或浏览行为，以提供更加及时和相关的推荐。

捆绑推荐：GNN可以用于推荐一组相关联的物品作为捆绑包，例如，推荐一套电子产品或一系列旅游服务。

多行为推荐：在多行为推荐中，GNN可以处理和分析用户展现的多种行为信号，如点击、评分、浏览等，以提供更全面的推荐。

**召回主流的做法，推荐系统冷启动怎么解决**

基于内容的推荐：通过分析用户的历史行为或物品的特征，为新用户或新物品提供推荐。

协同过滤：利用用户或物品之间的相似性进行推荐，即使这些用户或物品的数据很少。

物品属性：根据新物品的属性，如类别、标签、品牌等，推荐给可能感兴趣的用户。

请求用户反馈：在用户注册或首次使用时，请求用户提供一些关于他们兴趣的信息，或对一些物品进行评分。使用热门物品推荐一些热门或高评分的物品给新用户，因为这些物品被广泛接受。

基于图的推荐：构建用户-物品交互图，并使用图算法来发现潜在的推荐。

上下文感知推荐：考虑用户使用推荐系统时的上下文信息，如时间、地点、设备等，

跨域推荐：如果新用户在其他领域有行为数据，可以尝试将这些数据迁移到当前领域进行推荐。

社交网络信息：利用用户的社交网络信息，如好友推荐或社交影响，来提供推荐。

**推荐系统一定需要召回吗？**

小型数据集：如果候选集足够小，可以直接应用排序模型，而不经过复杂的召回过程。

简单场景：在一些简单的推荐场景中，可能直接使用基于规则的方法方法来生成推荐列表。

实时推荐：在需要极高实时性的系统中，可能会尽量减少模型的复杂度，直接进行排序而省略召回。

**召回和排序比较大的差异点**

**目标、上下文**

召回：可能使用较为简单或广泛适用的特征，以便于快速处理大量数据。

排序：可以使用更丰富、更细粒度的特征，包括用户行为的详细模式、物品的深层属性等。

召回：为了效率，通常使用较简单的模型，如基于内容的推荐、协同过滤或一些浅层机器学习模型。

排序：可以使用更复杂的模型，如深度学习模型、集成学习方法或多任务学习模型，以提高推荐的准确性。

**为啥排序会比召回的结果要准呢**

而排序阶段则对这些候选物品进行更精细的评估，使用更复杂的模型和特征来确定最终的推荐列表。排序模型通常会集成更多个性化的信息，如用户的详细历史行为、偏好、上下文信息，会同时考虑多个目标，如点击率、转化率、用户满意度。更好地考虑当前上下文，如时间、地点、用户当前的情境

## LSTM

###### 记忆（candidate memory）是使用tanh激活函数生成的，原因包括：

如前所述，tanh将数据压缩到-1到1之间，有助于控制信息的动态范围。tanh的压缩特性有助于防止梯度消失，因为即使在反向传播过程中，梯度也不会变得非常小。tanh确保了这个更新在一个小的范围内，有助于模型学习到更细粒度的状态更新。

###### LSTM中有没有用BN？

BN是不适用于RNN这种动态结构的。如果将BN应用在RNN中，由于BN是以BN为单位统计归一化变量的，所以不同长度的样本的时间片如何计算将十分困难。对一个batch内的每个position做标准化。多个sequence的同一个position，很难说来自相似的分布。所以RNN中BN很难学到合适的μ和σ。

## Python

###### python的列表生成器

squares = [x\*\*2 for x in range(10)]

###### shell的指定文本查找

grep "要查找的文本" 文件名

###### python 垃圾处理机制

在Python中，主要通过引用计数进行垃圾回收；

通过 “标记-清除” 解决容器对象可能产生的循环引用问题；

通过 “分代回收” 以空间换时间的方法提高垃圾回收效率。

也就是说，python采用的是引用计数机制为主，标记-清除和分代收集（隔代回收）两种机制为辅的策略。

###### yield 关键字作用

yield是一个类似 return 的关键字，只是这个函数返回的是个生成器，可以节省巨大的时间、空间开销。

**python 多继承，两个父类有同名方法怎么办？**

super(Classname,self).methodname()或super(Classname, cls).methodname() 调用"下一个"父类中的方法

###### python异常处理

try块：

代码首先尝试在try块中执行。如果try块中的代码引发异常，则转而执行except块。

except块：

except块用于捕获并处理特定的异常。你可以为不同的异常类型定义多个except块。

###### 进程与线程

进程是资源分配和调度的基本单位，线程是程序执行和调度的基本单位

多线程是指在同一个进程中并行运行多个线程，每个线程可以执行不同的任务。

特点：线程之间共享内存和资源，因此线程之间的通信成本较低。

多进程是指创建多个独立的进程，每个进程有自己的栈和程序计数器。进程之间不共享内存，因此进程之间的通信需要使用进程间通信（IPC）机制，如管道、队列等。每个进程有自己的Python解释器和内存空间，因此不受GIL的限制，可以充分利用多核处理器的计算能力。

###### python怎么快速将channel反向

列表名.reverse() 列表名[x:y:z]

###### 函数装饰器

python装饰器就是用于拓展原来函数功能的一种函数，这个函数的特殊之处在于它的返回值也是一个函数，使用python装饰器的好处就是在不用更改原函数的代码前提下给函数增加新的功能。使用时，再需要的函数前加上@demo

**8、熟悉那些分布式训练框架，目前在用那些，能否详细说明？**

DDP是PyTorch中用于在多个GPU或多个计算节点上并行训练的分布式训练技术。它通过数据并行的方式，将模型的参数和梯度分布到多个设备或节点上

###### 9.nn.ModuleList()和python的list的区别？

nn.ModuleList()是一个储存不同 module，并自动将每个 module 的 parameters 添加到网络之中的容器。可以把任意 nn.Module 的子类 (比如 nn.Conv2d, nn.Linear 之类的) 加到这个 list 里面，方法和 Python 自带的 list 一样, module 的 parameters 也会自动添加到整个网络中。使用 Python 的 list 添加的卷积层和它们的 parameters 并没有自动注册到我们的网络中。网络参数也不会被更新，也就是无法训练。

###### 数组和链表插入和删除的时间复杂度



str.find()：正序字符串查找函数

###### 列表和元组之间的区别是？

答：二者的主要区别是列表是可变的，而元组是不可变的

Python中的字典是一种内置的数据类型，用于存储键值对（key-value pairs）。字典是一种可变容器，

###### 负索引和正索引不同

它是从右边开始检索数组分页。比如将数组[1,2,3,4,9,6,2,2,2,6,7]分为每3个一个子数组即[[1,2,3],[4,9,6],[2,2,6],[2,7]]，要求每个子数组内最多有两个重复数值。在构造下一个数组时，优先使用上次没用上的重复值。用队列

###### Python比C++好在哪里？

内存管理：Python具有自动内存管理和垃圾回收机制，这减少了内存泄漏的风险，并简化了内存管理。

开发速度：Python语法简洁，拥有丰富的库和框架，可以快速开发和原型制作。

社区和库：Python有一个庞大的开发者社区和大量的库。

###### Python 的 with 语句

用于简化资源管理，如文件操作，它确保即使在发生异常的情况下，资源也能被正确地关闭和清理。使用 with 语句，不需要显式地调用 close() 方法来关闭文件，因为它会在代码块执行完成后自动释放资源。

###### 哈希表查找时间复杂度

如果所有元素都映射到同一个哈希桶（哈希冲突），查找可能需要 O(n) 的时间

###### 浅拷贝和深拷贝

创建一个新对象，它是原始对象的一个副本，但它不会递归复制对象内部的元素。对于对象内部的元素，浅拷贝会复制元素的引用，而不是元素本身。这意味着如果对象内部包含可变对象，那么这些内部元素在原始对象和拷贝对象之间是共享的。使用copy模块中的copy()函数。

深拷贝创建一个新对象，并且递归复制对象的所有内部对象。这意味着新对象和原始对象之间不会有任何共享的元素。对于原始对象中的每一个对象，深拷贝都会创建一个新的独立副本。使用copy模块中的deepcopy()函数

###### **New和init**。

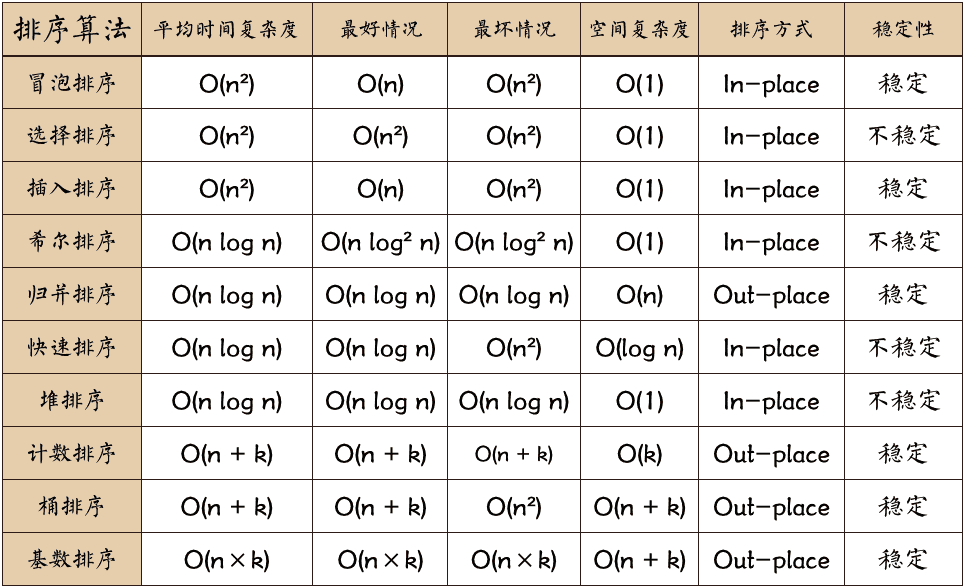
\_\_new\_\_ 是一个静态方法，负责分配内存和返回新对象的实例。它是创建对象的第一步，返回一个对象实例，然后这个实例被传递给\_\_init\_\_方法。

\_\_init\_\_ 是一个实例方法，负责初始化对象的状态。它是创建对象的第二步，通常用于设置对象的属性

###### Python和C语言在多线程方面有几个显著的区别：

1. **线程模型**：
   * **Python**：在多核处理器上，Python的线程也不能实现真正的并行执行
   * **C**：。C语言允许多个线程同时在多核处理器上运行。
2. **易用性**：
   * **Python**：Python提供了一个相对简单易用的线程库（threading模块）
   * **C**：在C语言中，多线程编程通常需要更多的工作量，包括线程的创建、同步和销毁等，这通常涉及到更多的底层细节。
3. **内存管理**：
   * **Python**：Python自动管理内存，包括线程的内存分配和回收。
   * **C**：在C语言中，内存管理需要程序员手动进行，这包括线程堆栈的分配和释放。
4. **可移植性**：
   * **Python**：Python的线程库在不同平台上具有较好的可移植性。
   * **C**：C语言的多线程实现可能依赖于特定的操作系统API，因此在不同平台之间移植时可能需要额外的工作。

在Python 3中，range函数的内存效率得到了提升，因为它现在返回一个迭代器



###### PyTorch和TensorFlow的区别

2.1 计算图的构建方式

PyTorch使用动态计算图，这意味着计算图是根据代码的实际执行过程动态构建的。这种方式使得调试和编写代码更加方便，但也导致了一些性能上的损失。

TensorFlow最初采用的是静态计算图，即需要在构建阶段定义完整的计算图，然后才能执行。这种方式可以进行更多的优化，提高性能，但在调试和开发过程中较为繁琐。

###### Python 生成器了解吗，在训练数据时，直接加载到一个list和用生成器有什么区别?

List：将所有数据加载到一个 list 中会一次性在内存中创建整个数据集的副本

生成器：生成器在每次迭代时只产生一个值，而不是预先计算所有值并存储在内存中。使用生成器时，数据是在迭代过程中按需生成的

List：如果数据集较小，直接加载到 list 可能更快，因为所有的数据都在内存中，访问速度快。

生成器：由于生成器不会一次性加载所有数据，因此可以用于处理大规模数据集，而不必担心内存限制。

List：在 list 中，可以随机访问任何元素，即可以通过索引直接访问特定的数据点。

生成器：生成器是顺序访问的，不能随机访问。一旦迭代完成，必须重新开始。