**《数据结构与算法基础》**

**效率**

计算机结构：由寄存器、缓存cache(有多级缓存)、RAM内存、SSD固态硬盘、HHD机械硬盘组成。CPU在寄存器中计算，但获取数据需要一级级的调用。各类存储器从前往后性能越低但价格越便宜，要在降低价格的同时让处理速度接近寄存器

缓存命中：寄存器从缓存中读取数据，为缓存命中。如果缓存中没有需要的数据，为缓存未命中，要从下一级缓存或者RAM内存中逐级获取

缓存行：缓存从下一级存储器中读取内容时，不会只读取缺少的数据，而是读取一块固定大小的缓存行(比如为64byte)

**数据结构**

**数组：**

数组：线性表数据结构。用一组连续的内存空间来储存相同类型的数据，当内存不够时需要扩容和复制，此时比较耗时。适合查找，不适合插入和删除某个元素。访问速度为O(1)，插入删除速度为O(n)

**链表：**

链表：有单链表、循环链表和双向链表。内存不是连续的，因此扩容方便。每个链表的节点除了记录存储数据外，还要记录下一个数据的地址。不适合查找，适合插入和删除某个元素

单链表：单链表的最后一个节点记录的是一个空地址NULL

循环链表：相当于单链表的最后一个节点记录的是第一个节点的地址

带头链表：在链表拆入和删除时，首尾节点和其他节点的操作代码有区别，如果每次都加入判断会耽误时间，因此直接在开头插入一个内容为NULL的节点，并让结尾也指向它。也是循环链表

双向链表：相比于单链表每个节点还要记录上一个节点的地址，双向链表可以双向遍历。通常比单链表更高效，属于用空间换时间

跳表：链表只记录相邻数据的地址不适合查找，跳表是在链表的基础上加入有间隔的索引，查找时先通过索引缩小查找范围，再回到链表查找。跳表的索引可以有多层，高层索引间隔大，低层索引间隔小，从大到小查找以不断缩小范围。Redis数据库中使用了跳表

**栈与堆：**

栈：先进后出的内存空间。通常局部变量都放在栈中，函数运行结束后被清空，程序内部会自动管理栈内存，栈的内存比较连续，但大小有限，数据太大容易栈溢出

堆：其他内存空间。通常全局变量和不确定大小的数据放在堆中，根据程序员的需要手动创建和删除，堆内存容易碎片化，但堆区容量大，适合存放大数据。比如cpp中的new

栈：操作受限的线性表数据结构。存储和删除数据时按照先进先出的原则，好比堆放盘子。栈需要栈顶指针。有顺序栈和链式栈，出栈和入栈时间复杂度为O(n)

顺序栈：用数组实现的栈。通常顺序栈大小是固定的，也有可以扩容的动态顺序栈，和数组的内存一样扩容和复制时比较耗时

链式栈：用链表实现的栈

队列：与栈类似，是一种操作受限的线性表数据结构。存储和删除数据时按照先进先出的原则。队列需要栈顶和栈尾指针。有顺序队列和链式队列

**树：**

树：像树枝一样由许多节点组成，节点分为根节点、父节点、子节点、兄弟节点、叶节点。节点高度：节点到叶节点的最长路径。节点深度：节点到根节点的路径长度。节点层数：节点深度+1。树的高度：根节点的高度

二叉树：每个节点最多有两个节点(可以只有一个或没有)。数据存储方式有链式存储法和顺序存储法。二叉树的查找、插入、删除的时间复杂度一致，与树的高度有关，如果是平衡二叉树复杂度为O(logn)，如果是极不平衡的二叉树(如一条线)复杂度为O(n)

链式存储法：每个节点有3个字段，1个存储数据，另外两个指向左右子节点

顺序存储法：每个节点占数组中的固定1个位置，0位置空出，剩下的位置从根节点开始，依次填补每层节点，如果实际中没有该节点则填入空，完全二叉树的浪费空间最少

满二叉树：除了最后一层的节点，每个节点都有两个子节点。

完全二叉树：满二叉树去掉最右边的n+1个叶节点。使用顺序存储法时，数组除了0位置都能填满数据

平衡二叉树：任意一个节点的左右子节点树的高度差不大于1。包括但不限于满二叉树和完全二叉树

二叉树查找：前提条件是任意父节点的左子节点更小、右子节点更大。通过判断大小来递归查找。有相同的值时，相同的值放到节点的右子节点，查找到目标值时还要不断查找其右子节点

二叉树插入：先进行二叉树查找。当搜寻到叶节点时，如果值更小插入到左侧；如果值更大插入到右侧。有相同的值时，相同的值放到节点的右子节点

二叉树删除：先进行二叉树查找找到要删除的值。如果是叶节点则直接删除；如果该节点有1个子节点，则去除节点后要用其子节点代替原来的位置；如果该节点有2个子节点，要找到右子节点树中的最小叶节点，用其值替换要删除节点的值，并删除这个最小叶节点

红黑树：接近平衡的二叉树，节点被分为黑和红两类，所有叶节点的值都是NULL(方便代码编写)。根节点、叶节点是黑节点，红节点被黑节点隔开。红黑树是为了防止二叉树在插入和删除的过程中变得不平衡的一种限定，红黑树在进行插入和删除时有特殊规则，从而使二叉树一直接近平衡，保持查找、插入、删除的效率

**其他：**

散列表(哈希表)：输入的键(关键字)经过散列函数(哈希算法)的运算会得到一个固定的下标(散列值)，通过下标快速找到数组中的数据。但由于散列函数的局限性，存在少数不同的键运算后得到相同的地址，即散列冲突，可以用开放寻址法、链表法等方法弥补

python元组：长度固定、元素类型不需相同的特殊数组

python列表：动态数组

python字典：散列表(哈希表)

world单词检查：散列表(哈希表)

**排序**

排序算法：冒泡、插入、选择的时间复杂度O(n\*n)。快排、归并的时间复杂度为O(nlogn)。桶、计数、基数的时间复杂度O(n)。实际使用排序算法时，还要考虑空间复杂度、稳定性、比较次数、交换次数、系数、常数、低阶。各排序算法在实现时也有各种版本。目前常用的是Timsort排序，综合了二分插入排序和归并排序且做了优化

原地排序：指不使用额外的内存空间,空间复杂度O(1)，如冒泡排序、

冒泡排序：每轮遍历依次比较2个数的大小，找出最大值。稳定。原地排序。最大时间复杂度O(n\*(n-1)/2)，最小时间复杂度O(n)，平均时间复杂度O(n\*(n-1)/4)。比较次数n\*(n-1)/2。交换次数为3倍逆序度(赋值操作需要3个变量)

插入排序：从左到右遍历所有值，比较大小后插入到左侧。稳定。原地排序。最大时间复杂度O(n\*(n-1)/2)，最小时间复杂度O(n)，平均时间复杂度O(n\*(n-1)/4)。比较次数n\*(n-1)/2。交换次数为逆序度。因此插入排序比冒泡排序好

选择排序：每轮遍历找出剩余元素的最小值并与剩余的第一个元素交换位置。不稳定。原地排序。最大时间复杂度O(n\*(n-1)/2)，最小时间复杂度O(n\*(n-1)/2)，平均时间复杂度O(n\*(n-1)/2)。比较次数n\*(n-1)/2

归并排序：将元素用递归的方式拆成多份排序，再不断合并排序直到合为一个，合并时需要创建临时空间。稳定。不是原地排序。时间复杂度均为O(nlogn)

快速排序：选取一个的元素作为划分点，将大于和大于的元素分为两组，再用递归的方式不断执行下去。不稳定。原地排序。平均时间复杂度为O(nlogn)

桶排序：前置条件是要提前划分区域。先把数据从小到大划分到多个区间，再分别使用归并/快速排序。比如对庞大的数据进行进行排序时内存可能不够，需要先将其按照大小划分到多个范围，再分别使用归并/快速排序并拼接。不是原地排序。划分为1个桶时间复杂度为O(nlogn)，划分接近n个桶为O(n)

计数排序：前置条件是要提前划分区域。如果排序对象的值的范围可以确定且数据量\*logn大于值的范围跨度(比如高考分数)，可以直接以每个值来划分数据。类似于桶排序，只是桶内部不需要再排序。不是原地排序。时间复杂度为O(n)

基数排序：前置条件是要提前划分区域。是计数排序在值范围比较大时的划分改进，虽然增加了时间复杂度但减少了区域划分数。如果排序对象的值的范围比较大(比如手机号)，可以先用计数排序对个位进行排序，再依次对十位、百位、...k位进行排序。不是原地排序。时间复杂度为O(k\*n)

Timsort排序：对长度小于32/64的数组使用二分插入排序，大于时使用归并排序，同时使用了节约内存和加速合并的方法

**查找**

数据查找：有二分查找、跳表查找、散列表、二叉树查找等。二分查找依赖数组，内存占用最小。跳表查找依赖链表。散列表和二叉树依赖链表。二叉树查找依赖数组

二分查找：前置条件是需要提前排序、是数组结构。 选取数组的中位数与目标值进行比较来缩小范围，然后在剩余数据中重复操作直到找到目标值。时间复杂度O(logn)。如果是链表结构，查找的速度会降低。在大量数据时使用二分查找相比遍历才有明显优势

跳表查找：利用跳表结构查找。时间复杂度O(logn)。此外插入、删除的时间复杂度也为O(logn)

**树模型**

RandomForest、XGBoost、LightGBM。对于数据量少的表格数据，树模型往往比一般的深度学习算法效果好

**深度优先搜索(DFS)**

从树的根部开始，往叶节点搜索，搜索到叶节点就回退，遍历过的节点记录状态不再重复，直到遍历完整棵树。对于不是明显树结构的问题，要当成树结构来操作

**代码：**

地图寻路中有4个部分：终点判断、边界判断、四周搜索、状态回退

def dfs(y, x):

global end, map, h, w, len\_min, len\_now, distance

if (y, x) == end:

len\_min = len\_now if len\_min > len\_now else len\_min

return

if not -1 < y < h or not -1 < x < w or map[y, x]:

return

len\_now += 1

map[y, x] = 1

dfs(y + 1, x)

dfs(y - 1, x)

dfs(y, x + 1)

dfs(y, x - 1)

len\_now -= 1

map[y, x] = 0

**最优性剪枝：**

当路径不可能最优时直接跳过

if len\_now + abs(end[0] - y) + abs(end[1] - x) >= len\_min: # 最优性剪枝

return

**动态规划剪枝：**

利用动态规划的思想，记录每个位置的最短路径，当路径小于/等于历史最佳值时舍弃

if len\_now >= distance[y, x]: # 动态规划剪枝

return

distance[y, x] = len\_now

**启发式剪枝：**

根据经验优先搜索离目标近的方向，但如果遇到U型障碍反而会比较慢，根据经验调整

**广度优先搜索(BFS)**

从树的根部开始，先遍历子节点，再遍历子节点的子节点，遍历过的节点记录状态不再重复，直到遍历完整棵树

**代码：**

def bfs(y, x):

global end, map, h, w, len\_min

map[y, x] = 1

history = {(y, x): 0}

step\_list = [(y, x)]

if (y, x) == end:

len\_min = 0

return

while step\_list:

y, x = step\_list.pop(0)

for y1, x1 in [(y + 1, x), (y - 1, x), (y, x + 1), (y, x - 1)]:

if (y1, x1) == end:

len\_min = history[(y, x)] + 1

return

if not -1 < y1 < h or not -1 < x1 < w or map[y1, x1]:

continue

map[y1, x1] = 1

history[(y1, x1)] = history[(y, x)] + 1

step\_list.append((y1, x1))

**最佳优先搜索(Best\_FS)**

在广度优先的基础上优先对离目标近的点搜索。在BFS基础上用优先队列替换原列表，列表中添加当前点到目标点的距离作为限制。python中可以使用

**A\*搜索**

在最佳优先的基础上添加当前点到原点的距离作为限制

**寻路**

寻路可视化网站：<https://qiao.github.io/PathFinding.js/visual/>

**距离**

欧氏距离(Euclidean)：两点之间的直线距离

曼哈顿距离(Manhattan)：只能上下左右走网格的距离，适用于城市中两地距离

切比雪夫距离(Chebyshev)：可以上下左右和斜着走网格的距离

octile距离：只保留障碍寻路图中的关键点，探索关键点之间的连接方式。在预先知道地图的情况下，将地图关键区域处理为关键点可以大幅提高寻路效率

**《图像基础》**

红绿蓝(RGB)通道：最常用的颜色标准，一幅图由三种颜色叠加而成(0-255)，人眼所能看到的颜色，都可以通过调整红绿蓝三种颜色的比例无限接近。jpg图片为RGB格式

RGBA通道：相比RGB通道，多出的A通道(alpha)代表透明度，A通道的值为0-255，0时为透明。png图片为RGBA格式

色相、饱和度、亮度(HSV)通道：色相主要控制色彩的变化(0°-360°)，饱和度控制颜色的纯度(0-255)，亮度控制图片的亮度(0-255)。三通道互有影响

图片格式：在一张1920\*1080(1920,1080)的图片中，1920指的是宽w、1080指的是高h。RGB图片数据为(1080,1920,3)(h,w,3)，RGBA图片数据为(1080,1920,4)(h,w,4)

图像特征：形状、颜色、纹理

语义鸿沟：相同视觉特性不同语义概念，相同语义概念不同视觉特性

邻域平均法：，邻域平均法去高斯噪声较好，缺点是造成图像模糊，可以设置阈值检测异常灰度值改进。优化算法有：K近邻平均法、梯度导数加权平滑、最大均匀性平滑、小斜面平滑等。

均值滤波：。

高斯滤波：，更好保留轮廓，比均值滤波好

中值滤波法：选取方框内的中位数。非线性，适合滤除脉冲干扰和图像扫描噪声，不适合细节多的图像，去椒盐噪声好。

Sobel算子：和

Scharr算子：和，相比Sobel算子加强了轮廓

Laplacian算子：，不单独用

Canny边缘检测：高斯滤波去噪；计算每个像素点的梯度和方向；非极大值抑制消除散杂响应；双阈值检测；抑制孤立弱边缘。

图像金字塔：高斯金字塔、拉普拉斯金字塔

模板匹配：待匹配的区域在原图中从左到右从上到下进行平方差/相关性/相关系数的计算，选出最接近的区别。模板匹配是最简单的目标检测、计算效率高、可以检测多个相似对象，但只适合静态图与模板匹配。

傅里叶变换作用：高频：变化剧烈的灰度分量，如边界，高通滤波器保留高频分量只保留边界；低频：变化平缓的灰度分量，如大海低通滤波器保留低频分量使图像模糊

**《神经网络》**

静态图和动态图：静态图先定义好网络再计算，速度快；动态图边计算边构建网络，计算速度较慢但便于调试

单层神经网络/感知机：线性分类，凸函数，在前馈型神经网络中，只要有一个隐含层并且神经元足够多，就能近似任何的连续函数或空间分布

多层神经网络：非线性分类，非凸函数，多隐含层可以表示非连续的函数或空间分布、减少每层神经元数量和泛化误差

感受野：每个神经元所能感受到的数据范围，使用卷积提高神经元感受野。如经过一次的卷积感受野就从1提高到了9，因此感受越高理论上越能判断物体的全貌。但在目标检测中，会出现预测的框比感受野大的情况，一种解释是网络可以通过局部预测全貌

特征提取层：对神经网络模块的泛称，特征提取是一个模糊的概念，可抽象理解为网络在卷积和参数更新的过程中，不断把重要的参数权重分配给能区分物体的部分，使要检测的部位通过训练成为一个’感兴趣’区域

上采样：插值法、反池化、反卷积

下采样：缩小图像。当卷积时的步长大于1时也是一种下采样，常用步长为2的卷积代替池化层

循环神经网络(Recurrent Neural Networks)(1990)：可以处理任意长度的输入序列，可以对序列中的任意位置进行操作；训练过程中容易出现梯度消失或梯度爆炸问题，难以处理长期依赖关系

损失函数(Loss)：衡量输出和标签值的误差，每次训练中根据一个批量数据的平均损失更新参数。在网络的训练中损失函数应该下降一个以上数量级

误差反向传播(Back Propagation)：利用链式法则，计算出损失函数对所有参数的梯度，根据梯度更新参数。在特征提取网络中，损失函数(任务)可能有多个，此时参数的更新方向取决于所有损失函数，只有特征提取网络后输出层的参数才取决于对应的损失函数。通常各个损失之间会存在对抗，可在特征提取网络后单独对每个任务进行卷积加强区分

**linear(FC)**

全连接层：分别建立每一个输出和每一个输入的线性关系，具有强大的拟合能力，但容易过拟合。全连接层的输入为一维向量，输入长度，输出长度时，模型参数为，1为常数项。通常单纯的叠加全连接层不会进一步提高模型性能，反而容易过拟合。pytorch中函数为torch.nn.Linear，通常网络中间的linear不加入偏置项

多层感知器(Multi-Layer Perceptron)：由多层全连接层组成的网络

**Dropout**

丢弃层：降低训练集准确率，提高泛化性，提高测试准确率，通常取0.2左右。丢弃层通常会搭配linear一起使用，但在很多大模型中不加入丢弃层

**BatchNorm**

批量归一化：输入(batch,dim,feature...)后会在dim维度进行归一化。通常加在卷积网络和激活函数中间，防止梯度消失有助于收敛。多GPU训练时可以用SyncBatchNorm统计所有卡的批量。train模式会根据输入数据动态归一化，eval模式会使用固定的值。训练数据太少时归一化也会过拟合，train和eval的区别会很大，会出现train很准但eval很差。训练批量减少为1可以改善归一化的问题

其中和为可训练参数，为归一化的逆操作。总体相当于一个可训练的动态归一化，在防止梯度消失和保持更多非线性之间权衡。代表了该网络层的重要程度，如果模型训练后的某个的值很小，则该层在后续的计算中影响程度很小，因此也作为模型剪枝的参考标准

**LayerNorm**

层归一化：与BatchNorm相比的区别是对每个批量单独进行归一化，输入(batch,dim...,feature)后有两种归一化方式，可以对每个batch进行归一化，或者对每个batch的每个feature分别进行归一化

**RMSNorm(2019)**

均方差层归一化，是层归一化LayerNorm的改进版，取消了偏置项，效果与LayerNorm差不多，但能加快收敛、节省训练时间。llama3等大模型中使用RMSNorm替代LayerNorm

**Pool**

池化层：下采样的一种，减少参数，防止过拟合，一般用最大池化层。

最大池化层(MaxPool)：目标偏向纹理轮廓，网络浅层使用最大池化过滤无用信息。

平均池化层(AdaptiveAvgPool)：目标偏向背景等，网络深层使用平均池化层防止丢失过多信息

**CNN**

由卷积层组成的网络

卷积层(Conv)：由多个卷积核组成，作用为提取关键信息。提高卷积层中卷积核的数量可提高训练准确率，但测试准确率有瓶颈，后1卷积层的卷积核个数一般大于/等于前1层

卷积：形状为n\*n的卷积核在图片上滑动进行互相关/卷积运算会得到一组二维数据a\*a，n为奇数一般为3。

卷积核：卷积核的参数channel\_2\*channel\_1\*n\*n，其中channel\_1为前1层通道数，channel\_2为结果的通道数。首先每个n\*n分别在前1层通道中运算得到channel\_2\*channel\_1\*a\*a，然后在1维相加得到channel\_2\*n\*n，再加上channel\_2个偏置项，因此总的参数为channel\_2\*(channel\_1\*n\*n+1)。通常所说的卷积核为指的只是n\*n，比如1\*1卷积核用于升降维、减少参数量或加在3\*3卷积前后缓冲。相同参数下，多用3\*3卷积核比大卷积核有更多感受野。torch.nn.Conv2d中，要提前定义输入维度dim、输出维度、n、滑动步长等信息，运算时传入张量

膨胀卷积(Dilated Convolution)：卷积核中隔行参数参与计算，相同计算量下膨胀卷积的卷积核更大，感受野更广

因果卷积(Causal Convolution)：在一维数据中为了使卷积前后长度不变要在数据前后填充，因果卷积全部填充在前面

**LSTM(1997)**

长短期记忆网络：加强对序列近期位置的记忆，减少对历史久远位置的记忆。内部有输入们、遗忘门、输出门来实现记忆功能

**Transformer(2017)**

注意力框架：<https://blog.csdn.net/zhaohongfei_358/article/details/126019181>

标准transformer：由编码器和解码器组成。在torch.nn.Transformer函数中为了灵活将词向量编码、位置编码、输出解码部分单独取出。实际中应用中，语言生成模型(GPT)只使用其解码器，多模态文本模型只使用其编码器

词向量编码(embedding)：编码器和解码器的第1层，编解码器的词向量编码层是共用的。通过特定的算法将词索引向量(batch,n)变为词向量(batch,n,feature)，并且能考虑到词与词之间的距离关系。词索引向量的长度n可以是任意的，但索引值要在词表总数的范围内，词向量长度dim是固定的，词向量编码层的参数是可训练的。计算方式：x=词向量编码(x)

位置编码(Positional Encoding)：编码器和解码器的第2层，编解码器的位置编码层是共用的，记录词索引的位置信息。注意力机制相比于RNN、LSTM等网络不依赖词的位置，所有词同时进行运算，但也是去了位置信息，因此需要加入位置编码。标准transformer框架中只是在词向量编码后进行绝对位置编码，还有加偏置项版本、相对位置编码、旋转位置编码等，很多大语言模型会把位置编码放到每个注意力层中。计算方式：x=x+位置编码矩阵

绝对位置编码：为每个位置分配一个唯一的向量，但没有捕捉相对位置信息。形状如(dim,feature)，计算方式：x=x+位置编码矩阵。位置编码矩阵如下：

词向量解码：解码器的最后1层，可为全连接层。若输入词向量(batch,n,feature)，词表长度m，则经过全连接层后为(batch,n,m)，经过argmax后得到词索引向量(batch,n)。如果是语言生成模型，不能直接使用argmax，而是要根据数据的概率分布来选值。如果是在训练，不需要argmax，而是变形为(batch\*n,m)与标签进行交叉熵损失。使用torch.nn.CrossEntropyLoss时标签可以直接传入词索引向量，不用变成独热编码

编码器(encoder)：对输入的词索引向量进行运算，只需运算一次。输入：(batch,n)，输出：(batch,n,feature)。构成：词向量编码、位置编码、6个(可变)编码模块、归一化层

编码模块：自注意力层+2全连接层+2归一化层+2丢弃层+1激活函数。计算方式：x=LayerNorm(x+Dropout(自注意力层(x)))，x=LayerNorm(x+Linear(Dropout(gelu(Linear(x)))))

解码器(decoder)：对输入的词索引向量进行运算，会进行多轮运算，将之前生成的结果合并输入到下一轮的预测中。输入：词索引向量(batch,n)、编码器输出(batch,n,feature)，输出：(batch,n,feature)。预测时只取(batch,-1,feature)作为本轮输出，然后会与之前的结果合并再输入到下一轮中，训练时有掩码矩阵因此可以批量计算损失。构成：词向量编码、位置编码、6个(可变)解码模块、归一化层、词向量解码

解码模块：自注意力层+编解码注意力层+2全连接层+3归一化层+3丢弃层+1激活函数。计算方式：x=LayerNorm(x+Dropout(自注意力层(x)))，x=LayerNorm(x+Dropout(编解码注意力层(x)))，x=LayerNorm(x+Linear(Dropout(gelu(Linear(x)))))

自注意力层：多头注意力层，输入的查询向量、键向量、值向量为同一个

编解码注意力层：多头注意力层，输入的查询向量为解码器的输入，键向量、值向量为编码器的输出

**Attention**

注意力层：建立每个词之间的联系，4全连接层+1归一化层+1丢弃层。

输入为查询向量(batch,dim,feature)、键向量(batch,dim,feature)、值向量(batch,dim,feature)，分别经过3个linear层，形状保持不变。每个词的查询向量(feature,)与所有词的键向量(dim,feature)逐一点乘得到(dim,)，表示该词对所有词的注意力得分，所有词就是(dim,dim)，所有批量就是(batch,dim,dim)。解码器中为了批量训练，注意力得分要加上掩码矩阵。得分除以len(feature)的开方后进行softmax归一化，再进行dropout。将得分(dim,dim)与值向量(dim,feature)相乘得到结果(dim,feature)，所有批量就是(batch,dim,feature)。最后再经过linear层得到(batch,dim,feature)。快速计算：torch.matmul(查询向量,键向量.permute(0,2,1))和torch.matmul(得分,值向量)。实际应用中键向量、值向量的dim2可以与查询向量的dim不同，最后结果仍为(batch,dim,feature)

掩码矩阵：模型训练时为了批量计算需要加入解码器的掩码矩阵，编码器通常不需要加入。模型预测时不需要掩码矩阵。广义上的掩码矩阵已经与掩码向量融合、包含了填充符掩码，没有填充符的原始掩码矩阵的左下边和对角线为0，右上边为-torch.inf，有填充符时，填充符对应的列都设为-torch.inf，之所以用-inf是因为经过softmax时会得到0。掩码矩阵会与注意力得分直接相加

掩码向量：形如[0,0...-inf,-inf]，-inf对应填充符位置。在torch.nn.Transformer函数中，需要传入原始掩码矩阵和掩码向量，函数中会对它们进行融合变成最终的掩码矩阵

**MultiheadAttention**

多头注意力层：注意力层的改进版本，基本取代了原始的注意力层。

改进点：输入的查询向量、键向量、值向量在分别经过3个linear层后，会从(batch,dim,feature)拆分为(batch,head,dim,feature//head),归一化时得分除以len(feature//head)的开方后进行softmax，得到结果(batch,head,dim,feature//head)后先合并为(batch,dim,feature)，再经过最后的linear层，其余的计算一致。head可以为8

**GroupQueryAttention(2019)**

多头查询注意力层：多头注意力层的改进版本，在基本不影响性能的同时减少了参数量，应用于大模型llama3中

改进点：查询向量(batch,dim,feature)经过linear后形状不变，然后拆分为(batch,head,dim,feature//head)。但键向量、值向量经过linear后形状压缩变为(batch,dim,feature//group)，再拆分为(batch,head//group,dim,feature//head)，复制group份后合并为(batch,head,dim,feature//head)。相关于减少了2个linear层的参数，llama3等大模型中group为8

**MultiheadLatentAttention(2025)**

多头潜在注意力层：多头注意力层的改进版本，在基本不影响性能的同时减少了参数量，应用于大模型deepseek中

改进点：在特征传入MultiheadAttention前先经过1个linear层降维，从MultiheadAttention输出后再经过1个linear层升维

**Vision Transformer(2020)**

视觉注意力机制：将Transformer对图片进行特征提取。具体是先用大尺度卷积大幅减少尺寸的同时大幅提高维度，再将尺寸的两个维度展平为一个维度，然后送入到Transformer中

**GNN**

图神经网络的统称，包含GCN等网络

**GCN(2013)**

图卷积网络：由图卷积层组成的网络，虽然叫图卷积，但和卷积网络无关。与CV和NLP的区别为数据格式不固定，数据之间有特殊关系，并非所有的点都有标签，计算损失时也只计算有标签的点(因此经常会是半监督任务)。由于GCN巧妙的设计，使用未训练的初始化参数就能初步的分类。由于GCN每次的输入是所有图数据，当图改变时要重新训练，图太大时显存会不够

图卷积层(Graph Conv)：N0 \* A \* N1 \* B \* C。N0和N1为归一化矩阵，A为邻接矩阵，B为特征矩阵，C为权重矩阵。之所以前后都乘N，是因为要对行和列都做归一化(假如A只认识B一人，而B认识很多人，经过归一化后A受B的影响权重会很小)。随着图卷积层数的增加，每个点的参数更新就会受到更多点的影响，感受野增大，但通常2、3层时效果就达到最好，过深的层数反而容易导致效果变差(可以理解为感受野增大减少了个体的差异)。torch\_geometric.nn.GCNConv中，要提前定义权重矩阵C的形状，运算时传入特征矩阵B和邻接关系(2,n)(不直接传入邻接矩阵A)

邻接矩阵：由0和1组成的矩阵，表示点与点之间的关系，有关系为1，无关系为0，自己与自己的关系也为1。比如有5个物体，邻接矩阵的尺寸为25\*25

特征矩阵：特征矩阵(a,b)的0维长度等于节点数，1维代表特征长度。首次传入的特征矩阵为各节点的初始特征，在没提供特征的简单情况下可为(a,a)：对角线为1，其他位置为0。之后的特征矩阵为之前网络运算的结果

邻接关系(2,n)：形如([[0,0,1...],[1,2,2...]])，前后两列不能互换位置，比如0认识1但1不一定认识0，0与0本身默认有关系不用列出

损失计算：只计算有标签部分的损失(pred[mask]和true[mask])

**TGCN**

时序图卷积网络：带有时间信息的图卷积网络。先将每一个时刻的图进行GCN运算，然后用类似RNN的网络组成时序网络

**GraphSAGE(2017)**

图采样聚合。GraphSAGE相比GCN不直接把所有数据进行训练，而是用采样的方式拆分成多组数据，也可以叠加较多的层数

聚合：对相邻节点做平均、池化、LSTM等运算，融合邻居关系。矩阵相乘也算一种聚合

**PNN**

概率神经网络：基于贝叶斯决策规则的神经网络技术，其神经网络的训练期望误差较小，是一种基于统计原理的人工神经网络。擅长识别与分类任务

**FNO**

傅里叶神经算子：能够学习无限维函数空间映射的深度学习架构。通过学习从任意函数参数到解的映射来解决一类偏微分方程的求解问题

**《激活函数》**

激活函数(Activation)：不加激活函数时无论有多少层隐含层都可以化简为一层线性变换，激活函数应该是简单的非线性函数、连续可导、单调函数

**sigmoid**

函数：torch.nn.Sigmoid()

对数几率函数。将数值压缩在0-1之间。常用于二分类和精确的归一化



**softmax**

函数：torch.nn.Softmax(dim)

将数值压缩在0-1之间，值可以理解为概率分布。、为第i、j个节点的输出值。常用于多分类和不精确的归一化

**tanh**

函数：torch.nn.Tanh()



**relu**

函数：torch.nn.ReLU()



**leaky relu**

函数：torch.nn.LeakyReLu()



**softplus**



**mish**



**silu**

函数：torch.nn.SiLU()

最早在强化学习中提出，具有自稳定性，隐性正则效果。目前大多数模型都使用silu函数，包括大语言模型的mlp网络



**gelu**

函数：torch.nn.GELU()

gelu的效果与silu差不多，但计算速度略慢



**《损失函数》**

**MAE**

平均绝对误差。函数：torch.nn.L1Loss()

**MSE**

平均平方误差。函数：torch.nn.MSELoss()

平方损失函数对w和b求导时要对求导，值可能非常小，导致更新速度缓慢，逻辑回归中可能有多个极小值点

**MAPE**

平均绝对占比误差。在平均绝对误差MAE的基础上增加了分母，除以原真实值得到误差比例，进行归一化

**Crossentropy**

平均交叉熵损失。函数：torch.nn.BCEWithLogitsLoss、torch.nn.CrossEntropyLoss

平均交叉熵损失函数对w和b求导时不需要对求导

**IOU(Intersection over Union)**

a为真实框，b为预测框，(,)为左上坐标，(,)为右下坐标。w、h为宽和高。c为极小值防止union为0(特殊情况如边框错误等)，由于计算时+c，也+c

IoU问题：当IoU=0时梯度为零；收敛较慢、回归不够准确

**GIOU(2019)**

其中G为两框的最小外接框面积

GIoU解决了梯度为零的问题，效果略比IoU好，但当预测框和真实框水平或垂直时，退化成了IoU，依然收敛较慢、回归不够准确

**DIOU**

其中为两框的中心点距离，为最小外接框左上到右下的对角线距离

DIoU收敛速度和准确率都有所提高，但当预测框和真实框接近时退化为IoU

**CIOU**

为极小值防止分母为0，其中当作常数，因此计算损失时关闭自动求导

CIoU中v相对于w和h的梯度：

当w和h在时容易导致梯度爆炸，因此实际中常用1代替。但YOLOV5中由于会对w,h进行筛选从而防止梯度爆炸，因此不需要调整

**正则化**

在损失函数中加入正则化可以让模型的参数尽量变小，但一般不显示出来

L1正则化：函数是线性的(V字)，会让模型中部分不重要的参数接近于0，简化了模型结构，缓解了过拟合，权重系数可为0.0001。模型剪枝时可以根据L1正则化的结果去除不必要的结构。L1正则化一般加在批归一化层的梯度中

L2正则化：函数是曲线的(U字)，会让模型中所有的参数减小但不至于接近于0，减少某些较大参数对模型的影响，防止了过拟合，权重系数可为0.0001。模型剪枝时也可以根据L2正则化的结果去除不必要的结构。L2正则化在函数中有封装

**《CV》**

**数据处理**

[图片]-->[数据增强]-->[归一化]-->[合成批量(batch,size,size,3)]-->[模型]

归一化：除以255、减均值、除以标准差、等比缩放图片和填充灰边(128)，由于图片各通道标准差差不多，可以直接除以255。在有的模型中可以直接缩放图片

数据增强：旋转、仿射变换、翻转、缩放、平移、对比度变换、噪声扰动

马赛克增强(mosaic)(2020)：将四张图片随机进行大小调整、翻转、裁剪等后合在一起，同时标签也做对应的调整。优点：增加了训练的速度；缩小了图像以提高对小目标的检测能力；排除检测目标的背景干扰。但由于增强后的图片与真实场景不一致，因此最后几轮的训练通常不使用马赛克增强

**指标**

TP：存在真实框，置信度大于阈值，iou大于阈值，类别正确

FN：存在真实框，置信度、iou、类别任意不满足

TN：不存在真实框，置信度小于阈值

FP：不存在真实框，置信度大于阈值

**图像分类**

输入固定大小的图片，得到图片的类别

[预处理] --> [模型] --> [(batch,dim,n,n)] --> [平均池化(batch,dim,1,1)] --> [(batch,dim)] --> [(batch,类别数)]

**目标检测**

输入固定大小的图片，得到图片中目标物体的边框

[预处理] --> [模型] --> [(batch,dim,大层长度n,大层长度n)\*大层数3] --> [(batch,小层数3,n,n,内容m)\*3] --> [内容m：(Cx,Cy,w,h,置信度,类别独热编码)] --> [归一化] --> [边框解码] --> [置信度筛选] --> [非极大值抑制]

**目标检测解码**

YOLO输出的特征图后需要归一化(sigmoid)，然后人为转换成，并赋予其意义：3为小层数、(80,80)为图像的位置、4为物体的边框(cx,cy,w,h)、1为置信度、80为物体的种类

如YOLOV5的解码方式为：

先验框(anchor)：固定比例或由大数据kmeans聚类得到，有w和h组成，YOLOV5中每个输出中有3个通道，分别对应3个先验框，使每个预测可用多种先验框匹配。总共有3个输出、9个先验框

**目标检测损失**

a+b+c=1。输出特征图较大时置信度的损失权重要大些，让网络更专注于识别物体的位置。一般置信度和类别损失用交叉熵损失，边框损失用IoU系列。输出所有位置的置信度都计算损失；但边框和类别只有有标签对应的位置计算损失

边框、置信度、类别损失之间会有一点冲突，在特征提取网络之后可以加入几层卷积网络对各任务分别计算

一个的输出有个预测位置，只有部分位置的预测有用、参与损失计算、有标签矩阵对应，可以利用判断矩阵筛选

在计算置信度损失时，由于大多数地方都不需要预测为0，因此需要提高正样本的权重

**聚焦损失(Focal Loss)**

当训练集中正样本很少时，即使模型全预测为负样本准确率也会比较高，因此需要加大正样本预测错时的损失

**判断矩阵**

输出的标签矩阵为，判断矩阵为，pred\_matrix[judge\_matrix]取出需要计算损失的位置，label\_matrix[judge\_matrix]取出对应位置的标签

**标签矩阵**

标签矩阵的形状和网络输出一致，在制作标签矩阵时，需要预测的位置填入对应标签，不需要的位置label\_matrix[judge\_matrix]会自动过滤，填入的标签需要满足一定条件，因为网络输出解码后的范围要包含这个标签的值才能预测。有的网络如YOLOV5扩充了标签中有物体的位置，物体可以由四周最近的两个网格预测

**wh筛选条件**

若解码方式为：

为网络输出，为真是输出。因此标签筛选范围为：

**其他**

标签平滑(Label Smoothing)：类别标签一般为0或1，但由于sigmoid等归一化函数会让值无法达到0和1、模型的原始输出通常在+-5之间，因此标签值改为0.01和0.99

建议框(proposal)：根据置信度筛选后还未进行非极大值抑制的框

非极大值抑制(nms)：将某一种类别的建议框按置信度筛选后，按置信度排序，选出置信度最高的框依次遍历计算与其他框的iou，根据iou阈值去掉重叠的框，记录选出的框，再用置信度次高的框接着遍历剩下的框，直到列表为空

衡量指标：准确率、精确率、召回率、AP

正负样本：一般空标签图片取10%-20%

目标检测指标：目标检测一般采用非极大值抑制后的，因此没有TN和准确率

**人脸识别**

用人脸图片提取特征建立人脸特征数据库、目标检测识别人脸、根据人脸关键点提取特征、在人脸特征数据库中比对

与图片库中比对：可以使用np.dot(a=(),b=())，其中a为要比对的人脸特征，b为库中的所有人脸特征，k为特征数量，n为库中所有人脸数

人脸关键点定位：一般用5个或68个关键点定位

**OCR识别**

OCR是对图片中的文本进行识别。通常需要三步：文字区域检测、文字方向检测、文字内容识别。还可以加上文本图片矫正等修正模型

文字区域检测：图片分割，输出为0-1的掩码矩阵

**行人重识别**

先用目标检测检测出行人的截图，然后进行相似度匹配

rank1：与目标匹配相似度最高的图片正确的概率

map：所有行人的ap平均值

ap：假设n个匹配结果中第1、3、4、...张图片正确，则ap=，表示第三张图片的位置本应该在第二个位置

**《NLP》**

**基础知识**

词：一个词由一个字符或多个字符组成，每个词对应词表中的一个词索引(token)

词索引(token)：词对应的整数索引，从0开始，有多少词就有多少词索引

词表(vocab)：写入所有模型支持的词，词索引=词的序号-1。一般词索引0是填充符、1是开始符、2是结束符，此外还有特殊符号、单词、汉字、表情等。支持中英文和特殊符号的大语言模型词表长度：llama2为5万多、baichuan2为12万多、qwen为15万多

词索引向量(tokens)：由词索引组成的向量(n,)，通常要补齐长度合成批量(batch,n)

词索引编码(tokenizer)：根据词表的对应关系将字符串变为词索引向量，会在前后加上开始、结束标志(1和2)，合成批量时会在短的句子后面填充0

词索引解码：词索引编码的逆过程，将每个词索引变回对应的字符

空白填充(pad\_token\_id)：为了使输入长度一样以合成批量，需要对短的向量填充0

开始标记(bos\_token\_id)：词索引向量的开始标记，通常为1

结束标记(eos\_token\_id)：词索引向量的结束标记，通常为2

自回归(Autoregressive)：解码器将本轮的输入与输出合并，作为下一轮的输入

相对位置编码：增强模型对位置关系的理解，但网络结构较为复杂

旋转位置编码(Rotary Position Embedding)(2023)：旋转位置编码RoPE综合了绝对位置编码和相对位置编码，在处理超长文本时有显著提升，被llama3等大模型采用，与标准transformer不同的是大模型是将位置编码放到每个注意力层中。GroupQueryAttention中的查询向量、键向量分别经过linear后拆分为(batch,head,dim,feature//head)和(batch,head//group,dim,feature//head)，旋转位置编码的sin和cos矩阵形状为(dim,feature//head)。计算公式如下：

翻译任务：使用编码器和解码器，每一轮都要将编码器的结果输入到解码器的中部。解码器的第一轮输入<bos>生成第一个词索引，第二轮输入[<bos>+生成的词索引]得到新的词索引，之后不断重复直到最后一轮生成<eos>停止。生成长度是输入长度-1

翻译模型训练：模型预测时需要获取之前的预测结果，所以只能一个字一个字的生成，而训练时由于有标签，可以批量输入。比如输入AB得到C，输入ABC得到D。假设编码器输入=(batch,n)，解码器则输入=(batch,n-1)(不需要最后的结束符)，解码器输出=(batch,n-1)，标签为(n-1)(不需要前面的开始符)。由于批量长度是固定的，因此在代码中为了批量训练还需要配套掩码矩阵

语言生成任务(GPT)：由transformer的解码器改编而来，解码器由很多解码模块堆叠而成，比如llama3的解码模块为：归一化+注意力层+残差+归一化+mlp网络+残差。语言模型的基础训练是输入一句话来预测下一个字，当模型生成能力达标后需要用指令(instruction)数据训练才能达到对话的效果，具体是在输入中加入特殊的提示(prompt)模版(template)。模型在词向量解码为词索引时，不能直接使用argmax，而是要根据数据的概率分布来选值

mlp网络：这一类网络包含一个门控机制，比如llama3等大模型中mlp由3个linear和1个silu函数组成，输入(batch,dim,feature)后复制成两份，一份经过linear0和silu，另一份经过linear1，然后将两者结果相乘得到(batch,dim,n\*feature)，再经过linear2得到(batch,dim,feature)。linear0和silu实现了一个门控单元，起到筛选信息的作用

语言模型解码(generation)：会对模型的输出进行一些调整操作

提示学习(prompt learning)：监督微调、指令微调，将人类的对话套入到提示模版中，让模型更好理解，比如sft数据格式

投机采样(Speculative Decoding)：2022年提出的大模型推理加速方法。它可以在不损失生成效果前提下，获得3倍以上的加速。GPT-4的OpenAI线上模型推理使用了投机采样

忽略符(ignore\_index)：标签中不需要计算的地方要变为-100，因为交叉熵损失CrossEntropyLoss不会对值为-100的部分计算。通常输入和填充对应的部分要变为-100，部分模型在特殊字符对应的地方也会变为-100

peft模型训练(lora)：peft模型相当于在原模型中插入一个小模型，在尽量不影响原模型参数的基础上调整模型，同时可以大幅节约训练资源。对于效果很好的预训练模型，peft方法不弱于全量参数训练

混合专家模型(MOE)：由多个子模型与一个门控网络组成。比如将多个子模型的结果(相同大小)与门控网络结果(等于子模型数量)相乘后再相加，门控网络学习的是对于不同的输入让输出结果更偏向于哪个子模型

思维链CoT(Chain of Thought)：2022年谷歌论文提出让大模型通过逐步拆解、推理问题的方式解答，可以显著提升模型的性能，推理的中间过程称为思维链。在2024年的很多开源模型中都使用思维链的方式训练，可以选择开启或关闭。注意思维链和深度思考(R1)是两种方案，思维链是在指令微调的数据中加入人工标注的推理步骤，扩充模型的回答标签，让模型学习推理数据；深度思考(R1)是模型通过内部网络隐式的循环迭代、不断推理，训练时不需要推理数据

智能助手(agent)：由大模型、知识库、多种小模型和算法组成的复杂系统。比如用户输入问题后，会经过关键词匹配、或文本分类、或大模型分析用户的意图，然后根据用户的意图调用不同的模型或工具，最后整合到大模型输出结果。可以用langchain做功能集成框架

意图识别：可以使用facebook开发的文本分类库fasttext，有关键词匹配和模型训练功能。关键词匹配的速度非常快，训练模型需要1万以上的数据，可以使用开源模型。通常输入fasttext前，先将段落分为句子，使用分词模型(jieba、HanLP)，然后用空格连起来再输入fasttext

**对话模型**

[通用数据自监督训练] --> [通用提示数据有监督训练] --> [预训练模型] --> [专业数据自监督增量训练] --> [提示数据有监督训练] --> [奖励反馈强化学习]/[直接偏好优化] --> [知识库检索] --> [实际应用的模型]

自监督训练：数据为一段话，数据的前半段作为模型的输入，后半段作为预测的标签，任意文章和段落中的一段话都可以作为数据进行训练，标签不需要提前制作

有监督训练：需要提前制作好数据和标签，让模型一一对应的进行学习

预训练模型：有文字理解能力的基础语言模型，通过在庞大的文章数据上自监督学习以获得通用文字理解能力，这一步是最耗时的

prompt：提示/提示词，严格意义上的prompt应该是经过tokenizer编码前的完整字符串，由系统提示词、输入、提示模版共同组成。但通常说的prompt可能指的是系统提示词

system：系统提示词(system prompt)

提示数据有监督微调：即提示训练、指令训练，包括单轮对话训练和多轮对话训练

单轮对话训练：输入数据为[提示词+问题+提示模版]，输出数据为问题的答案

多轮对话训练(上下文训练)：输入数据为[上轮的输入+上轮回答+此轮问题+多轮模版]，输出数据为此轮问题的答案

单轮对话：一问一答的方式，模型不会考虑之前的对话内容

多轮问答(上下文对话)：模型在回答当前问题时会考虑之前的内容

对话模型的输出中小标题可能会带上“\*\*”号，这是markdown格式的加粗，因为训练时的数据中就带有\*\*号。通常回答中的换行处为\n\n，偶尔会出现\n \n或\n等，除此之外多个空格、小标题格式等都需要注意，因此模型的输出还需进行一定的处理

提示词注入：提示词中通常有开始符和结束符等，如果用户在问题中输入这些内容，可能会干扰模型的理解。如果用户输入一些规则，可能让模型回答违规内容

奖励反馈强化学习(RLHF)：先训练出一个奖励模型(reward model)，然后用策略梯度强化学习(PGRL)和近端策略优化(PPO)的方式微调原模型

直接偏好优化(DPO)：基本替代了RLHF方法，情绪控制上超越了RLHF。DPO在模型得到优化的同时尽量保持输出分布与原模型一致。DPO先将对话的答案分为好、坏2份，模型分为训练模型、参考模型(可为原模型的复制)2个，训练时传入问题到2个模型中得到2个输出，用2个输出和两份答案分别计算出A输出好概率、A输出坏概率、B输出好概率、B输出坏概率，用4个概率来计算kl散度损失并更新训练模型

**知识库检索**

检索增强生成RAG(Retrieval Augmented Generation)(2019)

介绍：<https://zhuanlan.zhihu.com/p/675509396>

[问题] -- > [编码为查询向量] --> [匹配文本块向量] --> [阈值筛选] --> [文本块] -- > [将文本块与问题一起输入模型]

当用户提出问题时，会先用问题到数据库检索相关信息，然后将相关信息作为背景和问题一起输入模型。可以借助llama\_index、ollama、langchain库实现。

以qwen的提示词为例：<|im\_start|>system\n{system}\n你的回答要参考以下资料：\n{text}<|im\_end|>\n<|im\_start|>user\n{input}<|im\_end|>\n<|im\_start|>assistant\n

文本向量制作：将文档划分为块分开存储，然后将文本块编码为向量。文本块的大小要考虑文本块长度对模型理解的影响，属于提示词工程。可以借助llama\_index库实现

**数据处理**

[字符串] --> [词索引向量(tokens)] --> [填充0合成批量(长度不定)] --> [词向量(长度固定)] --> [模型]

词向量编码(embedding)也可算作模型的一部分

通常用openaiAPI制作的高质量对话数据的回答字数在700-900，如果按照1token=1.4汉字计算，加上问题和提示模版，训练数据长度约500-700

详细参考python.docx文档中的各模型部分

**格式处理：**

数据处理适用于训练数据处理和模型输出数据的处理，通常模型输出数据的格式、风格会与训练数据一致。指令微调时可以用那些强大的GPT模型的输出作为自己模型的训练数据，此处的数据主要参考GPT4等输出格式

图片/链接/表情/颜文字：训练时最好删除

换行符\n：通常使用\n\n，偶尔会出现\n，也可能多出空格[\n \n]等

小标题：通常小标题的形式为[1. \*\*内容\*\*\n\n]，\*\*为markdown格式的加粗，也可能出现[\*\*1. 内容\*\*\n\n]等。可能出现1个\*等错误。可能出现[- \*\*内容\*\*]

空格：通常小标题[1. \*\*内容\*\*]中有1个空格、单词之间有1个空格。空格可能出现在其他地方，也可能出现多个空格

**提示词**

提示词工程：在不训练模型的情况下，通过调整提示词(包括系统提示词)提高回答质量

提要求：提出的要求用小标题分别列出，写在不同的行。对于强制性要求不要混合，比如可以先把所有的[你必须XXX]写完，再写所有的[你不能XXX]

前后一致：比如前面提到[写文章]，后面不要变为[写文本]

表达清楚：形容词后面不要省略[的]，句子尽量完整，保证理解上没有歧义

多用肯定句：[你的回答不要太短]改为[你的回答要丰富]

关键信息放最后+加上三引号：[请根据以下内容{做一件事}：{内容}]改为['''{内容}'''。请根据以上内容{做一件事}]

添加预设：只针对背景内容进行回答时，应该加上[请严格按照以上内容回答问题，如果找不到答案要回答“我不知道”]

编程题模版：

'''

一、题目

1，{小要求1}

2，{小要求2}

...

n，输入和输出的内容和格式参考后面的示例。

二、示例

...

'''

请根据以上内容写一个{python}程序。

**任务类型**

**(1)文本分类**

输入一个句子得到句子的类别

[预处理] --> [模型] --> [输出(batch,词数n,长度dim)] --> [(batch,dim,n)] --> [平均池化(batch,dim,1)] --> [(batch,dim)] --> [(batch,类别数)]

衡量指标：准确率、精确率、召回率、F1指标

**(2)序列标注**

对句子中的每一个词做分类

[预处理] --> [模型] --> [输出(batch,词数n,长度dim)] --> [(batch,n,类别数)]

**《TSF》**

**基础知识**

时间序列预测的通常输入为(batch,len,feature)，len表示时间长度。但在模型中运算时通常需要转为(batch,feature,len)，linear层对时间维度(len)拟合的结果比对变量维度(feature)拟合的结果更好，然后用conv1d层对变量(feature)进行升降维变换

单变量自标签预测：输入一段序列的前半段，预测的标签是这段序列的后半段。注意有时虽然模型输入的是多个变量，但它们之间没有进行融合，本质上也是单变量。模型学习的是序列的周期变化

单变量异标签预测：输入一段序列，预测的标签是另一段/多段与其相关的序列。模型学习的是各序列之间的关系

多标签预测：输入多段序列，预测的标签是一段/多段与其相关的序列，模型内部要对各输入序列进行融合。模型学习的是各序列之间的关系

**数据处理**

数值可能没有固定的范围，常用z-score：减均值、除以标准差，但如果训练集和测试集数据差别比较大时，以历史数据为标准的处理方法可能不适用于未来数据

z-score：假定数据基本满足高斯分布，

**注意力应用**

词向量编码(embedding)：可以用linear替代。计算方式：x=网络(x)

位置编码：可以定义参数torch.nn.Parameter作为位置编码，将词向量编码后的结果与其相加，然后进行归一化torch.nn.LayerNorm(feature)。计算方式：x=归一化(x+网络(x))

编码层：序列从编码层输入，得到编码层的输出

解码层：可以定义参数torch.nn.Parameter作为解码层输入，形状为(1,n,feature)，需要repeat为(batch,n,feature)。编码层的输出也要输入到解码层的编解码注意力层中

经典的transformer在时序预测中的效果可能还不如简单的linear层，通常都会对编码模块和解码模块进行改造

序列平稳性：传统时序预测模型要求输入的序列具有平稳性，可以利用一阶差分实现

一阶差分：用后1时刻的值减去前1时刻的值。在深度学习中加入影响不大

自回归模型(AR)：根据历史对当前值进行预测，需要满足平稳性，需要确定参数p

移动平均模型(MA)：用于消除预测中的随机波动，需要确定参数q

自回归差分移动平均模型(ARIMA)：用于时序预测的机器学习算法。statsmodels库

**《3D》**

**基础知识**

3D模型：3D模型有非常多的格式，因为不同的任务中需要的模型信息不同。通常一个完整的模型由点坐标、面(点之间的连线)、法线、纹理、材质等组成，点与面构成了最基础的模型，其他部分控制模型的细节，比如弯曲的面、模型颜色、光线角度等。模型的点越多则细节越丰富，面通常为三角形

渲染：将3D模型在某个角度变成可视化的2D图像

可微渲染：一种渲染技术，渲染过程可微分，将计算机图形学和深度学习相结合

硬边：完全平整的面组成的模型是充满棱角的，比如正方体

软边：渲染时对面和面之间的边进行平滑的弧度处理，比如球体。在3D软件中使用软边渲染需要2个面之间的夹角较大

法线：垂直于面的线，每个面和点都有法线，点的法线由点四周的面加权平均得到。法线的存在是为了控制面和边的弯曲、光照反射。法线分为正法线和负法线，负法线所在的那一侧面不接收光照，节约计算时间。通常只需要面法线就可以计算出点法线

材质(material)：抽象概念，模型表面的外观特性，比如颜色、反射率、光泽度等。一个模型可能有多个材质，分别管理不同的区域

材质球(material ball)：贴图与模型的媒介。建模软件中先将贴图和参数存放到材质球中，然后将材质球应用到模型上

纹理(texture)：抽象概念，物体表面的纹路图案

贴图(map)：有基础贴图和法线贴图，左下角为原点，通过贴图着色的方式可以节约存储空间和渲染时间

基础贴图(base map)：充满颜色纹理的图片。基础贴图可以是材质烘焙后的结果

法线贴图(normal map)：法线贴图可以改变物体表面的凹凸细节

UV坐标(x,y)：模型每个顶点都有一个UV坐标，用于在纹理贴图上找到对应的位置，如果一个面有3个点，则该面的图案就是纹理贴图中3个点内的区域。UV坐标数值在0-1，因此运算时纹理贴图的坐标也压缩到0-1，如果UV坐标超过1比如(1.1,0.5)，则通常会把其当做(0.1,0.5)

UV贴图：将UV坐标展开(画在透明图上)再根据面连线得到的图。在软件中为一个透明的框架，悬浮在纹理贴图上，如果模型没有纹理贴图，则背景是透明的。如果UV贴图和纹理贴图不对应(UV坐标不对应)则物体的颜色会乱

着色器(shader)：用于给模型上色，有贴图着色、顶点着色、混合着色。贴图着色是把纹理贴图贴到模型上；顶点着色是给每个顶点一个颜色，点间面的颜色取决于点的颜色

烘焙(baking)：将模型的着色或法线信息等按照UV展开的方式变成一张纹理贴图

**文件格式**

stl：最简单的基本几何图形模型，只是一个三角形列表

ply：在stl的基础上支持索引面和颜色

obj：常见的格式，可以用txt文本打开查看，优点是支持各种类型的网格，缺点是没有统一的规范、使用纹理需要一或多个mtl文件(可能有纹理贴图)。obj文件中要指定mtl文件位置，mtl文件中包含材质信息，每个材质使用纹理贴图时需指定其位置

fbx：比较完整的格式，包含材质

glb：比较完整的格式，包含材质，glb中只有三角形网格。推荐用glb格式

ghtf：文件内容与glb一致，只是将文件和纹理分开存储

**obj文件：**

用txt格式打开后有很多v开头、vt开头、vn开头、f开头等的行

mtllib 名称.mtl：指定模型渲染使用的材质文件

v x y z：顶点坐标

vt u v w：纹理图片坐标，通常w默认为0会省略

vn x y z：法线坐标

f v1/vt1/vn1 v2/vt2/vn2...：面索引

**mtl文件：**

可以用txt格式打开，里面有很多行

newmtl 材质：材质名称，可能有多个材质

Ka r g b：环境光照，(r,g,b)的值为0-1对应0-255

Kd r g b：漫反射光照

Ks r g b：镜面光照

Ni：折射率(0.001-10)，为1时光通过物体不弯曲，玻璃折射率为1.5

Ns：反射指数(0-1000)，值越高则反射高光越密集

illum 整数：光照模型，0时颜色为Ka，1时颜色为Ka+Kd，2时为Ka+Kd+Ks

map\_Kd 纹理图片.png：指定纹理图片位置

**图形渲染**

三维图像在电脑显示需要考虑模型的图形数，需要用到图形渲染库，常见的库：OpenGL、Vulkan、DirectX

OpenGL：最早的图形渲染库，兼容性很强但封装比较多，不方便利用并行计算和多线程能力，在复杂图形应用中性能略差

DirectX：包含Direct2D、Direct3D等，广泛应用于windows系统和游戏中

**三维重建**

[图片坐标系]-->[拍摄坐标系]-->[重建坐标系]

图片坐标系：拍摄时物体会通过小孔映射到摄像机内部形成图片，二维坐标系

拍摄坐标系：拍摄的时候以相机为中心建立的三维坐标系

重建坐标系：自定义的三维坐标系，需要先构建一个空白的立方体(volume)，再不断填充立方体的体积元素(voxel)。立方体越大、体积元素越小则重建模型越细致，但耗性能

[图片坐标系]-->[拍摄坐标系]：经过一个变换矩阵，需要摄像机的内参

[拍摄坐标系]-->[重建坐标系]：经过一个变换矩阵，由旋转矩阵和平移矩阵组成，需要摄像机的外参。旋转矩阵：仿射变换，形状为3\*3。位置矩阵：形状为3\*1，代表在x、y、z上的平移量

相机标定：从图片坐标系到重建坐标系需要摄像机的内外参数，反之如果知道部分点在图片坐标系和重建坐标系的位置就可以求出摄像机的内外参数

《**多模态》**

clip(2021)是多模态发展的经典之作，在clip模型中图片模型和文本模型是分开的，因此可以做多个类型的任务。除此之外也有一些合在一起端到端的模型，但基本原理也是通过特征向量将图片与文本结合

从视觉多模态的角度看，RAM(2023)是图片分类的未来发展方向，GroundingDINO(2023)是目标检测的未来发展方向。即使它们目前的表现还存在不稳定，但在未来或许能集成一种视觉类的通用模型，就像GPT在nlp领域一样，并且通过与GPT结合实现通过语言对图像的精确操控

多模态模型对齐：由于输入数据有多个(如图片和文本)，经过网络后要让它们的特征对齐，这个特征类似标签，让不同的输入数据之间建立关系

对比学习：图片和文本先经过各自的模型得到各自的特征，再利用特征进行相似度匹配。如CLIP(2021)模型

**AI绘画模型lora训练**

可以使用网上封装好的GUI界面方便训练：<https://github.com/bmaltais/kohya_ss>

单个风格：训练一种风格的至少需要100多张图片，通常2批量下需要训练约3000步

图片标签：先使用自动打标签的软件初步标记，再使用对应的英文词汇翻译软件理解，最后进行标签的删除与增加。要学习的风格等元素不要加上标签，不想学习到的部分比如背景颜色、人物颜色等尽量加上标签

**《模型训练》**

**基础知识**

数据类型：一般训练时用float32，GPU可启用混合精度训练，预测时GPU可用float16

训练集：训练模型的数据

验证集/开发集：每轮训练后测试模型的数据，并以此调整模型和参数

测试集：训练后测试模型性能的数据，不能以此调整模型和参数

预训练模型：加载模型官方在大型数据集上训练好的模型权重，可以大幅加快训练速度

训练速度：通常参数越多的模型训练和推理速度越慢，但除了参数量也与网络结构有关，好的网络结构即使参数很多但训练和推理速度很快，指标也更好。很多模型有官方的预训练权重，可以加快训练速度。模型训练速度比模型验证速度快很多，因为不需要反向求导和更新参数

训练震荡：在训练开始后的一段时间或训练中途，有可能会出现指标异常的情况，越大的模型训练越慢，也越容易出现震荡，之后会慢慢变得平稳。在训练稳定之后继续训练可能出现指标下降的现象，需要保存最佳结果。在加载已有模型接着训练时，学习率、指数移动平均等最好也为保存时的状态，否则可能出现震荡

图片分类训练异常：在多分类的训练过程中，类别样本有的多、有的少，可能会出现只有一个类别指标正常，其他类别指标下降为0的异常现象，一般要等到正常的类别指标稳定之后，其他类别的指标才会依次回归正常。因为模型为了达到损失最小，会优先学习样本多的类别而暂时‘忽略’样本少的类别

轮次(epoch)：当[迭代次数\*批量大小<=训练集]时为一轮，每轮训练前都要打乱训练数据，增加随机性，防止模型对前面的数据遗忘

迭代次数：推理一次为一次迭代。若训练集100个，批量大小为10，则每轮迭代10次

批量大小(batch\_size)：每次迭代放入多少数据（即训练完多少数据更新一次参数）。相同数据时批量越大更新迭代次数越少、收敛越快，在内存允许时一般尽量用最大批量训练，最好是CPU/GPU数2的幂次，但要留有一定的GPU显存，一般在训练+验证完一轮后会略微增加显存占用。对于GNN的数据，可能每组数据都有上万的点，在不大幅度调整学习率的情况下采用批量为1时的效果可能会更好

零样本学习(zero-shot learning)：模型没有学习过类似样本但能预测

一样本学习(one-shot learning)：模型通过1个样本学习就能进行预测

少量样本学习(few-shot learning)：模型通过少量样本学习就能进行预测

**训练方法**

混合精度训练：通常采用float32训练，预测用float16。但对于复杂的网络通常有参数不太重要的层，可以用float16精度混合训练

指数移动平均(EMA)：将原模型复制一份作为ema模型，在训练中平滑的记录原模型参数，减少极端值的影响，关闭梯度，不影响模型训练，只影响最终结果。训练中每次原模型更新参数后，ema模型会根据原模型更新自己的参数。分布式时只在主机上创建和更新ema模型。EMA指数平均移动公式：

\*其中为当前原始值，加权后的值，为上一次加权后的值，a为权重系数

分布式训练：有数据并行、模型并行、流水线并行，有单机多卡、多机多卡等情况。通常使用数据并行。分布式训练效果可能会比单卡差，刚开始损失下降可能会很慢

数据并行(DP)：计算效率高但占显存。运行时代码会在不同GPU上运行，模型也会复制并放到不同GPU上，每个GPU对一个batch推理和计算损失(先完成的GPU会一直等待)，之后将损失结果平均并进行反向传播，所有GPU上的模型同步更新参数。验证和计算指标也可以设置为分布式。模型保存等代码只需在主设备上运行

模型并行(MP)：显存利用率高但计算效率低。会把模型分成多份放到不同的GPU上，数据按顺序在不同的GPU上计算。训练速度受GPU之间的数据传输影响较大。训练大语言模型时如果显存不够可以考虑模型并行，但通常模型并行会和数据并行一起，也就是流水线并行

流水线并行(PP)：多个模型并行、数据并行的复杂形式，需要对模型进行精细分割

ZeRO-Offload：充分时利用CPU内存，将优化器状态和梯度卸到CPU内存中以减少显存

DeepSpeed：综合性的大语言模型训练框架，包含各类模型训练优化方案，针对transformer结构通过块稀疏计算降低注意力层的内存需求

BGD(batch gradient descent)批量梯度下降：每次训练时放入所有样本

SGD(stochastic gradient descent)随机梯度下降：每次训练时放入单个样本，会导致模型对最后使用的样本拟合更好，容易导致参数来回更新，收敛慢

MBGD(mini-batch gradient descent)小批量梯度下降：每次训练时放入部分样本，小批量样本与整体样本之间存在偏差，提高了泛化能力。1，使用前要打乱样本顺序；2，样本数量选取是关键（GPU运算可以考虑使用2的幂次数量）



**学习率**

学习率：学习率优化器通常使用adam和L2，一般sgd的学习率要比adam的要高10倍，通常所说的学习率指adam的情况下。注意当训练批量很大时，学习率也要适当变大，通常批量增加a倍时学习率可以增加根号a倍，但学习率增加不要超过10倍

图像类adam：学习率可以从0.001开始下降到0.001\*0.01

时间序列预测adam：学习率可以从0.0001开始下降到0.0001\*0.01。归一化后的mse损失要到0.1以内才能有好的结果

大模型adam：自监督增量训练学习率在0.00001以内，可以下降到\*0.1。指令训练在0.0001以内，可以下降到\*0.1。通常peft模型训练10-20轮就有很好的效果

图卷积adam：学习率可以从0.001开始下降到0.001\*0.01

模型微调：使用模型官方在大型数据集上训练好的模型权重，可以利用模型在其他数据上学习到的参数，用很小的学习率训练和少量的数据就能得到很好的效果。bert分类模型微调学习率通常在0.00001左右，如果学习率大容易破坏参数

学习率预热：刚开始训练时让学习率从很小开始增加，在几次迭代之后逐渐变为初始学习率。通常取总步数的0.01步为学习率预热，学习率从lr\_start\*0.1逐渐上升到lr\_start。在训练的初始阶段，模型的权重通常是随机初始化的，模型在初始阶段更新过快可能会影响模型的收敛

学习率衰减：每轮训练后，一般要调整学习率的大小，使学习率随着训练轮次的增加越来越小。对于复杂的网络，末尾学习率的设置可以不用太小，比如使用adam算法时，初始学习率设置为0.001，末尾学习率可以设置为0.001\*0.01。可以使用余玄下降公式：

其中为训练轮次，n\_为预热训练总轮次，为学习率调整总次数，是初始学习率

Adagrad(adaptive gradient)自适应梯度：在凸函数上有快速单调收敛的效果，但存在学习率过早减小的情况，在非凸函数上效果不好。为梯度，c为防止分母为零的极小常熟，可为

RMSProp(Root Mean Square Prop)均方根道具：在Adagrad基础上加了超参数来衰减历史值，目前为常用优化算法

**指标**

TP：正确判断正类为正类

FN：错误判断正类为负类

TN：正确判断负类为负类

FP：错误判断负类为正类

准确率(accuracy)：T/(T+F)

精确率(precision)：TP/(TP+FP)

召回率(recall)：TP/(TP+FN)

F1\_score：(2\*precision\*recall)/(precision+recall)。其中F1——micro先统计总TP等再计算；F1\_macro计算各类别F1再平均(常用)

排列熵(Permutation Entropy)：度量时间序列复杂性的一种方法，时间序列复杂性越低，序列的可预测性就越强

模型大小：约等于[参数量\*字节数/1024/1024](MB)。float32占4字节

**《模型预测》**

[模型] --> [模型剪枝训练] --> [模型蒸馏训练] --> [模型量化训练] --> [预测代码] --> [gradio可视化] --> [falsk服务部署] --> [gunicorn多进程] --> [docker镜像打包]

模型预热：模型刚开始预测速度会比较慢，之后速度会加快并趋于稳定

模型推理预处理耗时：包括数据解码、图片等尺寸转换、将张量放到gpu上，一般解码和尺寸转换时间略高，将批量张量一起放到gpu上比单个放快很多。但如果是trt模型，批量预测相对单张预测只有在小模型上有一点优势，大模型上差不多，甚至单张预测更快，反而单张预测占显存更少

模型推理：目标检测一般置信度阈值为0.35，非极大值抑制阈值为0.65。一般在内存允许时用大批量推理会快很多。但对于trt模型无论占gpu显存大小，单张预测和批量预测只有在小模型上有一点优势，大模型上差不多甚至单张更快

推理精度：模型推理使用的数据类型一般为float32、float16或int8，一般使用float16几乎不改变精度(归一化到0-1的输出，误差数量级在0.001以内，在图片中不到一个像素)，但量化为int8会有较明显的影响。相同的非极大值抑制阈值下，可能量化前抑制只有一个框，量化后抑制会有两个框，此时需要降低非极大值抑制阈值，但即使只留下一个框，框边的精确度也会下降。使用float16精度时，如果pytorch版本不一样，可能会出现nan的情况

剪枝：使用L1或L2正则化后，批归一化层中不重要结构的参数会很小，根据参数的大小去除卷积层中不必要的通道，一般裁剪后保留的通道数可为30%-80%

蒸馏训练：用一个性能更好的大模型来微调一个小模型。将数据同时输入到两个模型中，然后把大模型的输出作为标签来调整小模型

量化**：**将模型浮点数(如float32)替换为整数(int8)

非饱和方式：若浮点数正负绝对值的最大值为a，则将-a至a映射到整数的最大值和最小值，方便计算，但有效值的动态范围会减小

仿射方式：将浮点数最大最小值对应映射到整数的最大最小值

饱和方式：先计算浮点数的阈值T，然后根据浮点数的正负阈值T饱和截断，之后映射到整数的最大最小值

动态离线量化：非饱和方式量化权重，不需要样本数据。

静态离线量化(PTQ)：饱和方式量化权重，非饱和方式量化激活。需使用少量的样本数据进行模型的前向计算，对激活进行数值采样。PTQ在大模型上精度损失比小模型少

量化训练(QAT)：非饱和方式量化权重和激活，并在训练过程中更新权重，需要使用大量有标签样本数据

**《模型项目》**

**ImageNet**

2007年ImageNet项目创立，为ILSVRC比赛提供数据支持

**AlexNet(2012)**

论文：ImageNet Classifification with Deep Convolutional Neural Networks

2012年ILSVRC比赛冠军，8层(不算pool)



**OverFeat(2013)**

论文：OverFeat:Integrated Recognition,Localization and Detection using Convolutional Networks

2013年ILSVRC比赛冠军，OverFeat为AlexNet的改进版，有快速版和精确版两个版本

**VGGNet(2014)**

论文：VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

2014年ILSVRC比赛分类任务亚军和定位任务冠军，有A-E五种模型

**GoogleNet(2014)**

论文：Going deeper with convolutions

2014年ILSVRC比赛分类任务冠军和定位任务亚军。GoogleNet有3个分类器，另外两个辅助分类器接4a和4d后，但在比赛中Googlenet只使用了最后的分类器。

GoogleNet提出inception层，用不同卷积核提取网络再合并，利用稀疏性解决参数量大和节约计算资源，加入卷积减少参数量

**ResNet(2015)**

论文：Deep Residual Learning for Image Recognition



2015年ILSVRC比赛冠军。模型常为50和101层残差网络。加入residual结构，特征融合时采用相加，当输入和输出形状不同时，要用的卷积核对分支升维(不加激活函数)。对于50/101/152层的网络用右边的结构可大幅减少参数

**ResNext(2016)**

论文：Aggregated Residual Transformations for Deep Neural Networks



2016年ILSVRC比赛冠军。ResNext为ResNet的改进版，相当于inception和residual的结合

**SENet(2017)**

论文：Squeeze-and-Excitation Networks



2017年ILSVRC比赛冠军

**图像分类**

**Private Detector(2022)**

github：https://github.com/bumble-tech/private-detector

开源色情分类模型

**Efficientnetv2(2021)**

<https://github.com/google/automl/tree/master/efficientnetv2>

**INTERN(2023)**

论文：InternImage: Exploring Large-Scale Vision Foundation Models with Deformable Convolutions

github：<https://github.com/OpenGVLab/InternImage>

商汤科技发布的多模态多任务通用大模型，拥有30亿参数，以CNN网络为核心。在相同的主干网络上微调，可以实现不同的任务类型

**目标检测**

**Mask RCNN(2017)**

论文：Mask R-CNN

github：<https://github.com/matterport/Mask_RCNN>

或https://github.com/facebookresearch/Detectron



在Faster RCNN(2015)上做了改进，速度差不多但性能提高，可用于目标检测、图像分割和人体姿态估计。Mask RCNN最后只取出对应类别的通道做损失计算，不存在类别对抗

**CSPNet(2019)**



增强了网络学习能力，使网络轻量化、减少内存的同时保持准确性

**Vision Transformer(2020)**

论文：An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

github：<https://github.com/google-research/vision_transformer>

2020年goodle提出的将transformer应用于视觉的结构。

**YOLOV4(2020)**

论文：YOLOv4:Optimal Speed and Accuracy of Object Detection



将Darknet53改为CSPDarknet53。使用特殊池化金字塔SPP(Spatial Pyramid Pooling)和PANet结构。使用CIOU作为损失函数。CSPDarknet使用Mish激活函数，其他依然是Leaky Relu。使用马赛克(Mosaic)增强

SPP结构前后都用卷积降维，池化卷积核大小为5、9、13

**YOLOV5(2020)**

github：https://github.com/ultralytics/yolov5



P为模型最终预测值，,为预测框的中心点，Y为模型输出，G为网格左上角坐标，stride为缩小的倍数[8,16,32]。YOLOV5的预测值域变为了和[0,4]，是为了将标签网格临近的两个网格也变为标签，[0,4]是为了对应标签筛选条件

各版本YOLOV5的主体网络框架不变，只改变C3层的残差数和通道维度。[第一层输出维度,C3层残差数]：YOLOV5n[16,1]，YOLOV5s[32,1]，YOLOV5m[48,2]，YOLOV5l[64,3]，YOLOV5x[80,4]

SPP层改为SPPF层，输出前后同样用卷积降维，池化卷积核都为5

YOLOV5目前使用CSP层、C3层和SPPF层，使用Silu激活函数，使用CIoU，使用标签平滑。标准输入，输出三层为、、

**YOLOX(2021)**

论文：YOLOX Exceeding YOLO Series in 2021

github：<https://github.com/Megvii-BaseDetection/YOLOX>

使用Focus和YOLOV4中的CSPnet、SPP进行特征提取

YOLOVX改进解耦合头，原YOLO中只有最后一层的卷积用于得到边框、置信度、类别三种结果，即使前面的的特征提取网络提取很充分，但不用任务之间存在对抗，只用一层卷积难以区分三者的差异。YOLOVX解耦头先将边框与置信度、类别分开卷积，再将置信度与类别分开卷积。YOLOVX不需要先验框(Anchor Free)

SimOTA动态匹配正样本：根据真实框和预测框的重合度计算真实框的k，根据真实框和特征点的预测准确度及包含情况计算Cost代价矩阵，将Cost最低的k个特征点作为真实框的正样本

**YOLOV7(2022)**

论文：YOLOv7:Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors

github：[https://github.com/WongKinYiu/yolov7](https://github.com/Megvii-BaseDetection/YOLOX)



提出了elan、elan-h层作为YOLOV5的C3层改进版，提出了加速推理的repconv结构

repconv：其思想源于repVGG，在训练时加入特殊的残差结构辅助训练，但在预测时可以等效为一个的卷积，在网络复杂度下降的同时预测性能却没有下降

官方版本训练：与YOLOV5基本一致，只是.yaml文件读取略有区别，小模型使用train.py训练，大模型(W,E,D)使用train\_aux.py训练

**YOLOV8(2023)**

github：<https://github.com/ultralytics/ultralytics>

根据YOLOV5的C3和提出了YOLOv7的elan提出了C2f结构

**insightface(2021)**

github：<https://github.com/deepinsight/insightface>

github：<https://github.com/TWK2022/insightface>

github：<https://github.com/TWK2022/FaceRecognition>

insightface是2021年的人脸检测模型。模型输出(xmin,yxmin,xmax,ymax)

**ultralytics(2025)**

(pip install ultralytics)

github：<https://github.com/ultralytics/ultralytics>

ultralytics库中整合了所有yolo系列的模型

**图片分割**

**SAM(2023)**

论文：Segment Anything

github：<https://github.com/facebookresearch/segment-anything>

示例网址：<https://segment-anything.com/>

2023年facebook发布的图像分割模型，能对图片进行分割，可用于抠图等。有vit\_b(358M)、vit\_l(1.16G)、vit\_h(2.39G)三个型号，要单独下载模型

模型可以传入正面提示点、负面提示点、正面提示框，根据传入的信息分割模型

**Semantic-SAM(2023)**

github：<https://github.com/UX-Decoder/Semantic-SAM>

2023年的图像分割模型，在SAM模型的基础上进行了改进，提升了分割性能，并且可以更灵活的选择不同的分割程度(比如只分割出窗户还是分割出整辆车)

**SEEM(2023)**

论文：Segment Everything Everywhere All at Once

github：<https://github.com/UX-Decoder/Segment-Everything-Everywhere-All-At-Once>

示例网址：<https://huggingface.co/spaces/xdecoder/SEEM>

2023年的图像分割和识别模型，在对图片进行分割的同时，还会对分割的物体进行识别。分割性能相比SAM有一定的提升

**SAM2(2024)**

github：<https://github.com/facebookresearch/segment-anything-2>

SAM2转为onnx：<https://github.com/ibaiGorordo/ONNX-SAM2-Segment-Anything>

2024年facebook发布的图像分割模型，在SAM模型的基础上做了改进，模型体量变小，并支持视频分割

**图片修复**

**Real-ESRGAN(2022)**

github：<https://github.com/xinntao/Real-ESRGAN>

对模糊图片进行细节重建，对漫画、物体等效果较好，对人脸、文字效果一般。其中的realesrgan-x4plus为推荐模型，realesrgan-x4plus-anime针对动漫插画，realesr-animevideov3针对动漫视频

**时间序列预测**

**LSTM(1997)**

论文：LONG SHORT-TERM MEMORY

LSTM规避了标准RNN中梯度爆炸和梯度消失的问题，学习速度更快

****

其中为函数。LSTM中丢弃层不宜太高，一般为0.2；LSTM输出维度太低对结果有明显影响，可为64/128...；LSTM后加卷积可小幅度提高网络性能，但卷积不宜过多

**Informer(2020)**

论文：Informer Beyond Efficient Transformer for Long Sequence Time Series Forecasting

github：<https://github.com/zhouhaoyi/Informer2020>

2020年提出的模型，目的是在transformer 的基础上改进，降低时间复杂、提高计算效率和提高预测能力。它提出一种新的结构：ProbeSparse自注意机制，这是一种稀疏的注意力机制，只对部分的输入使用注意力机制，并证明可以在保持性能的同时大幅减少计算量

**Autoformer(2021)**

论文：Autoformer Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting

github：<https://github.com/thuml/Autoformer>

2021年提出的模型，目的是在transformer 的基础上改进，提高对长期时间序列预测的效果。transformer 有强大的学习能力，但在长期时间序列预测上效果不佳。而Autoformer将序列分解成两部分：周期趋势部分和偏差部分，周期趋势部分代表序列主要的趋势，偏差部分代表噪声和季节性等偏离部分

**SCINet(2021)**

论文：Appendix: Time Series is a Special Sequence: Forecasting with Sample Convolution and Interaction

github：<https://github.com/cure-lab/SCINet>

论文在7个流行的数据集上测试，大多数情况下优于现有的时间序列预测（TSF）方法。论文证明TCN网络中的因果卷积并非必要。SCINet可以降低时间序列的排序熵，理论上更容易预测

**FEDformer(2022)**

论文：FEDformer Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting

github：<https://github.com/MAZiqing/FEDformer>

**NLinear(2022)**

论文：Are transformers effective for time series forecasting?

github：<https://github.com/cure-lab/LTSF-Linear>

文章证明用一些简单的全连接层其实就比几乎所有基于的transformer的模型好

NLinear主体就是全连接层，只是在全连接前/后要减去/加上序列的最后一个值，起到归一化的效果，让模型更快收敛

**Crossformer(2023)**

论文：Crossformer: Transformer Utilizing Cross-Dimension Dependency for Multivariate Time Series Forecasting

github：<https://github.com/Thinklab-SJTU/Crossformer>

**iTransformer(2024)**

论文：iTransformer: Inverted Transformers Are Effective for Time Series Forecasting

github：<https://github.com/thuml/iTransformer>

iTransformer擅长变量数很多时间序列预测。对于很长的序列依然有很好的预测效果

**Pathformer(2024)**

论文：Pathformer: Multi-scale Transformers with Adaptive Pathways for Time Series Forecasting

github：<https://github.com/decisionintelligence/pathformer>

**自然语言处理**

**Transformer(2017)**

论文：Attention is All You Need

github：<https://github.com/huggingface/transformers>

使用文档：<https://blog.csdn.net/zhaohongfei_358/article/details/126019181>

内部机制：<https://zhuanlan.zhihu.com/p/109484539>



**ERNIE 3.0(2022)**

github：<https://github.com/PaddlePaddle/PaddleNLP/tree/develop/model_zoo>

百度提出的超大语言模型，在多个数据集上均优于以往模型

**NLLB(2022)**

github：<https://github.com/facebookresearch/fairseq/tree/nllb/examples/nllb/modeling>

2022年facebook发布的语言翻译模型，能够在200种语言之间翻译

**LLaMA(2022)**

2022年facebook发布的大型语言模型

**Chinese-Vicuna(2023)**

github：<https://github.com/Facico/Chinese-Vicuna>

2022年Meta发布LLaMa模型，有7B、13B、33B三个型号型号。此后中文团队训练出中文模型补丁，中文模型只是部分权重，要与原模型合并才能使用。由于LLaMa的使用需要申请，因此官方不提供直接合并好的中文模型

**ChatGLM3(2023)**

github：<https://github.com/THUDM/ChatGLM3>

2023年智谱AI和清华大学实验室联合发布chatglm3模型，有6B、6B-32K、6B-128K三个型号。词表长度：64798

**Chinese-LLaMA-Plus/Chinese-Alpaca-Plus(2023)**

github：<https://github.com/ymcui/Chinese-LLaMA-Alpaca>

2022年Meta发布LLaMa模型，有7B、13B、33B三个型号。此后中文团队训练出中文模型补丁，中文模型只是部分权重，要与原模型合并才能使用。由于LLaMa的使用需要申请，因此官方不提供直接合并好的中文模型。词表长度：49954

**Chinese-LLaMA-Alpaca-2(2023)**

github：<https://github.com/ymcui/Chinese-LLaMA-Alpaca-2>

2023年Meta发布LLaMa2模型，有1.3B、7B、13B三个型号。此后中文团队训练出中文模型版本。词表长度：55296

**Baichuan2(2023)**

github：<https://github.com/baichuan-inc/Baichuan2>

2023年百川智能发布Baichuan2模型，有7B、13B两个型号。词表长度：125696

**Qwen(2023)**

github：<https://github.com/QwenLM/Qwen>

2023年阿里发布qwen模型，有1.8B、7B、14B、72B四个型号。词表长度：151851

**DeepSpeed(2023)**

github：<https://github.com/microsoft/DeepSpeed>

2023年微软发布DeepSpeed训练框架，解决了小显存无法训练大语言模型的问题，并提高大语言模型的训练效率

**aiXcoder(2024)**

github：<https://github.com/aixcoder-plugin/aiXcoder-7B>

2024年硅心科技发布aiXcoder-7B模型，侧重于代码生成。词表长度：49152

**Llama3-Chinese(2024)**

github：<https://github.com/LlamaFamily/Llama-Chinese>

github：<https://github.com/ymcui/Chinese-LLaMA-Alpaca-3>

2024年Meta发布LLaMa3模型，有8B、70B两个型号，使用15万亿token数据训练。此后中文团队训练出中文模型版本。llama3的提示模版与llama2的不同。词表长度：128256

使用GroupQueryAttention、RMSNorm、silu。GroupQueryAttention的group为8、不加入偏置项、不使用丢弃层。RMS归一化参数为0.00001

**Nemotron-4-340B(2024)**

2024年英伟达开源的大模型，有Nemotron-4-340B-Base、Nemotron-4-340B-Instruct和Nemotron-4-340B-Reward三个型号。其中Nemotron-4-340B-Reward可以帮助其他模型筛选AI回答的质量

**qwen2(2024)**

github：<https://github.com/QwenLM/Qwen2>

2024年阿里发布qwen2模型，有0.5B、1.5B、7B、14B、72B五个型号。词表长度：151646

除了词表长度151646与qwen的151851不一致外，其余的特殊字符、提示词模版、训练方法等完全一致。qwen2相比qwen在网络结构方面将解码模块的堆叠数减少，增加了mlp网络的中间隐藏层数。72B能处理128k长度输入，安全测试得分与GPT4差不多

tokenizer变化：'<|im\_start|>system\n{system}<|im\_end|>\n'为系统提示词标志，同理将system改为user为用户输入标志，改为assistant为模型回答标志，模型回答的结束标志为<|im\_end|>\n。填充符为<|endoftext|>对应151643，<|im\_start|>对应151644，<|im\_end|>对应151645

generation\_config变化：eos\_token\_id=[151645,151643]，pad\_token\_id=151643

**llama.cpp(2024)**

github：<https://github.com/ggml-org/llama.cpp>

2024年使用c++实现的替代pytorch进行高效推理的库，提供将模型量化为float16、int8、int4的方法，针对不同平台进行了优化。需要先将模型转为gguf格式，gguf是一种包含了模型本身和所有配置文件的二进制格式。llama.cpp需要c++环境运行，安装时用cmake构建，安装后可以从python中的llama\_cpp\_python库调用

**ollama(2024)**

github：<https://github.com/ollama/ollama>

大模型部署程序，相当于把llama.cpp和flask进行了集成，使用gguf模型推理

**llama\_index(2024)**

官方文档：<https://docs.llamaindex.ai/en/stable/>

llama\_index是2023年推出的数据检索库，专注于高性能的检索和rag构建，包含文本编码模型使用(需额外下载)、对数据库的高效检索和管理，注意它与llama2模型没有关系。llama\_index是一个大的框架，里面集成了很多其他的库，会略为杂乱

**langchain(2024)**

github：<https://github.com/langchain-ai/langchain>

langchain是2022年10月开源的一个项目，后来成为一家初创公司。langchain是一个大的框架，里面集成了很多其他的库。可以调用ollama、llama\_index等库

langchain用于智能助手的构建，包括多个模型、知识库、用户历史等整个系统的管理

**qwen3(2025)**

github：<https://github.com/QwenLM/Qwen3>

2025年阿里发布qwen3模型，有0.6B、1.7B、4B、8B、14B、32B、30B-A3B、235B-A22B共8个型号。词表长度：151851

qwen3模型加入了思维链，提高了模型的推理能力，通过enable\_thinking参数控制，训练时如果要调整思维链，需要加入思维链的标签

**deepseek\_v3(2025)**

github：<https://github.com/deepseek-ai/DeepSeek-V3>

2025年deepseek开源的混合专家(MoE)大模型，671B。使用多头潜在注意力层(MLA)

**deepseek\_r1(2025)**

github：<https://github.com/deepseek-ai/DeepSeek-R1>

2025年deepseek开源的深度思考大模型DeepSeek-R1，671B。除此之外还提供了qwen等开源模型经过DeepSeek-R1知识蒸馏得到的模型，比原版性能更好

**图神经网络**

**GCN(2013)**

论文：Semi-supervised classification with graph convolutional networks

GCN(Graph Convolutional Network)是2017年提出的图卷积网络，更多理论详见《神经网络》->图卷积网络(GCN)

**Graphormer(2021)**

github：<https://github.com/microsoft/Graphormer>

Graphormer是2021年微软开源的图神经网络，获得了2021年知名图网络比赛冠军

**PDEformer(2024)**

**gitee：**<https://gitee.com/mindspore/mindscience/tree/master/MindFlow/applications/pdeformer1d>

PDEformer是偏微分方程 (PDE) 的神经求解器，能够同时求解各种类型的 PDE。通过图 注意力层(graph transformer)和隐式神经表示(INR)来生成无网格预测解

PDEformer在三百万条不同形式的一维偏微分方程(PDE)数据上训练后，预测效果超过FNO和DeepONet等仅针对特定方程预测的模型，具有很强的泛化性

**多模态**

**CLIP(2021)**

论文：Learning Transferable Visual Models From Natural Language Supervision

github：<https://github.com/openai/CLIP>

2021年openai发布的基于对比学习的多模态模型，用于将文字描述和图片匹配。由一个图片编码模型和一个文本编码模型组成，一张图片经过图片模型得到的特征向量和这张图片的描述经过文本模型得到的特征向量会相近，通过计算余弦相似度可以通过文本找图片、通过图片找文本(官方文本模型只支持英文)

CLIP模型训练：在互联网上用4亿个图片文本对进行训练。图片送入图片模型，文本送入文本模型，将两者输出的特征向量用交叉熵计算损失，也叫对比学习

**\*粤港澳数字经济研究院太乙模型(2022)：**

国内配套的中文文本模型：<https://huggingface.co/IDEA-CCNL/Taiyi-CLIP-Roberta-large-326M-Chinese>

中文文本模型训练：用1.23亿个图片文本训练了一个中文文本模型，图片模型不变，中文文本模型以图片模型的输出特征作为标签，进行交叉熵损失计算

**\*阿里达摩院Chinese-CLIP(2022)：**

github：<https://github.com/OFA-Sys/Chinese-CLIP>

用2亿个图片文本对重新训练了clip模型，训练方法与原clip一致

**Stable Diffusion(2022)**

github：<https://github.com/Stability-AI/StableDiffusion>

提示网站CIVIT：<https://civitai.com/?tag=base+model>

Stable Diffusion主要由clip的文本编码+unet去噪生成+vae变分自编码3个模型组成，实际应用中还会加入图片编码模型、控制模型、各类风格lora模型等，是一类模型的集成。在Hugging Face的开源模型中，通常会提供合在一起的完整模型和拆分开的各模块模型，只是加载方式不同。可以在diffusers库中灵活的加载和替换各模块

扩散(diffusion)：图片的加噪或去噪

文本编码模型(500MB以内)：输入的文本先经过tokenizer编码，再输入到文本编码模型中得到特征向量，然后将特征向量输入unet模型

unet去噪生成模型(5G以内)：图片生成的核心结构，在unet去噪生成过程中，需要很多轮次的迭代，将上1轮的结果输入到下一轮中，迭代次数越多细节越丰富

scheduler采样器：scheduler会对上1轮unet的结果进行处理，再作为下一轮unet的输入，不同的scheduler配置会影响模型生成的结果和时长

vae变分自编码模型(500MB以内)：对最后一轮unet生成的结果进行处理和放大，得到最终的图片

**IP-Adapter(2023)**

github：<https://github.com/tencent-ailab/IP-Adapter>

IP-Adapter是一个主要针对人像的图片编码插件(100MB以内)，需要配合Stable Diffusion模型、clip图片编码模型(2.5G)使用，可以将提示图片信息融入到文生图模型中，实现文+图生图的功能

**RAM(2023)**

github：<https://github.com/xinyu1205/recognize-anything>

2023年的图像识别模型——高级图片分类模型。在clip的基础上进行了改进，输入一张图片后，可以识别图片中常见的所有物体的名称，能够应用到图片的数据标注中

**GroundingDINO(2023)**

github：<https://github.com/IDEA-Research/GroundingDINO>

2023年的文字提示目标检测模型——高级目标检测模型。输入一张图后可以通过文本选择性的检测图片中的物体，比如：“左边的人”

**ComfyUI(2023)**

github：<https://github.com/comfyanonymous/ComfyUI>

模型集成可视化界面，可以将各类多模态模型和工具组合在一起，支持大多数的开源模型，比如将文生图模型和3D生成模型连接到一起就可以组合成文生3D模型

**Grounded-SAM-2(2024)**

github：<https://github.com/IDEA-Research/Grounded-SAM-2>

2024年的文字提示目标检测和分割模型，将GroundingDINO与SAM2模型结合

**deepseek\_vl2(2025)**

github：<https://github.com/deepseek-ai/DeepSeek-VL2>

2025年deepseek开源的视觉模型：DeepSeek-VL2-Tiny(2B)、DeepSeek-VL2-Small(7B)和DeepSeek-VL2(21B)。输入图片，生成图片描述

**3D**

**PointNet(2016)**

PointNet是斯垣福大学在2016年提出的一种点云分类/分割深度学习框架。刚提出的原始模型有些冗余的地方，后来人们提出了改进版PointNet++

**NeuralRecon(2021)**

github：<https://github.com/zju3dv/NeuralRecon>

NeuralRecon是商汤2021年提出的单摄像头三维重建模型。三维重建坐标合成阶段会非常消耗性能，但NeuralRecon没有很复杂的结构速度很快，还能取得不错的效果

模型的输入是一个物体不同视角的9张图片

模型要先根据摄像头中的参数将图片中的像素对应到重建后的结果上

**GrabCAD**

网址：<https://grabcad.com/engineers>

GrabCAD 是全球工程师的CAD 文件共享平台，有400万个可免费下载的CAD文件。但文件格式有近10种：stl、3d object、dxf、c4d、step、stp...

**TripoSR(2024)**

github：<https://github.com/VAST-AI-Research/TripoSR>

2024年开源的3D生成模型，输入一张图片后生成立体3D模型

**Unique3D(2024)**

github：<https://github.com/AiuniAI/Unique3D>

2024年开源的3D生成模型，输入一张图片后生成立体3D模型

**Hunyuan3D-2(2025)**

github：<https://github.com/Tencent/Hunyuan3D-2>

2025年腾讯开源的图生3D模型。由几何生成模型混元3D-DiT、纹理生成模型混元3D-Paint组成，模型先根据图片生成几何网格模型，再根据图片和网格生成纹理贴图