目录

1 介绍及使用方法 1

2 transformer 八股文 2
2.1 attention 计算方式以及参数量 2
2.2 参数量计算 2
2.3 模型的训练参数量分析 2
2.4 为什么要对 QK 进行 scaling，可以不除以 d_K 吗 3
2.5 self-attention 一定要这样表达吗 3
2.6 tranformer 为什么用 Layer Norm 3
2.7 为什么不用 BN 3
2.8 为什么要加一个 position embedding 3
2.9 Bert 为什么三个 embedding 可以相加 3
2.10 transformer 为什么要用三个不一样的 QKV 4
2.11 为什么要用多头注意力 4
2.12 bert 中为什么要用 wordpiece 或者 BPE 这种 subtoken 4
2.13 为什么 bert 要加一个 cls token 4
2.14 bert 中用了哪些 mask 4
2.15 介绍一下 MLM 中的 mask 5
2.16 bert 中 attention 中 mask 有什么用 5
2.17 解释 decoder 中的 mask 5
2.18 有没有什么降低复杂度但是增加输入长度的 5
2.19 为什么之前的 char level 和 subword level 的效果差，但是 bert 里面的效果好 5
2.20 什么是非线性学习路线，为什么 bert 降低了输入复杂度，为什么 bert 提升了模型复杂度 5
2.21 bert 中为什么使用 warm up 的学习率 6
2.22 bert 怎么做到一词多义 6

2.23 Bert 中的 transformer 和传统的 transformer 有什么区别 6

3 tokenization 7
3.1 什么是分词 7
3.2 总结一下文本的三种分词粒度的优缺点 7
3.3 Byte-Pair Encoding(BPE) 怎么构造词典 7
3.4 WordPiece 7

4 位置编码 8
4.1 正余弦位置编码是什么，其具有的性质是什么 8
4.2 理想的位置编码有什么特点 8
4.3 传统的正余弦位置编码看起来很完美，为什么后来没人用了 8
4.4 学习式绝对位置编码 APE 9
4.5 相对位置编码 9
4.6 XLNet 10
4.7 T5 bia 式 10
4.8 DeBERTa 式 11
4.9 ALiBi 位置编码 11
4.10 混合位置编码 12
4.11 ROPE 12

5 attention 及其变体 13
5.1 Multi-query attention 13
5.2 group-query attention 13
5.3 flash attention 13

6 大模型加速 14
6.1 KV cache 14
6.2 大模型的推理时的参数 14
6.3 介绍下 vLLM 14
6.4 vLLM 有哪些优点? 14

7 大模型面试基础问题 15
7.1 目前主流的开源模型体系有哪些 15
7.2 大模型 LLM 的训练目标是什么 15

7.3 为什么现在的大模型都是 decoder-only 的结构 . . . 15
7.4 为什么输入序列不能无限长? . . . 16
7.5 介绍下 FP32,FP16,int8，混合精度 . . . 16
7.6 大模型的评测 . . . 16
7.7 预训练和 SFT 有什么区别? . . . 16
7.8 介绍下 Layer Norm . . . 16
7.9 介绍下 RMS Norm 均方根 norm . . . 16
7.10 介绍下不同的 LN . . . 17
7.11 介绍下激活函数 . . . 17
7.12 什么是稀疏向量和稠密向量 . . . 17
7.13 a . . . 17

8 评价指标和损失函数 . . . 18
8.1 介绍一下 bleu 以及 rouge . . . 18
8.2 介绍下微调的损失函数 . . . 18
8.3 信息墒的公式 . . . 18

9 PEFT . . . 19
9.1 介绍下 fine-tuning . . . 19
9.2 为什么需要 peft? . . . 19
9.3 peft 有什么优点? . . . 19
9.4 peft 对比 . . . 19
9.5 adapter tuning . . . 19
9.6 prompting 提示学习 . . . 19
9.7 介绍下 prompt-tuning? . . . 20
9.8 prompt-tuning 的优缺点? . . . 20
9.9 介绍下 prefix tuning . . . 20
9.10 介绍下 prefix tuning 的优缺点 . . . 20
9.11 p-tuning 与 prefix tuning 的对比 . . . 20
9.12 介绍下 p-tuning . . . 21
9.13 p-tuning 的优缺点 . . . 21
9.14 为什么要有 p-tuning v2 . . . 21
9.15 介绍下 LoRA . . . 21
9.16 介绍下 Qlora . . . 22

9.17 介绍下 Adalora . . . 22
9.18 lora 的优点 . . . 22
9.19 lora 中 rank 怎么选取? . . . 22
9.20 为什么大模型那么耗显存? . . . 22
9.21 什么是指令微调 instructing tuning . . . 22

10 RAG . . . 23
10.1 为什么要有 RAG? 或者为什么 LLMs 需要外挂向量知识库? . . . 23
10.2 什么是 RAG . . . 23
10.3 RAG 的对话思路流程是什么? . . . 23
10.4 RAG 的核心技术是什么? . . . 23
10.5 RAG 的评估指标 . . . 24
10.6 除了余弦距离还了解过其他的距离吗? . . . 24
10.7 了解过对比学习吗? 介绍下 simcse . . . 24

11 RLHF . . . 25
11.1 知乎专栏 https://zhuanlan.zhihu.com/p/624589622 . . . 25
11.2 介绍下 LLM 的经典预训练 pipline . . . 25
11.3 预训练和 SFT 的异同点 . . . 25
11.4 什么是对齐 . . . 25
11.5 介绍一下 RLHF 的流程 . . . 25
11.6 怎么训练 reward model . . . 25
11.7 介绍一下 PPO 算法 . . . 25
11.8 RLHF 的缺点 . . . 26
11.9 如何解决人工产生数据集成本较高的问题 . . . 26
11.10 介绍下 RLHF 里面的四个模型 . . . 26

12 deepspeed . . . 27
12.1 多卡训练的目的是什么 . . . 27
12.2 介绍下 deepspeed . . . 27
12.3 介绍一下数据并行 . . . 27
12.4 介绍一下流水线并行 . . . 27
12.5 介绍下张量并行 . . . 27
12.6 介绍下 3D 并行 . . . 27

12.7 介绍一下 Gpipi 流水线并行 27
12.8 deepspeed 下的 ZeRO 的三个 stage 28
12.9 ZeRO 的核心思想是什么 28
12.10 什么是 offload 28

13 embedding 模型 29
13.1 embedding 模型的作用是什么? 29
13.2 BERT-avg 29
13.3 SBERT 29
13.4 SimCSE 29
13.5 BGE 29

14 面试中的程序题 30
14.1 python 中的 *args,**args 的用法 30
14.2 介绍下 python 的魔术方法 30
14.3 python 里面的深拷贝和浅拷贝 30

15 Linux 中常见命令 31
15.1 readme 31
15.2 cd:change direction 31
15.3 ls 31
15.4 mkdir：Make Directoty 32
15.5 touch 32
15.6 cat,more: 查看文件内容 32
15.7 cp:copy 32
15.8 mv:move 32
15.9 rm:remove 32
15.10 通配符 * 33
15.11 which 33
15.12 find 33
15.13 linux 里 docker-compose 安装 33
15.14 grep：用于搜索文本 34

16 python 处理 pdf 文件 35
16.1 readme 35

16.2 提取文本 . . . 35
16.3 PDF 拆分 . . . 36
16.4 PDF 合并 . . . 36
16.5 创建虚拟环境以及激活虚拟环境 . . . 37
16.6 Jupyter 无法找到 Conda 创建的虚拟环境 . . . 37

17 pandas 常见操作 . . . 38
17.1 readme . . . 38
17.2 创建 Dataframe . . . 38
17.3 获取 Dataframe df 数据的示例代码 . . . 38
17.4 获取 Dataframe 行、列索引的示例代码 . . . 38
17.5 删除一列 . . . 39
17.6 添加行 . . . 39
17.7 横向合并两个 Dataframe . . . 39
17.8 按列求和，求列均值，求列累加和以及一键生成多种统计数据，看 df 的信息，看每一列的值的分布次数，df 的维度 . . . 39
17.9 提取某一列、某一行、某几行某几列 . . . 40
17.10 根据某一列的值的范围进行筛选 . . . 40
17.11 根据某一列的值的范围进行筛选 . . . 40
17.12 改变一列的值，改变某行某列的值 . . . 40
17.13 混合索引 . . . 40
17.14 pandas 读取 excel，存储 excel . . . 40
17.15 数据清洗 . . . 41
17.16 删除重复项 . . . 41
17.17 重命名列名，删除值为空的行，某一行分列 . . . 41
17.18 groupby 分组操作 . . . 41
17.19 排序 . . . 41
17.20 快速索引 . . . 41
17.21 映射到二维平面 . . . 42
17.22 提取列表中某一列（比如'D' 列）包含特定字符的列表 . . . 42
17.23 读取 csv 中，文本中的列表 [] 其实是字符，把它转回列表 . . . 42
17.24 爆炸函数 . . . 42
17.25 切分字符串 . . . 42

17.26 一个列表去重 43
17.27 希望 a 里面没有 b 中的元素 43
17.28 合并列表中的文字如:['a','b','c']->'abc' 43

18 知识图谱 44
18.1 readme 44
18.2 neo4j 基础操作 44
18.3 python 连接 neo4j 45

19 文字处理方法 47
19.1 去除首尾的特定字符 47

20 os 库用法 48
20.1 readme 48
20.2 获取该文件所在的文件夹 48
20.3 获取文件中的文件名 48
20.4 将多个路径组合返回 48
20.5 返回文件路径的目录 49
20.6 判断路径是否存在，判断文件是否存在，判断目录是否存在 49

21 leetcode 50
21.1 readme 50
21.2 回溯算法:
77 组合
216 组合总和 3
17 电话号码的字母组合
39 组合总和
40 组合总和 2 50
21.3 最优化问题:
53 最大子数组和
84 柱状图中最大的矩形
85 最大矩形
363 矩形区域不超过 K 的最大数值和
40 组合总和 2 51

21.4 动态规划:

10 正则表达式匹配

73 编辑距离

124 二叉树到最大路径和

198 打家劫舍

494 目标和

最长公共子序列 52

21.5 动态规划:

37 解数独

51N 皇后

78 子集

90 子集 2

93 复原 ip 地址

491 递增子序列 52

参考文献 54

1 介绍及使用方法

本文档主要收集大模型算法岗的面试题目以及答案，用于面试各企业的大模型算法岗（目前只包含 NLP 岗位，希望有朝一日该文档能够包含 CV 及多模态方向）。本文档由 blk 撰写并维护，文档使用方法为根据目录中的问题自己思考答案，因为是面试文档，因此不适合初学者学习，初学者可根据提供的问题自行从知乎或其他地方系统学习。

2 transformer 八股文

2.1 attention 计算方式以及参数量

缩放点积注意力:

$$Q = xW_Q$$

$$K = xW_K$$

$$V = xW_V$$

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

2.2 参数量计算

多头后会多输出一层线性层，所以是四个权重矩阵，$W_q, W_k.W_v$ 和 W_o, 权重矩阵的维度为 $[h, h]$, 偏置为 $[h]$, 最后计算为 $4h^2 + 4h$。

MLP 分为两层，第一层是将维度为 $[h, 4h]$，偏置为 $4h$ 第二个维度为 $[4h, h]$，偏置为 h，MLP 层的总维度为 $8h^2 + 5h$。

LN 层的位置参数和尺度参数的维度都是 $[h]$, 两个 LN 的维度为 $4h$

一层 transformer 的总层数为 $12h^2 + 13h$

词嵌入矩阵的维度 Vh，l 层 transformer 的维度就为 $l(12h^2 + 13^h) + Vh$

在实现 Q, K, V 的时候是有偏置 bias 的，Llama 没有用，chatGLM v1 用了，Llama 的 hidden size 为 4096-8092,7B 的模型是 32 个头，每个为 $[128, 4096]$

2.3 模型的训练参数量分析

adam 优化器对应两个状态，一阶动量和二阶动量，训练时候的参数和为模型参数，梯度和优化器状态，在混合精度训练的过程中前向和后向的时候分别以 float16 传递，在优化器更新参数时使用 float32 的优化器状态，梯度和模型参数更新，设总参数为 ϕ, 那么训练的时候的参数总和为 $((2+4)+(2+4)+(4+4))\phi bytes$

2.4 为什么要对 QK 进行 scaling，可以不除以 d_K 吗

假设 q 和 k 为独立同分布的向量，且每个分量服从均值为 0，方差为 1 的分布，即可以证明；可以，只要能够缓解梯度消失的问题就可以，可以参考谷歌 T5 的 Xavier 初始化。

$$E(qk^T) = 0VAR(qk^T) = d_k$$

因此 scaliing 可以让方差和原来相同，防止梯度消失或者爆炸，让模型能够更容易训练。

2.5 self-attention 一定要这样表达吗

不一定，主要能够建模相关性即可，最好能高效运算利用矩阵乘法

2.6 tranformer 为什么用 Layer Norm

使 norm 后的网络的输入数据的分布变得更好，使得数值进入敏感区间，以减缓梯度消失，从而更容易训练。可以证明: 当 $W^* = \lambda W$ 时，$norm(Wx) = norm(w^*x)$

2.7 为什么不用 BN

BN 广泛用于 CV 场景，但是 BN 的缺陷就是他不能用在 NLP 这种句子长度不同的地方，比如把每个样本的第一个字归一化没有意义，而在 cv 里面一个点归一化是由实际意义的，比如黑白两色，原本亮 BN 后也亮。BN 只在训练用，推理没用，推理的时候就只有一个样本。BN 特别吃样本量

2.8 为什么要加一个 position embedding

因为向量之间的内积其实并不能反映位置信息，比如“我打了他”和“他打了我”之间这里面的“我”和“他”的向量内积就是相同的

2.9 Bert 为什么三个 embedding 可以相加

三个相加的 embedding 分别为 token embedding，position embedding，segment embedding。

在实际场景中，叠加是一个很常见的操作，比如声音和图像等就可以对不同的频率用不同的正弦波来叠加表示。一串文本也可以看作时一些时序信号，token，segment，position 对应不同的频率

2.10 transformer 为什么要用三个不一样的 QKV

是为了增加网络的容量和表达能力，如果只是用 x 本身来做也可以，但是表征能力太弱了

2.11 为什么要用多头注意力

进一步增加网络的容量和表达能力，类似于 cv 中的 channel，不同的头可以关注不同的信息，比如有的头关注主语，有的头关注谓语，有的头关注宾语，经过多头之后还是要经过线性层赋予不同头的权重。个人理解是在不同的基空间下训练，类似于强化学习的思想。

2.12 bert 中为什么要用 wordpiece 或者 BPE 这种 subtoken

避免 Out Of vocabulary，防止不在此表里的词出现，传统的 unk 是把未知的词用 <unk> 表示，但是这种方式会损害一部分信息，但是用了 wordpiece 后比如 unhappy 就会变成 un 和 happy, 同时 BPE 本身的语义粒度也不会太大，也不会太小

2.13 为什么 bert 要加一个 cls token

bert 最前面加一个 cls token，最后一层该位置可以代表整个语句的语义表示，从而用于下游任务的分类任务。cls token 和其他的 token 相比可以更加公平，因为他没有明显的语义信息，可以更好的表征语义。

2.14 bert 中用了哪些 mask

掩码语言模型 MLM，使用 mask 来做完形填空；self-attention 中的 mask；下游任务 decoder 中的 mask

2.15 介绍一下 MLM 中的 mask

MLM 可以理解为完形填空，把 85% 的词 mask 掉，其中%80 是 mask，10% 不变，10% 为随机词，目的是对 mask 进行预测。

2.16 bert 中 attention 中 mask 有什么用

不同输入的句子长度不同，但是为了让输出的长度相同，所以要对剩下的部分进行 padding。具体做法是在 softmax 之前复制为-inf，这样 softmax 后的值就是 0 了

2.17 解释 decoder 中的 mask

构造一个下三角矩阵，右上角全是 0，左上角全是 1。目的是为了模拟推理的过程，当前生成的词只能够看到前面的词而不能够看到后面的词，防止信息泄露

2.18 有没有什么降低复杂度但是增加输入长度的

sparse attention，放弃全文的关注，只关注局部的语义组合，相当于在 attention 的基础上再加上了一层 mask

2.19 为什么之前的 char level 和 subword level 的效果差，但是 bert 里面的效果好

主要还是归功于 transformer，transformer 能够使模型很好的利用 char level 和子词的级别来理解语句。之前的模型最多两个 LSTM，非线性路线过于陡峭。

2.20 什么是非线性学习路线，为什么 bert 降低了输入复杂度，为什么 bert 提升了模型复杂度

模型的深度看作 x 轴，模型的复杂度或训练难度看成 y 轴，随着深度的增加，y 的增长可能非常快。

使用 BPE 使得粒度为子词形式，比所有词的词表小

bert 可以把网络搭的非常深，所以提升了模型复杂度

2.21 bert 中为什么使用 warm up 的学习率

训练开始时参数更新过快，防止过拟合或者局部最优，warm up 就是使学习率以线性增加到预设值，使得训练比较平滑

2.22 bert 怎么做到一词多义

根据 self-attention，每个词会关注其他不同的词来得到最后一层的向量表示

2.23 Bert 中的 transformer 和传统的 transformer 有什么区别

bert 使用 transformer 中的 encoder 部分，其中的位置编码变成了可训练的。如果模型参数多，训练数据多，那么变成训练效果就会好，比如 cv 里面的卷积核变成可训练的

3 tokenization

3.1 什么是分词

分词的目的是将输入文本分成一个个词元，保证各个词元拥有相对完整和独立的语义

3.2 总结一下文本的三种分词粒度的优缺点

word：优点是词的边界和含义得到保留，缺点是词表大，稀有词难以训练，oov问题无法处理，无法处理单词形态和词缀的关系；char：优点是词表极小，缺点是无法承载丰富的语义，且序列长度大幅增长；subword：可以较好的平衡词表大小和语义表达能力

3.3 Byte-Pair Encoding(BPE) 怎么构造词典

- 准备足够的训练语料，以及期望的词表大小
- 将单词拆分为字符粒度，并在末尾添加后缀“”，统计单词频率
- 合并方式：统计每一个连续字节对出现的频率，将最高的连续字节合并为新的子词
- 重复第 3 步直到词表到达大小

3.4 WordPiece

4 位置编码

4.1 正余弦位置编码是什么，其具有的性质是什么

$$PE_{(\text{pos},2i)} = \sin\left(\text{pos}/10000^{2i/d_{\text{model}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{\text{model}}}\right)$$

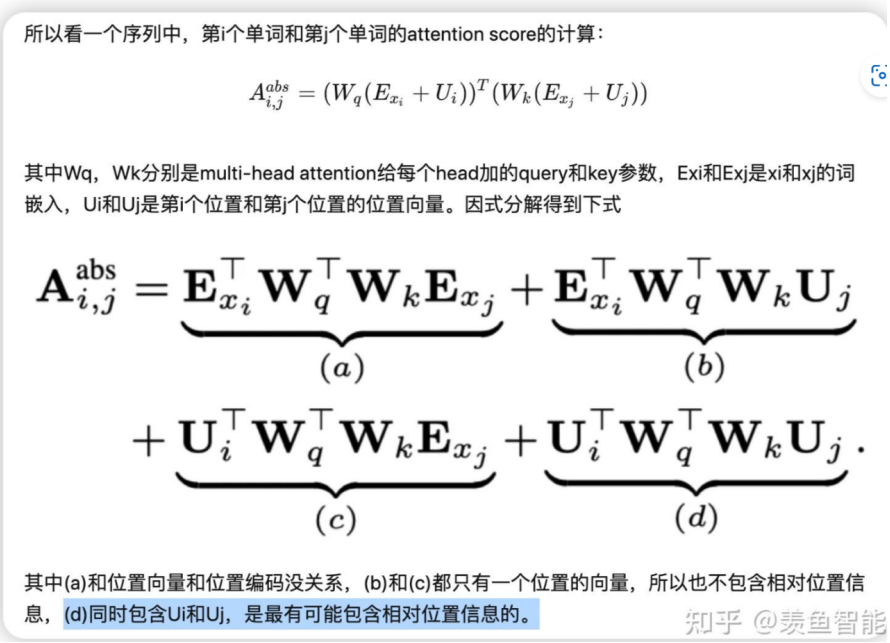
- 具有相对位置的表达能力，两个绝对位置编码的点积只与相对位置 $i-j$ 有关
- 表达相对位置时是无向的
- 远距离衰减性：两个位置编码的点积随位置差的增大而减小，呈现出远距离衰减性，也就是两个 token 距离越远，位置编码的关系越弱

4.2 理想的位置编码有什么特点

- 每个时间步输出唯一的编码
- 任何两个时间步之间的距离在不同的句子中应该一致
- 长度外推性
- 值有界

4.3 传统的正余弦位置编码看起来很完美，为什么后来没人用了

因为正余弦的相对位置表达能力被 attention 的投影矩阵破坏掉了


所以看一个序列中，第i个单词和第j个单词的attention score的计算：

$$A_{i,j}^{abs} = (W_q(E_{x_i} + U_i))^T (W_k(E_{x_j} + U_j))$$

其中Wq，Wk分别是multi-head attention给每个head加的query和key参数，Exi和Exj是xi和xj的词嵌入，Ui和Uj是第i个位置和第j个位置的位置向量。因式分解得到下式

$$\mathbf{A}_{i,j}^{\text{abs}} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(b)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(c)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(d)}.$$

其中(a)和位置向量和位置编码没关系，(b)和(c)都只有一个位置的向量，所以也不包含相对位置信息，(d)同时包含Ui和Uj，是最有可能包含相对位置信息的。

4.4 学习式绝对位置编码 APE

直接将位置编码当做可学习的参数，初始化一个 $[L, D]$ 的矩阵，随着训练过程更新即可。但缺点在于没有长度外推性，一般训练的长度是多少，推理的长度也是多少

4.5 相对位置编码

对相对位置 $i - j$ 进行建模，transformer 中的相对位置表达能力是在 attention 阶段损失的，那么就在这里做改动，把相对位置信息加在 K 和 V 上，并在多头之间共享。

$$e_{ij} = \frac{x_i W_q (x_j W_k + a_{ij}^k)^T}{\sqrt{d_z}}$$
$$z_i = \sum_j \alpha_{ij} (x_j W_v + a_{ij}^v)$$

$$a_{ij}^K = w_{\text{clip}(j-i,k)}^K$$
$$a_{ij}^V = w_{\text{clip}(j-i,k)}^V$$
$$\text{clip}(x, k) = \max(-k, \min(k, x))$$

clip 函数其实是分段函数，在 [-k,k] 为线性，在两侧截断

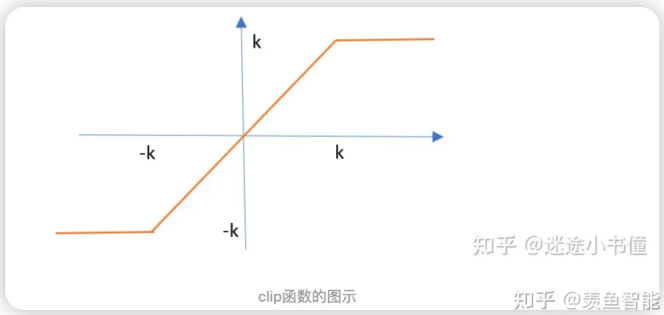


图 4.1 Enter Caption

4.6 XLNet

其值在 K 上引入了相对位置信息，把所有的 U_i 换成了 R_{i-j}; 把 $U_i^T W_q^T$ 换成了两个可以训练的向量 u, v

$$\mathbf{A}_{i,j}^{\text{abs}} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(b)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(c)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(d)}.$$

transformer-XL做了以下修改：

$$\mathbf{A}_{i,j}^{\text{rel}} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(b)} + \underbrace{u^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(c)} + \underbrace{v^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(d)}.$$

4.7 T5 bia 式

q_i 和 k_j 的乘积可以看做四项，content2content、content2position、position2content 和 position2position。如果我们认为输入信息与位置信息应该是独立的，那么就可以直接去掉 2,3 项，第 4 想是一个只依赖 i，j 的标量，我可以直接将它作为参数训练出来

$$\boldsymbol{x}_i \boldsymbol{W}_Q \boldsymbol{W}_K^\top \boldsymbol{x}_j^\top + \boldsymbol{\beta}_{i,j}$$

4.8 DeBERTa 式

将第二三项的位置编码换成相对位置编码，将第四项扔掉

$$Q_c = HW_{q,c}, K_c = HW_{k,c}, V_c = HW_{v,c}, Q_r = PW_{q,r}, K_r = PW_{k,r}$$
$$\tilde{A}_{i,j} = \underbrace{Q_i^c K_j^{c\intercal}}_{\text{(a) content-to-content}} + \underbrace{Q_i^c K_{\delta(i,j)}^{r\intercal}}_{\text{(b) content-to-position}} + \underbrace{K_j^c Q_{\delta(j,i)}^{r\intercal}}_{\text{(c) position-to-content}} \quad (4)$$
$$H_o = \text{softmax}(\frac{\tilde{A}}{\sqrt{3d}})V_c$$

4.9 ALiBi 位置编码

其做法是不加 position embedding，而是添加一个静态的不学习的 bias，其优点是具有良好的外推性，BLOOM 使用这种位置编码

$$\text{softmax}(\mathbf{q}_i\mathbf{K}^\top + m \cdot [-(i-1), ..., -2, -1, 0]),$$

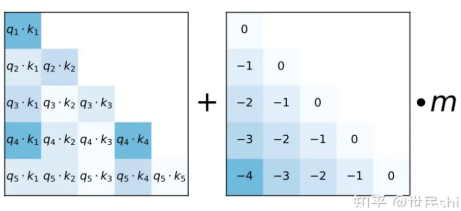


图 4.2 Enter Caption

4.10 混合位置编码

绝对位置编码具有实现简单，计算速度快的特点，而相对位置编码则直接地体现了相对位置信息，如何做到集两者之长呢，则是通过绝对位置编码来实现相对位置编码，比如 transformer 中的正余弦位置编码就实现了这种效果，但是投影矩阵部分损害了其功能。

4.11 ROPE

LlaMa，GLM-130B，PaLM 使用这种编码方式

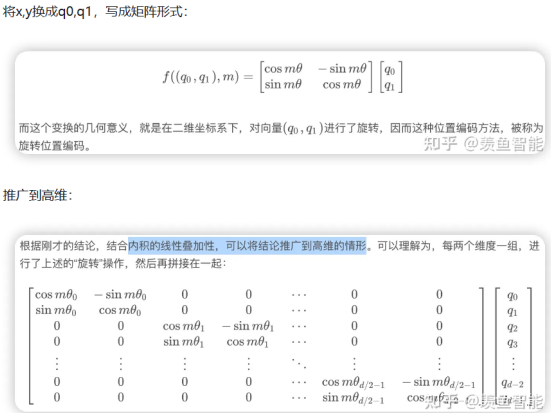


图 4.3 Enter Caption

5 attention 及其变体

5.1 Multi-query attention

默认是 Q 和 K 不用多头，而 V 用多头，ChatGLM2，Palm 用的是这个，减少 kv cache 的大小，减少缓存占用，提升推理速度

5.2 group-query attention

介于 mta 和 mqa 之间，有多个 q 和 k

5.3 flash attention

用分块 softmax 等价替代传统 softmax，省了大量显存，减少了训练时间

6 大模型加速

6.1 KV cache

在生成式模型中，推理阶段时会将输出的 token 和当前的输入 token 拼接作为下一个 step 的输入，因此第 i+1 的计算必然包括了第 i 次计算的结果。KV cache 就是缓存当前可重复利用的结果，下一轮计算时直接读取

6.2 大模型的推理时的参数

top-k，top-k，temperature，repetition penalty，do-sample = True

6.3 介绍下 vLLM

vLLM 运用了全新的注意力算法 paged attention，它将在操作系统中的虚拟内存的思想引入到 LLM 服务中，允许在不连续的空间中储存连续的 k 和 v，能够有效地管理 k 和 v。具体来说，paged attention 将每个序列的 kv 缓存成 block，每个 block 包含固定数量的 tokens 的 k 和 v，在注意力计算地过程中，paged attention 有效地识别这些 blocks。

6.4 vLLM 有哪些优点？

- 最先进的服务吞吐量
- 可以有效地管理 k 和 v
- 动态批处理请求
- 和 hugging face 模型无缝集成
- 支持各种解码算法如：beam search，parallel sampling
- 支持分布式推理

7 大模型面试基础问题

7.1 目前主流的开源模型体系有哪些

7.1.1 encoder decoder 系

encoder 是双向注意力，decoder 是单向注意力。代表模型是 T5 和 BART，最早的 transformer 也是这种类型，优点是对问题的编码理解更充分，更加适用于理解性的 NLP 任务，缺点是训练效率低。

7.1.2 causal decoder 系

从左到右的单向注意力，代表模型为 LLama 系列，严格遵守只有后面的 token 能看到前面的 token，与下游任务一致，因此在自然语言生成任务上表现较好，优点是训练效率高，zero-shot 效果好，容易产生涌现能力。

7.1.3 prefix-decoder 系

prefix 部分是双向注意力，其余部分是单项注意力。代表模型：chat-GLM，chat-GLM2，U-PaLM。优点是 prefix 的 token 能够互相看到，是 encoder-decoder 和 causal decoder 的折中，缺点是训练效率低。

7.2 大模型 LLM 的训练目标是什么

训练目标为极大似然函数，根据已有的词预测下一个词，训练效率来说 prefix decoder < causal decoder，因为 causal 在所以 token 上计算损失，而 prefix decoder 旨在输出上计算损失。

$$L(x) = \sum_{i=1}^{n} logP(x_i|x_{<i})$$

7.3 为什么现在的大模型都是 decoder-only 的结构

在没有任何数据的微调下，decoder-only 的 zero-shot 表现能力最好。而 encoder-decoder 则需要一定标注量的数据做多任务微调才能够激发最佳潜能。然而目前的 LLM 都是在大规模的无监督语料上做自监督学习；就生成式任务而言，引入双向

注意力机制并无实质好处，而之所以 encoder decoder 在某些场景下表现更好，大概是因为它多了近一倍参数，在同等参数，同等推理成本的情况下，decoder-only 就是最好的选择。

7.4 为什么输入序列不能无限长？

transformer 中的自注意力机制中的时间复杂度和空间复杂度都是输入序列的平方，当输入序列变长之后会增大训练量，增加训练难度，从而降低模型的效率。

7.5 介绍下 FP32,FP16,int8，混合精度

- FP32 采用 32 个 bites，4bytes，其中 8bits 表示指数，23bits 表示小数
- FP16 采用 16 个 bites，2bytes，其中 5bits 表示指数，10bits 表示小数
- int8 采用 8 个 bites，1 byte，用 8bits 来表示一个数字
- 混合精度指的是用 fp16 进行乘法和储存，用 fp32 进行加法

7.6 大模型的评测

可以使用数据集进行做题回答，比如 MMLU,HumanEval，Math 等；可以用 GPT4 进行判断，也可以就是双盲由人工打分

7.7 预训练和 SFT 有什么区别？

比如在问答中，预训练同时计算问句和答句的损失，而微调只计算答句的损失

7.8 介绍下 Layer Norm

（每个神经元-均值）/标准差

7.9 介绍下 RMS Norm 均方根 norm

去除了均值的计算

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

$$x = \frac{x}{RMS(x)} \cdot \gamma$$

7.10 介绍下不同的 LN

- post-LN，传统的 transformer 的 LN，LN 在 attention 后面，在 FFN 后面，在残差连接后，训练不稳定，chatGLM 系列
- pre-LN，LN 在 mha 和 FFN 前面，在残差连接前，训练不稳定但是模型效果差，GPT3，LLAMA
- sandwich-LN，LN 在 mha 和 FFN 前后都有

7.11 介绍下激活函数

部分用的 GeLU, 大部分用的 SwiGLU, 如 Llama。chatglm1 是 GeLU, chatGLM2 是 SwiGLu

7.12 什么是稀疏向量和稠密向量

- 密集向量：也称为稠密向量，使用普通的数组来存储向量的值
- 稀疏向量：通常用两部分表示：一部分是索引向量，另一部分是值向量。
- 举例：向量 (1.0,0.0,1.0,3.0)
- 密集格式：表示为 [1.0,0.0,1.0,3.0]
- 稀疏格式表示为 (4,[0,2,3],[1.0,1.0,3.0])
- 第一个 4 表示向量的长度 (元素个数)，[0,2,3] 就是 indices 数组；[1.0,1.0,3.0] 是 values 数组，表示向量 0 的位置的值是 1.0，2 的位置的值是 1.0, 而 3 的位置的值是 3.0, 其他的位置都是 0

7.13 a

* 1 * 2

8 评价指标和损失函数

8.1 介绍一下 bleu 以及 rouge

在机器翻译中，bleu 和 rouge 是两个常见的评价指标，bleu 根据的是精确率，而 rouge 根据的是召回率，bleu 通过计算 N-gram 的匹配程度，来评估机器翻译的精确度。rouge 是评估文本摘要的指标，其关注是否生成的摘要是否捕捉到了参考摘要的信息，其通过 N-gram 的共现程度来衡量召回率。

8.2 介绍下微调的损失函数

微调的每一步会生成一个概率分布表，用来看模型的预测和实际答案有多接近，最终每个时间步的损失会加在一起来求平均作为最终损失，交叉熵损失函数的公式：

$$L = -\sum_{i=1}^{N} y_i \log(p_i)$$

8.3 信息墒的公式

$$\sum -p_i log_2 p_i$$

9 PEFT

9.1 介绍下 fine-tuning

更新全部参数以适配领域数据

9.2 为什么需要 peft？

针对特定的下游任务时，全参数微调太费时间和现存，而只调后面的几层效果又不好

9.3 peft 有什么优点？

peft 可以大幅度缓解显存和时间，同时也可以减缓全参微调导致的大模型的遗忘问题

9.4 peft 对比

p-tuning v2，lora 都是综合效果很不错的 peft 方法，如果现存不足可以使用 Qlora，简单的任务可以考虑 p-tuning 和 prompt tuning

9.5 adapter tuning

在 transformer 中的 mha 和前馈层后面加入一个 adapter，先将维度从 d 维降到 m 维，再从 m 维增到 d 维，一般来说 d»m。

9.6 prompting 提示学习

9.6.1 什么是提示学习？

prompt 提示上下文和任务相关信息，比如问答中可以是相关场景或问题的描述，情感分析可以是：这段话的情感是：

9.7 介绍下 prompt-tuning？

在输入层随机初始化 virtual tokens 向量，训练时只更新这些向量的参数，自动学习 prompt

9.8 prompt-tuning 的优缺点？

- 仅在输入层添加 prompt 向量，并且不需要引入 MLP 来解决难训练的问题
- 随着预训练模型参数量的增加，p-tuning 能够接近于全参数微调的效果
- 训练难度加大了，省了显存但是不能省时间
- 多个 prompt token 之间独立，可能会影响效果
- 在 NLU 任务上，如果预训练模型规模正常则表现效果较差

9.9 介绍下 prefix tuning

在前面加入一段虚拟的 prompt，在输入 token 之前构造一段和任务相关的 virtual tokens 作为 Prefix，前缀完全由自由参数组成，与真实的 token 不对应，训练时只更新 prefix 部分的参数,prefix 层前要加一个 MLP 层，防止直接更新 prefix 的参数导致不稳定。

9.10 介绍下 prefix tuning 的优缺点

- prefix-tuning 相比于真实的 tokens 能够学习隐式的 prompts
- 基于前缀的架构可以在一个批次中处理多个任务
- 占用序列长度，有额外的计算开销
- 每一层都增加了 prefix 的参数，改动较大
- 并且只能在开头添加 virtual token，只适用于 NLG 任务

9.11 p-tuning 与 prefix tuning 的对比

- prompt tuning 可以看作是 p-tuning 的简化版本
- prefix-tuning 只适合于 NLG 任务，而 prompt-tuning 适合所有任务
- prompt-tuning 可以在任何位置加 virtual tokens，而 prefix-tuning 可以在任何位置加
- prefix-tuning 每一层都加，prompt-tuning 只加在输入层

9.12 介绍下 p-tuning

将一个 bi-LSTM 和两层的 MLP 作为 prompt encoder，建立伪 token 的相互依赖，能够提供更好的初始化

9.13 p-tuning 的优缺点

- 将一个 bi-LSTM 和两层的 MLP 作为 prompt encoder，建立伪 token 的相互依赖，能够提供更好的初始化
- 复杂度增加，因为直观上看起来太不像 prompt 了
- 在一些复杂的 NLU 任务上表现很差，同时预训练模型规模不能太小，否则效果就会很差

9.14 为什么要有 p-tuning v2

为了让 p-tuning 能够在任何规模的预训练模型和任何下游任务都取得接近于 fine-tuning 的效果

介绍下 p-tuning v2

- 在每一层都加入了可微调的 prompt tokens
- 去除了重编码器，如 p-tuning 中的 LSTM
- 针对不同任务采用不同的 prompt 长度
- 抛弃了传统的 verbalizer，回归到了传统的 cls 和 token label 分类样式

p-tuning v2 的优缺点

- 在每一层都加入了可微调的 prompt tokens，可训练参数量从 0.01% 增加到 0.1%-0.3%
- 在小规模的预训练模型上效果也足够好
- 针对不同下游任务都有较好的表现
- 抛弃了传统的 verbalizer，回归到了传统的 cls 和 token label 分类样式，弱化了 prompt 的味道

9.15 介绍下 LoRA

- 在原来的矩阵旁边增加一个旁路矩阵 AB，先降维，再增维
- 训练的时候只训练 AB 矩阵
- 训练开始时，A 高斯初始化，B 全为 0，保证开始时 AB 为 0

- 推理时把原来的矩阵 W 加上 AB，可以保证不引入额外推理延迟
- 可插拔式替换 AB，可用于多任务切换

9.16 介绍下 Qlora

将预训练模型量化为 4bit，可以有效的降低显存需求

9.17 介绍下 Adalora

根据重要性评分动态分配参数预算给权重矩阵，重要的矩阵给高秩，不那么重要的给低秩

9.18 lora 的优点

- 简单直观，效果好
- 推理不引入额外的计算量
- 和其他 peft 方法正交，可以有效组合
- 一个中心模型搭配多个下游任务，可以减少参数存储

9.19 lora 中 rank 怎么选取？

作者对比了 1-64，发现在 4-8 是最好的，再提升效果没有那么明显

9.20 为什么大模型那么耗显存？

因为有很多个 QKV，还要进行 kv 缓存

9.21 什么是指令微调 instructing tuning

10 RAG

10.1 为什么要有 RAG? 或者为什么 LLMs 需要外挂向量知识库？

- 大语言模型具有幻觉问题，可能会出现一本正经的胡说八道的现象
- 大语言模型越大，训练规模和周期也就越大，很多时效性的问题往往难以得到解答，比如说：给我推荐几部最新的电影

10.2 什么是 RAG

RAG(Retrieval Augmented Generation)，检索增强生成技术，指的是在回答问题的时候，先从大量的文档中检索出相关的信息，然后基于这些信息回答问题，提高问题的准确率。

10.3 RAG 的对话思路流程是什么？

- 加载文件
- 读取文本
- 文本分割
- 文本向量化
- 问句向量化
- 在向量化后的文本匹配出与当前问句最相似的 K 个
- 将匹配的文本一起当作 prompt
- 提交给 LLM

10.4 RAG 的核心技术是什么？

是 embedding，将知识库中的内容通过 embedding 存入向量库，用户的提问也会通过 embedding，之后通过向量相似性算法进行匹配。

10.5 RAG 的评估指标

RAG 的评估方式主要有 Ragas 框架实现，需要提供：question，answer，context，grondtruth。

- 忠实度 faithfulness: 衡量生成的答案与上下文的一致性，根据 answer 和 context 计算得出
- 答案相关性 Answer relevancy: 评估 answer 和 question 的相关程度，通过多次提问和 ground truth 计算余弦相似度
- 上下文精度 context precision：评估上下文出现过的 ground truth 在检索出来的排名是否靠前
- 上下文召回率 context recall：评估检索到的 context 与 ground truth 的一致程度
- 上下文相关性 context relevant：评估问题和检索出来的上下文的相关程度

10.6 除了余弦距离还了解过其他的距离吗？

可以表示向量相似度的有，欧式距离，曼哈顿距离

10.7 了解过对比学习吗？介绍下 simcse

对比学习通常由三个构成，两个为相同的正样本，一个为负样本，使得相同样本的表示更接近，不同样本的表示更远。

11 RLHF

11.1 知乎专栏 https://zhuanlan.zhihu.com/p/624589622

11.2 介绍下 LLM 的经典预训练 pipline

- 在预训练阶段，模型会通过大量无标注数据学习通用知识
- 使用 SFT 以更好地遵循特定指令
- 使用对齐技术使 LLM 可以更有用且更安全地相应用户提示

11.3 预训练和 SFT 的异同点

- 训练都是通过已有的词预测下一个词
- 预训练数据很大，SFT 则小很多
- 计算损失时预训练需将问句计算，而 SFT 只计算答句，且 SFT 需要人工标注

11.4 什么是对齐

通过微调的方式，将模型与人类的价值观和偏好进行对齐

11.5 介绍一下 RLHF 的流程

- 在预训练模型的基础上进行 SFT 微调
- 在 SFT 的基础上训练一个 reward model
- 基于 RM 使用 PPO 算法微调

11.6 怎么训练 reward model

对于每个 prompt，让 LLM 输出 4-9 条的回复，并由标注人员对这些回复进行打分排序

11.7 介绍一下 PPO 算法

PPO 算法的过程分为采样、反馈和学习

- 采样是指模型根据 prompt 生成回复的过程，采样有两个模型，一个是 Actor，一个是 Critic，一个负责决策，一个负责总结得失

11.8 RLHF 的缺点

- 人工产生的数据集成本较高，难以量产
- SFT—>RM->PPO 周期较长，更新迭代较慢
- PPO 两个训练，两个推理，比较耗费资源

11.9 如何解决人工产生数据集成本较高的问题

RRHF，使用 chatGPT 或者是 GPT4 等当前的语言模型来产生回复，然后再进行得分排名来进行人类偏好

11.10 介绍下 RLHF 里面的四个模型

- Actor 模型是需要微调的大模型，用来给 prompt 生成回答
- ref-model 是 Actor model 的复制版，冻结参数，用 kl 散度防止 Actor 模型灾难性遗忘
- reward 模型用来评估及时收益
- critic model 用来预测未来的收益，最终把两个模型的收益加起来来更新参数

12 deepspeed

12.1 多卡训练的目的是什么

多卡是为了训练更大的模型，已经训练速度更快

12.2 介绍下 deepspeed

是一个分布式训练框架，能够支持更大规模的模型以及提供更多的优化策略和工具比如 ZeRO 和 Offload 等

12.3 介绍一下数据并行

将数据集分为多份，每张 GPU 分配到不同的数据进行训练

12.4 介绍一下流水线并行

把模型分为不同的层，每一层放到一块 GPU 上

12.5 介绍下张量并行

层内划分，切分一个独立的层划分到不同的 GPU 上，如对于一个简单的矩阵乘法 $Y = XA$，张量并行分为行并行和列并行

12.6 介绍下 3D 并行

就是数据并行，流水线并行，张量并行同时是用哪个

12.7 介绍一下 Gpipi 流水线并行

普通的流水线并行 GPU 利用度不够，且中间结果占用大量内存。解决方法：

- 切分 micro-batch，在模型并行的基础上，进一步引入数据并行的方法，在原先的数据再划分为 micro-batch
- rematerialization，也称 active checkpoint：几乎不存中间结果，等到 backward

的时候，再重新算一遍 forward，用时间换空间

12.8 deepspeed 下的 ZeRO 的三个 stage

- stage1: 把优化器状态 (optimizer states) 分配到每个 GPU 下
- stage2: 把优化器状态 (optimizer states) 和梯度 (gradients) 分配到每个 GPU 下
- stage3: 把优化器状态 (optimizer states) 和梯度 (gradients) 和模型参数 (parameters) 分配到每个 GPU 下

12.9 ZeRO 的核心思想是什么

计算的过程中分片，即每张卡只存 1/N 的模型状态量，混合精度训练，模型参数 (fp16), 模型梯度 (fp16),Adam 状态 (fp32 的模型参数备份，fp32 的一届动量，fp32 的二阶动量)

12.10 什么是 offload

offload 指将数据、梯度、优化器状态下沉到 CPU 内存上

13 embedding 模型

13.1 embedding 模型的作用是什么？

将输入的句子映射为向量，用于检索、分类、聚类或者语义匹配

13.2 BERT-avg

使用预训练模型做相似度匹配，使用 CLS token 最后一层作为 embedding；或者取最后一层做池化，分类任务 max 最好，匹配任务 mean 效果更好，对 embedding 后的两个句子计算余弦相似度

13.3 SBERT

使用双塔模型，也称孪生网络，把很多个问题 embedding 的结果存起来，来一个新的问题只需要过一次 embedding 就好。预测时使用 pooling 来固定维度的句子 embedding，损失函数为三元损失函数，要求包含两个正例和一个负例

13.4 SimCSE

运用对比学习的方法，使得语义相近的句子在空间中相近，语义不同的相互远离，无监督学习由于 dropout 的原因，可将一个句子 embedding 两次，将其余句子作为反例，监督学习的话就是将相同意思的句子做正例，不同的句子做反例

13.5 BGE

BGE 模型参数很小，但是他在语义检索的精度和整体表征能力都很出色。BGE 的训练数据规模很大并且多样化;BGE 使用对比学习的思想，同时大大增加反例的数量，BGE 使用指令微调，指令是为这个句子生成表示以检索文章

14 面试中的程序题

14.1 python 中的 *args,**args 的用法

*args,**kwargs 主要用于函数定义, 你可以将不定数量的参数传递给某个函数。

- *args：传入的参数个数未知，且不需要知道参数名称时
- **kwargs：将不定长度的键值对作为参数传递给一个函数

14.2 介绍下 python 的魔术方法

魔术方法是 python 内置方法，格式名为：“__ 方法名 __”：相关链接

- __init__(): 对象初始化
- __new__(): 对象初始化方法
- __str__(): 打印一个对象的时候，默认调用
- __call__(): 对象当做函数执行时被默认自动调用
- __len__(): 用来计算对象的长度

14.3 python 里面的深拷贝和浅拷贝

python 中，拷贝一个对象可以通过赋值、浅拷贝和深拷贝三种方式来实现：相应知乎链接

- 赋值操作：讲一个变量赋给另一个变量时，实际并没有创建新的对象，而是创建了原对象的一个引用
- 浅拷贝 : 会创建一个新的对象, 但是它只是复制了一级对象的引用, 对于对象中包含的可变类型的子对象, 仍共享一个引用, 比如 a=[1,2[3,4]],b=copy.copy(a), 改变 b 中的 3，a 中的 3 则也改变。
- 深拷贝：创建一个新的独立的对象，并且递归的复制所有层级的对象。如 a=[1,2[3,4]],b=copy.copy(a), 改变 b 中的 3，a 中的 3 不会发生改变。

15 Linux 中常见命令

15.1 readme

下文中如语法：mkdir [-p] Linux 路径，有 [] 表示里面参数可选可不选，没有 [] 表示参数必选，如：mkdir /root/abc

15.2 cd:change direction

- cd: 切换目录
- cd .. 返回上一级目录
- 绝对路径：凡是以/开头的路径一定是绝对路径，./ ： 当前目录
- 相对路径：
  - ./ : 当前目录
  - ../ : 上一级目录
- pwd: 显示当前目录

15.3 ls

- ls [-a -l -h] [Linux 路径]
  - ls -a: 查看当前目录下的所有文件
  - ls -l: 以列表形式展示更多内容
  - ls -h: 列出文件的大小，-h 必须搭配-l 一起使用
  - ls -al: 显示文件的详细信息
  - ls -alrt: 按时间显示文件 (l 表示详细列表，r 表示反向排序，t 表示按时间排序)
- cd .. 返回上一级目录
- 绝对路径：以根目录做起点，描述路径的方式，路径以/开头
- 相对路径：以当前目录做起点，描述路径的方式，路径不需以/开头
  - ./ : 当前目录
  - ../ : 上一级目录
  - ~ 表示用户的 HOME 目录，比如：cd ~ 或 cd ~/Desktop

15.4 mkdir：Make Directoty

- mkdir [-p] Linux 路径
  - mkdir -p Linux 路径:-p 表示自动创建不存在的父目录，如 home 文件夹里没有任何文件，适用于创建连续多层级的目录，mkdir -p /home/abc/abc1，自动在 home 文件夹建立 abc 文件，在 abc 文件夹建立 abc1 文件

15.5 touch

- touch [Linux 路径]: 用于创建文件，如 touch test.txt, 无 [Linux 路径] 创建在当前文件夹

15.6 cat,more: 查看文件内容

- cat Linux 路径：查看文件内容
- more Linux 路径：可用空格翻页查看，使用 q 退出查看

15.7 cp:copy

- copy [-r] 参数 1 参数 2: 用于复制文件或文件夹
  - -r 用于复制文件夹的时候加入
  - 参数 1：Linux 路径，表示被复制的文件或文件夹
  - 参数 2：Linux 路径，表示要复制去的地方

15.8 mv:move

- mv 参数 1 参数 2: 用于移动文件或文件夹
  - 参数 1：Linux 路径，表示被移动的文件或文件夹
  - 参数 2：Linux 路径，表示要移动去的地方，如果目标不存在，则会对其改名，确保目标存在

15.9 rm:remove

- rm [-r -f] 参数 1 参数 2... 参数 N: 用于移动文件或文件夹
  - :-r 表示删除文件夹
  - :-f 表示强制删除，一般用户用不到

  - 参数 1 参数 2... 参数 N：表示要删除的文件或文件夹路径，按照空格隔开

15.10 通配符 *

- rm 命令支持通配符 *，用来做模糊匹配
  - test*：表示匹配任何以 test 开头的内容
  - *test：表示匹配任何以 test 结尾的内容
  - *test*：表示匹配任何包含 test 的内容

15.11 which

- 通过 which 命令，可以查看所使用的一系列命令的程序文件存放在哪里
  - 输入：which cd，输出：/usr/bin/cd

15.12 find

- find 起始路径 -name “被查找文件名”：可以通过 find 命令搜索指定文件
  - 如：find / -name ” 被查找文件名”
  - 可以使用 * 通配符
- find 起始路径 -size +|-n[kMG]: 按文件大小查找文件
  - +,-表示大于和小于
  - n 表示大小
  - kMG 表示单位，k 表示 kb，M 表示 MB，G 表示 GB
  - 例：查找小于 10KB 的文件：find / -size -10k
  - 查找大于 100MB 的文件：find / -size +100M

15.13 linux 里 docker-compose 安装

- 前往官网下载对应的版本
- 对应超链接
- uname -s 看自己是什么系统
- uname -m 看自己是多少位操作系统
- 安装完后将文件放到/usr/local/bin/目录下
- 赋予可执行权限 sudo chmod +x /usr/local/bin/docker-compose

- 创建符号连接 sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
- 验证是否安装成功 docker-compose –version

15.14 grep：用于搜索文本

- grep [options] pattern [file...]
  - -i：忽略大小写
  - -n：显示匹配行的行号
  - -v：反转匹配，显示未匹配的行
  - -r：递归搜索目录中的文件
  - 例如：grep ”hello” example.txt
  - ‘-E‘ 选项用于在 ‘grep‘ 中启用扩展的正则表达式语法。当你使用 ‘-E‘ 时，‘grep‘ 将支持更多的正则表达式语法，包括使用 ‘+‘、‘¿、‘|‘ 等符号来表示重复、可选内容和逻辑或。
  - ‘-C‘ 选项在 ‘grep‘ 中用于指定匹配行的上下文行数。它允许你在输出中显示匹配行的上下文，以提供更多关于匹配的上下文信息。
  - 最常用 grep -ir ’ 里面填一个字符串’

16 python 处理 pdf 文件

16.1 readme

摘选自：

- 全面指南———用 python 提取 PDF 中各类文本内容的方法：相关链接
- Python 个人学习笔记 PyPDF2 库——PDF 操作:相关链接

安装相应的库

```
pip install PyPDF2 #从存储路径读取PDF文件
pip install pdfminer.six #执行布局分析并从PDF中提取文本和格式。(.six版本的库是支持Python 3的版本)
pip install pdfplumber #识别PDF页面中的table并从中提取信息。
pip install Pillow 读取PNG图像。
```

加载相应的库

```
# 读取PDF
import PyPDF2
# 分析PDF的layout，提取文本
from pdfminer.high_level import extract_pages, extract_text
from pdfminer.layout import LTTextContainer, LTChar, LTRect, LTFigure
# 从PDF的表格中提取文本
import pdfplumber
# 从PDF中提取图片
from PIL import Image
#from pdf2image import convert_from_path
# 运行OCR从图片中提取文本
import pytesseract
# 清除过程中的各种过程文件
import os
```

16.2 提取文本

```
import PyPDF2,os

os.chdir("/chj/blk") #绝对路径
pdf_path='历届IMO试题(1-44届).pdf' #文件名称

pdf_reader=PyPDF2.PdfReader(pdf_path)  #用PdfReader打开PDF文件
text=pdf_reader.pages[0].extract_text()  #获取第1页的文本
```

16.3 PDF 拆分

```python
import PyPDF2,os

os.chdir("/chj/heizijian")
pdf_path='BGE_M3.pdf'

#拆分PDF
def PDF_split(pdf_path,output_file,page_split_list):
    '''
    :param pdf_path: 需要拆分的PDF文件路径
    :param output_file: 拆分后的PDF文件的输出路径文件夹
    :param page_split_list: 需要拆分的页码的列表
    :return:
    '''
    #打开PDF文件
    pdf_reader=PyPDF2.PdfReader(pdf_path)
    #将最后一页加入到页码列表，用于拆分
    page_split_list.append(len(pdf_reader.pages))
    #遍历页码列表，拆分
    end_page=0
    for page_split in page_split_list:
        start_page=end_page
        end_page=page_split
        #创建PdfWriter
        pdf_writer=PyPDF2.PdfWriter()
        #遍历起止页之间的页码
        for page_num in range(start_page,end_page):
            page=pdf_reader.pages[page_num]
            #add_page()将页面添加到PdfWriter
            pdf_writer.add_page(page)
        #输出（保存）PdfWriter到指定文件夹，拆分后的文件命名为 原文件名 + 下划线 + 拆分页码
        pdf_writer.write(f'{output_file}\\{os.path.splitext(pdf_path)[0]}_{end_page}.pdf')
#切分为1，2-3，4-15，16
PDF_split(pdf_path,"/chj/heizijian/",[1,3,15])
```

16.4 PDF 合并

```python
import PyPDF2,os,glob,natsort

os.chdir("/chj/heizijian") #>-<
pdf_path="/chj/heizijian"

```

```
#合并PDF
def PDF_join(files_path, output_path):
    '''
    :param files_path: 需要合并的PDF文件所在的路径（文件夹）
    :param output_path: 输出路径
    :return:
    '''
    #获取路径中的PDF文件路径，并进行自然排序
    pdf_files=natsort.natsorted(glob.glob(f'{files_path}/*.pdf'))
    #创建PdfWriter
    pdf_writer = PyPDF2.PdfWriter()
    #遍历文件
    for pdf_file in pdf_files:
        #读取每个PDF文件
        pdf_reader = PyPDF2.PdfReader(pdf_file)
        #将页面添加到PdfWriter
        for page in pdf_reader.pages:
            pdf_writer.add_page(page)
    #输出
    pdf_writer.write(output_path)
# 会合并所有的带后缀为.pdf的文件，最好把他们放在一个文件夹并修改 >-< 处
PDF_join(pdf_path,'output.pdf')
```

16.5 创建虚拟环境以及激活虚拟环境

```
conda create -n py310_chat python=3.10
source activate py310_chat
```

16.6 Jupyter 无法找到 Conda 创建的虚拟环境

相关链接

```
conda activate myenv #myenv是虚拟环境名称
conda install jupyter
python -m ipykernel install --user --name=myenv
```

17 pandas 常见操作

17.1 readme

- 推荐收藏！3.5 万字图解 Pandas：相关链接

17.2 创建 Dataframe

```
data={'A':['x','y','z'],'B':[1000,2000,3000],'C':[10,20,30]}
df=pd.DataFrame(data,index=['a','b','c'])
df
```

- 数据，位于表格正中间的 9 个数据就是 DataFrame 的数据部分。
- 索引，最左边的 a、b、c 是索引，代表每一行数据的标识。这里的索引是显式指定的。如果没有指定，会自动生成从 0 开始的数字索引。
- 列标签，表头的 A、B、C 就是标签部分，代表了每一列的名称。

17.3 获取 Dataframe df 数据的示例代码

```
df.values
```

输出：

```
array([['x', 1000, 10],
       ['y', 2000, 20],
       ['z', 3000, 30]], dtype=object)
```

17.4 获取 Dataframe 行、列索引的示例代码

```
df.index
df.columns
```

输出：

```
Index(['a', 'b', 'c'], dtype='object')
Index(['A', 'B', 'C'], dtype='object')
```

创建时可以指定行列标签：

```
df=pd.DataFrame(data,columns=['C','B','A'],index=['a','b','c'])
#columns是列名，index是行名
```

```
df
```

17.5 删除一列

```
df['D'] = 10 #赋值给一列全是10
del df['D'] #删除列名为'D'的列
df
```

17.6 添加行

```
df.loc['e']=['new2',5000,50]
df
```

17.7 横向合并两个 Dataframe

```
df2 = pd.DataFrame(data={'x1':['A','B','C'],'x2' : [1,2,3]})
df3 = pd.DataFrame(data={'x1':['A','B','D'],'x3' : ['T','F','T']})

pd.merge(df2,df3,how = 'left',on = 'x1')
pd.merge(df2,df3,how = 'right',on = 'x1')
pd.merge(df2,df3,how = 'inner',on = 'x1')
pd.merge(df2,df3,how = 'outer',on = 'x1')
```

17.8 按列求和，求列均值，求列累加和以及一键生成多种统计数据，看 df 的信息，看每一列的值的分布次数，df 的维度

```
df=pd.DataFrame(np.random.randn(8,4),columns=list('ABCD'))
df.sum()
df.mean()
df.var()
df.std() #标准差
df.apply() #应用一个function
df.cumsum() #求列累加
df.describe() #一键生成多种统计数据
df.info() #看df多少行，多少列，每列的数据类型等等
df.min()
df.max()
df['D'].value_counts() #看D这一列的值的分布，即每一个值出现了多少次
df.shape #df的维度
```

17.9 提取某一列、某一行、某几行某几列

```
df['A'] #提取某一列
df.A #提取某一列
df.loc['a'] #提取某一行
df.loc['a':'c',['A','B']] #提取a到c行，A，B列
df.iloc[[0,1,2],0:2] #提取0，1，2行，0，1列
```

17.10 根据某一列的值的范围进行筛选

```
df.A>0
# df['A']>0
df[df.A>0] #选取df中A>0的行
```

17.11 根据某一列的值的范围进行筛选

```
df.A>0
# df['A']>0
df[df.A>0] #选取df中A>0的行
```

17.12 改变一列的值，改变某行某列的值

```
df.loc[:,'D']=1 #把名称为D的列全变成1
df.loc['a','D'] #提取‘a’行，‘D’列
df.loc[:,'D'] = np.array([1,2,3,4,5,6,7,8]) #把D行变成1-8
```

17.13 混合索引

```
df.loc[:,'D']=1 #把名称为D的列全变成1
df.loc['a','D'] #提取‘a’行，‘D’列
df.loc[:,'D'] = np.array([1,2,3,4,5,6,7,8]) #把D行变成1-8
```

17.14 pandas 读取 excel，存储 excel

```
df = pd.read_csv("data/轮胎QA结果.csv")
df = pd.read_excel('data/轮胎QA结果.xlsx')
df.to_csv('output.csv', encoding = "utf-8-sig",index=False)
```

17.15 数据清洗

17.16 删除重复项

```
df.loc[:,'D'] = 1
df.loc['i'] ={'A':-0.386688,'B':0.051907,'C':-0.486462,'D':1.0}
df2 = df
df2 = df2.drop_duplicates(subset = ['D']) #删除D列重复的行

#删除重复的行
data={'A':['x','x','x'],'B':[1000,1000,3000],'C':[10,10,30]}
df3=pd.DataFrame(data,index=['a','b','c'])
df3 = df3.drop_duplicates(subset = ['A','C'])
```

17.17 重命名列名，删除值为空的行，某一行分列

```
df.columns = ['D','C','B','A'] #重命名列名
df = df.loc[~df['D'].isna()] #删除D列为空值的行

#某一列分列
data={'A':['x 1','y 2','z 3'],'B':[1000,2000,3000],'C':[10,20,30]}
df=pd.DataFrame(data,index=['a','b','c'])
df['空格前面'] = df['A'].str.split(' ').str[0]
df['空格后面'] = df['A'].str.split(' ').str[1]
```

17.18 groupby 分组操作

```
data={'A':['x','y','z','t'],'B':[1000,2000,3000,1000],'C':[10,20,30,40]}
df= pd.DataFrame(data,index=['a','b','c','d'])
df = df.groupby(['B'])[['C']].mean()
#将B列相同的值分为一组，对C列求和
```

17.19 排序

```
df.sort_values('B')   # 按 B 这一列排序
df.sort_values(['A','B']) #按多列排序
```

17.20 快速索引

```
df.sort_values('B')    # 按 B 这一列排序
df.sort_values(['A','B']) #按多列排序
```

17.21 映射到二维平面

```
## 映射到二维平面
df = pd.DataFrame({'client':['John','John','Silvia','Silvia'],'product':['bananas','
    oranges','bananas','oranges'],'quantity':[5,3,4,2]})
df = df.pivot(index = 'client',columns = 'product',values = 'quantity')
```

17.22 提取列表中某一列（比如'D' 列）包含特定字符的列表

```
## 提取'B'列中含'd'字符的列表
df = pd.DataFrame({'A':['a','b','c'],'B':['dc','ec','fc']})
df[df['B'].str.contains('d')]
```

17.23 读取 csv 中，文本中的列表 [] 其实是字符，把它转回列表

```
df['A'] = df['A'].apply(eval)
```

17.24 爆炸函数

```
## 爆炸函数，把B列中列表变成竖的
a = pd.DataFrame({'A':['a','b','c'],'B':[[1,2,3],[4,5],[6]]})
a.explode('B')
```

17.25 切分字符串

```
a = 'Q：今天星期几？A：今天星期五'
a.split("A：", 1)
# output:['Q：今天星期几？', '今天星期五'],即分成'A：'左右两部分，取右部分为：
a.split("A：", 1)[1]

# 取出1、2、3、后面的字符
a = '1、语文2、数学3、英语'
a = a.split("1、",1)[1]
word1 = a.split("2、",1)[0]
a = a.split("2、",1)[1]
word2 = a.split("3、",1)[0]
```

```
a = a.split("3、",1)[1]
word3 = a
print(word1,word2,word3)
```

17.26 一个列表去重

```
## 本质是先转成集合再转成列表
a = [1,2,3,4,4,5,6,7,3,1]
a = list(set(a))
a
```

17.27 希望 a 里面没有 b 中的元素

```
a = [1,2,3,4,4,5,6,7,3,1]
b = [1,2,3,4,10,12]
a = [i for i in a if i in b]
```

17.28 合并列表中的文字如:[’a’,’b’,’c’]->’abc’

```
a = ['a','b','c']
''.join(a)
```

18 知识图谱

18.1 readme

参考网站

- GQL 里标签可以理解为职业，比如演员、歌手，属性可以理解为特征，比如身高、体重
- neo4j 基础语法（一）：链接

18.2 neo4j 基础操作

18.2.1 创建一个节点

```
#创建一个带时间的节点
create(n:Person{id: 1,name:'盘古',birthday:date("2000-08-18")})return n
#创建一个有标签的节点
create(a:Pearson)
#创建一个叫刘备的人的节点
create(c:Person{name:'刘备'})
#创建了一个name为阿凡达的节点，这个节点有两个标签，分别是Person和Moive
create(d:Person:Moive{name:'阿凡达'})
#创建了一个标签为Person的节点，这个节点有name和country两个属性。
create(e:Person{name:"诸葛亮",country:'蜀汉'})
```

18.2.2 创建关系

```
#给两个新节点创建无属性关系
create(f:Person{name:'曹雪芹'})-[r:write]->(g:Book{name:'红楼梦'})
#给两个新节点创建有属性关系
create(m:Person{name:'卡梅隆'})-[r:make{cost:'28亿RMB'}]->(n:Moive{name:'阿凡达'})
#给已有节点创建无属性关系
match(n:Person{name:'刘备'})
match(m:Person{name:'诸葛亮'})
create(n)-[r:headOf]->(m)
```

18.2.3 删除节点

```
#根据id来删除
match(n:Person) where id(n)=1
delete n
#根据属性来删除，比如'name'属性，但是必须是孤立节点，即该节点没有任何关系
match(n:Person{name:'阿凡达'})
delete n
#先删除节点关系，再删除节点
match(n:Person{name:'诸葛亮'})
match(n)-[r:headOf]-(m)
delete r,n
```

18.2.4 删除属性，标签

```
#删除属性
MATCH (n:Person{name:'曹雪芹'})
remove n.name
return n
#删除标签
match(n:Person{name:'卡梅隆'})
remove n:Person
return n
```

18.2.5 添加属性，更改属性值：两者操作一样

```
#添加属性
match (n:Person{name:'刘备'})
set n.title='蜀汉皇帝'
return n
#更改属性值
match (n:Person{name:'诸葛亮'})
set n.country='三国蜀汉'
return n
```

18.3 python 连接 neo4j

18.3.1 导入节点

```
from py2neo import Node, Relationship, Graph, NodeMatcher, RelationshipMatcher#导入我们需
    要的头文件
import pandas
import os
```

```
test_graph = Graph('http://localhost:7474',auth=('neo4j','neo4j'), name = 'neo4j')

node_1 = Node('英雄','帅哥',name = '张无忌')
node_2 = Node('英雄',name = '杨逍',武力值='100')
node_3 = Node('派别',name = '明教')

test_graph.create(node_1)
test_graph.create(node_2)
test_graph.create(node_3)
```

18.3.2 输出标签

```
#输出标签
test_graph.schema.node_labels
#输出关系
graph.schema.relationship_types
```

18.3.3 创建关系

```
a = Node("Person",name="Alice")
b = Node("Person",name="Bob")
r = Relationship(a,"KNOWS",b)
test_graph.create(r)
```

19 文字处理方法

19.1 去除首尾的特定字符

```
a = ',ab,x,'
a.strip(',')
#output:ab,x
a = ' ab,x '
a.strip(',')
output:ab,x
```

20 os 库用法

20.1 readme

本次使用代码中 python 文件的绝对路径是:/Users/heizijian/Desktop/test.ipynb, 注意路径中是'/' 还是'\',mac 是'/',linux 是'\'

20.2 获取该文件所在的文件夹

```
os.getcwd()
# output: '/Users/heizijian/Desktop'
```

20.3 获取文件中的文件名

```
# os.path.abspath(path)
# 返回为绝对路径

os.path.abspath('test.ipynb')
# output: '/Users/heizijian/Desktop/test.ipynb'
```

20.4 将多个路径组合返回

```
# os.path.join(path1,path2 ,...)
# 返回为path1/path2

abspath = os.getcwd()
text_file = 'test.ipynb'
os.path.join(abspath, text_file)
# output: '/Users/heizijian/Desktop/test.ipynb'

# 效果等同于 abspath+'/test.ipynb'
# 效果等同于 os.path.abspath('test.ipynb')
```

20.5 返回文件路径的目录

```
# os.path.dirname(path)
# 返回指定路径的目录名，不包括文件名。
# 既可以输入绝对路径，也可以输入相对路径
# os.path.dirname('/Users/ heizijian /Desktop/ test .ipynb')
# output: '/Users/ heizijian /Desktop'

os.path.dirname('heizijian/Desktop/test.ipynb')
# output: ' heizijian /Desktop'
```

20.6 判断路径是否存在，判断文件是否存在，判断目录是否存在

```
# 是返回True，否返回False

# 判断路径是否存在
os.path. exists (path)

# 判断文件是否存在
os.path. isfile (path)
os.path. isdir (path)
# 判断目录是否存在
```

21 leetcode

21.1 readme

leetcode 这部分仅仅是给一个分类，如果是想学习 leetcode 的具体解法的话还是需要去官网上寻找解法

21.2 回溯算法:

77 组合

216 组合总和 3

17 电话号码的字母组合

39 组合总和

40 组合总和 2

21.2.1 回溯算法

深度优先搜索，达到题目要求的条件时存入结果，剪枝

21.2.2 77 组合

- n = 4,k = 2
- output:[[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]
- 回溯算法

21.2.3 39 组合总和

- candidates = [2,3,6,7], target = 7
- output:[[2,2,3],[7]]
- 回溯算法

21.2.4 40 组合总和 2

- candidates = [10,1,2,7,6,1,5], target = 8 只能用一次
- output:[[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]]

- 回溯算法
- 注意去重

21.2.5 216 组合总和 3

- 描述：找出所有相加之和为 n 的 k 个数的组合，只用 1-9，只用 1 次
- k = 3, n = 9
- output:[[1,2,6], [1,3,5], [2,3,4]]
- 回溯算法

21.2.6 17 电话号码的字母组合

- digits = ”23”
- output:[”ad”,”ae”,”af”,”bd”,”be”,”bf”,”cd”,”ce”,”cf”]
- 回溯算法
- 注意：从 a 到 ad 到 ae 到 af 到 b
- 最后用”.join([’a’,’b’])

21.3 最优化问题:

53 最大子数组和

84 柱状图中最大的矩形

85 最大矩形

363 矩形区域不超过 K 的最大数值和

40 组合总和 2

21.3.1 53 最大子数组和

题目描述:

- nums = [-2,1,-3,4,-1,2,1,-5,4]
- 求 nums 连续的最大和
- 思路：dp[i] 为包含 nums[i] 的最大和
- dp[i-1] <= 0,dp[i] = max(dp[i-1] + nums[i],nums[i])
- dp[i-1] > 0,dp[i] = dp[i-1] + nums[i]

21.4 动态规划:

10 正则表达式匹配

73 编辑距离

124 二叉树到最大路径和

198 打家劫舍

494 目标和

最长公共子序列

21.4.1 10 正则表达式匹配

题目描述: 给两个字符串 s 和 p, 要求用 p 中的字符匹配 s, p 的范围'a-z','.','*'

- s = 'aaa'
- p = 'ab*.*'
- return True
- dp[i][j] 分别对应 s[i-1],p[j-1]
- 注意初始状态第一行, 因为 b* 可以匹配 0 个字符

21.5 动态规划:

37 解数独

51N 皇后

78 子集

90 子集 2

93 复原 ip 地址

491 递增子序列

21.5.1 37 解数独

21.5.2 51N 皇后

要求 Q 之间不能在同一行，同一列，同一个斜线

- 用三个列表保存，columns，left，right
- 如果存在 col in columns or row-col in left or row+col in right 返回

21.5.3 78 子集

- nums = [1,2,3]
- output:[[],[1],[2],[3],[1,2],[1,3],[2,3],[1,2,3]]
- 回溯算法
- 得这样理解 [[],[1],[1,2],[1,2,3],[1,3],[2],[2,3],[3]]

21.5.4 90 子集 2

- nums = [2,2,3]
- output:[[],[2],[3],[2,2],[2,3],[2,2,3]
- 回溯算法
- 注意剪枝
- j > start and nums[j] = nums[j-1] ->continue

```
print('hello') #注释
a = [1,2,3]
for i in range(len(a)):
    print(i)
```

参考文献