Long context3-上下文窗口分割

上下文窗口分割

通过将上下文分割成段，并采用滑动窗口方法来处理上下文。

PCW(Parallel context windows for large language models)

LLM的输入分为上下文token（上下文文档或者检索到的文档）和任务token（要分类的句子或者问题本身）。

PCW使用的是Decoder框架，输入和输出都在Decoder侧。

- 位置编码：LLM的上下文窗口长度为N，任务token长度为T，上下文的窗口长度为C=N-T。对于要处理的长上下文，将其分割为B段，每一段长度为C，总的长度为BC+T。PCW位置编码为：

$$\vec{p}_i^{PCW} = \begin{cases} \vec{p}_{(i-1 \bmod C)+1} & 1 \le i \le BC \\ \vec{p}_{i-(B-1)C} & BC < i \le BC + T \end{cases}$$

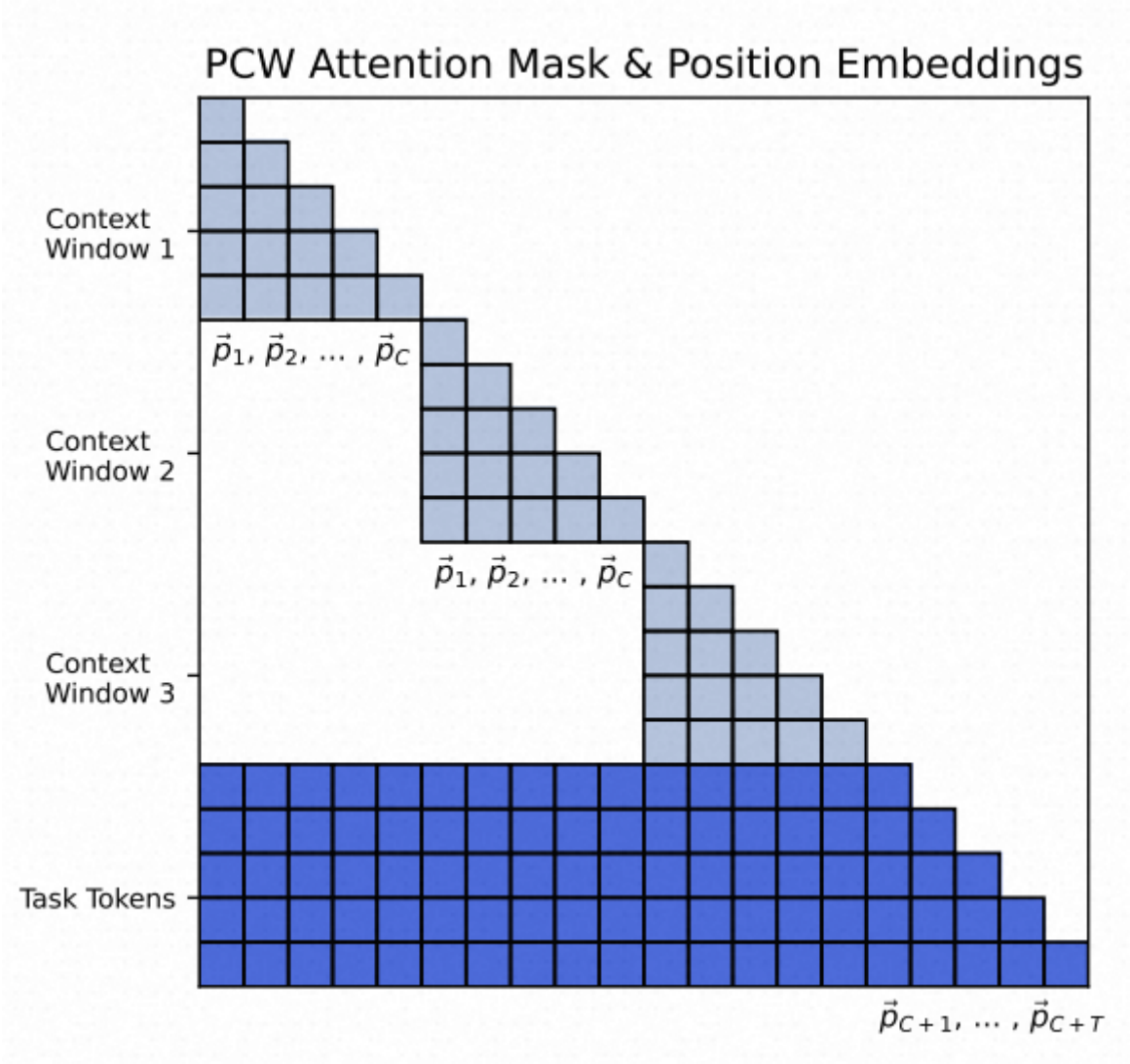
在解码时丢弃了上下文多段文本之间的位置关系，解码时只知道上下文多段文本都是在解码器之前，但无法区分文本之间的位置。不过因为上下文每段文本复用了相同的位置编码，因此位置编码的长度大幅降低，也就降低了对位置编码外推性的需求。

- 注意力矩阵：基于PCW编码，在执行注意力计算时，每个窗口内部进行自回归，然后将结果进行拼接，各个窗口复用同一个位置编码。任务token和所有上下文中的token都计算注意力。

PCW需要的计算复杂度正比于并行上下文数量B，但注意力矩阵很稀疏，多窗口并行的效率很高。

存在的问题：

1. 但是在长文本QA问题上表现比较一般，当上下文存在多段文本且无明显关系时，正确答案中会混杂很多无关的文本变短。
2. PCW是在输入层就开始对超长上文进行Attention，因为不同上文的位置编码相同，一定程度上会让解码注意力变得非常分散，导致注意力的熵值变高，解码的不确定性变大，更容易出现乱码。



NBCE

假设T是要生成的token序列，S1,S2,...,Sn是相对独立的Context集合（比如n个不同的段落，至少不是一个句子被分割为两个片段那种），假设它们的总长度已经超过了训练长度，而单个Sk加T还在训练长度内。我们需要根据S1,S2,...,Sn生成T，即估计p(T|S1,S2,...,Sn)。

基于独立假设的贝叶斯公式，即朴素贝叶斯：

$$p(T|S_1, S_2, \cdots, S_n) \propto p(S_1, S_2, \cdots, S_n|T)p(T) \tag{1}$$

这里 $P(S_1, S_2, ..., S_n)$ 是1所以被省略，由独立假设可以进一步得到：

$$p(S_1, S_2, \cdots, S_n|T) = \prod_{k=1}^{n} p(S_k|T) \tag{2}$$

$$p(T|S_1, S_2, \cdots, S_n) \propto p(T) \prod_{k=1}^{n} p(S_k|T) \tag{3}$$

另外根据贝叶斯公式：

$$p(S_k|T) \propto \frac{p(T|S_k)}{p(T)},$$

$$p(T|S_1, S_2, \cdots, S_n) \propto \frac{1}{p^{n-1}(T)} \prod_{k=1}^{n} p(T|S_k)$$

$$\log p(T|S_1, S_2, \cdots, S_n) = \sum_{k=1}^{n} \log p(T|S_k) - (n-1)\log p(T) + 常数 \tag{5}$$

这里的p(T|Sk)和p(T)都可以直接用现有的LLM进行计算,且不涉及长文本。

记 $\beta = n - 1$, 且

$$\overline{\log p(T|S)} = \frac{1}{n} \sum_{k=1}^{n} \log p(T|S_k) \tag{6}$$

就可以得到：

$$\log p(T|S_1, S_2, \cdots, S_n) = (\beta+1)\overline{\log p(T|S)} - \beta \log p(T) + 常数 \tag{7}$$

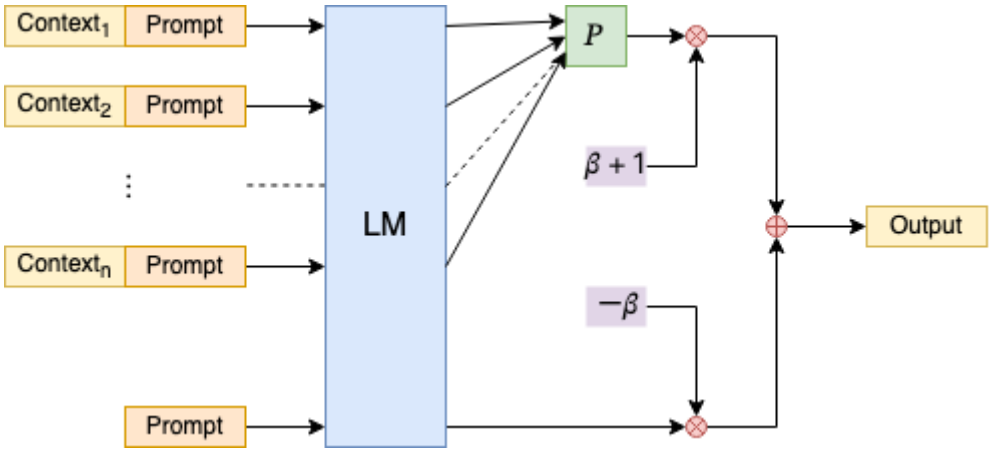
- ☐ 在阅读理解场景中Max Pooling配合β=0.25，用Greedy Search总体表现比较好，然而Random Sample出来的结果基本不可读。Random Sample是“按照分布采样”，它的效果差说明Max Pooling的结果不是一个合理的分布；而Greedy Search只关心最大概率者，而不关心分布的合理性，它的效果好告诉我们概率最大的token正确性较高。
- ☐ $log\bar{p}(T|S)$ 本质是在做Average Pooling，也可以换成其他的Pooling方法：

$$\log p(T|S_1, S_2, \cdots, S_n) = (\beta+1)\mathcal{P}[\log p(T|S)] - \beta \log p(T) + 常数 \tag{8}$$

概率越大说明结果的不确定性越低，将Pooling方式改为直接输出不确定性最低的那个分布，就得到了NBCE。

$$\begin{aligned} &\mathcal{P}[\log p(T|S)] = \log p(T|S_k) \\ &k = \operatorname{argmin}\left\{H_1, H_2, \cdots, H_n\right\} \\ &H_i = -\sum_{T} p(T|S_i)\log p(T|S_i) \end{aligned} \tag{9}$$

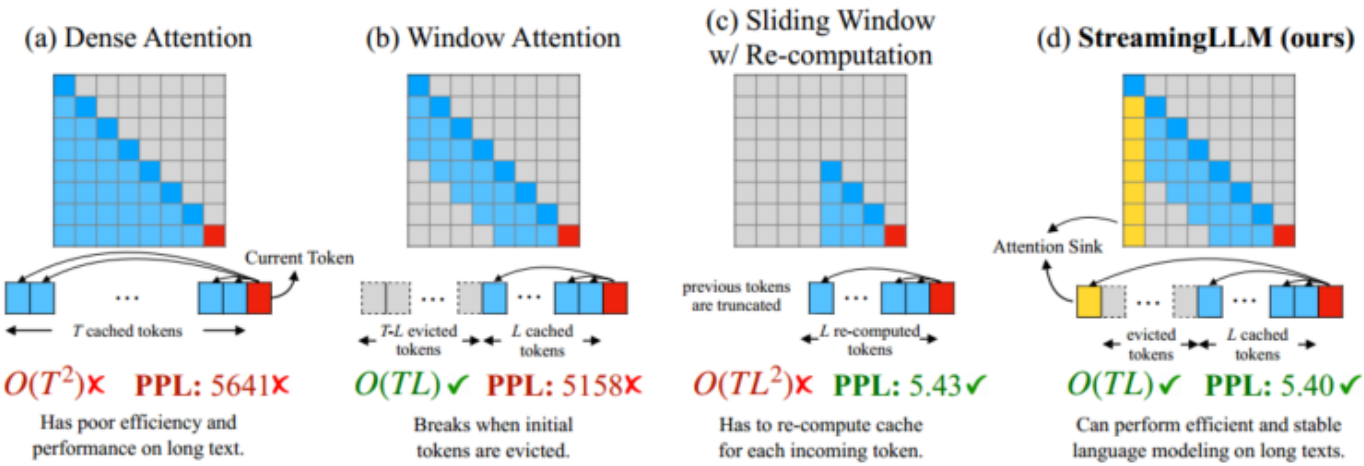
NBCE中不同Context的预测结果通过方法P聚合（或者说投票）在一起（权重为β+1），并减去无Context的预测结果（权重为β）。之所以要减去无Context预测结果，是为了让模型更加倾向于结合Context而不是纯粹根据自身知识储备来回答。



存在的问题：与PCW类似，当上下文增加时，输出的结果不准确，具体表现为主题相关，但是作为问题的答案来说是错误的。并且由于无法识别Context输入顺序，在故事续写等场景表现欠佳。

PCW大致上就是Average Pooling版的NBCE，我们实测也发现它跟Average Pooling版的NBCE有着相似的缺点。

streaming-LLM

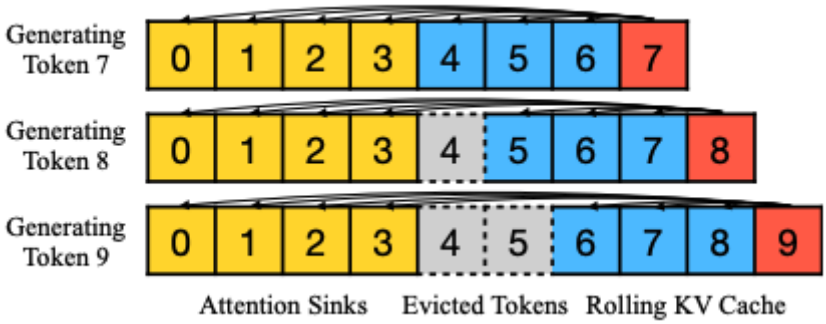


- a) 密集注意力：时间复杂度为O(T^2),当推理文本超过预训练长度时，困惑度大幅度上升
- b) 窗口注意力：只维护最近的L个token的KV，但是当序列长度超过缓存大小时，失去第一个token的KV，会导致困惑度增加。
- c) 滑动窗口与重新计算：为每个新的token重建最近token的KV状态（这样一直保持有初始token）。虽然它在长文本上表现良好，但它的 $O(TL^2)$ 复杂性(源于上下文重新计算中的二次注意力)使得它相当慢。
- d）streaming-LLM：保留attention sink(注意力汇聚，汇聚在初始的几个tokens) 与最近的token结合，用于稳定的注意力计算。

观察到大量的注意力得分被分配给初始的token，即使它们与任务的相关性不高（即模型重视初始tokens的绝对位置，而不是它们的语义价值）。主要原因是因为Softmax操作，要求所有上下文token的注意力分数总和为1。因此，即使当前任务和许多先前的token不匹配，模型仍然需要在某个地方分

配这些不需要的注意力分数，使得分数总和为1。由于初始token对几乎所有后续token都是可见的，所以这些额外的注意力都汇聚在初始的token上。

StreamingLLM将注意力汇聚的前4个初始token和滑动窗口的KV结合在一起，可以有效地推广到无限长的序列长度。



StreamingLLM在确定相对距离和添加位置信息时，关注缓存中的位置而非原文，以保障模型的效率。 例如，如果当前高速缓存具有令牌[0，1，2，3，6，7，8]并且正在解码第9个令牌的过程中，则分配的位置是[0,1,2,3,4,5,6,7]，而不是原始文本中的位置，该位置将是[0,1,2,3,6,7,8,9]。

为了避免模型过度关注初始的token：

- 引入一个全局可训练的注意力汇聚token。
- 修改softmax函数：不再使用真实的权重概率向量,允许所有位置的attention值都很低。

$$\text{SoftMax}_1(x)_i = \frac{e^{x_i}}{1 + \sum_{j=1}^{N} e^{x_j}},$$

参考：**NBCE：使用朴素贝叶斯扩展LLM的Context处理长度**

Efficient Streaming Language Models with Attention Sinks

解密Prompt系列8. 无需训练让LLM支持超长输入:知识库 & unlimiformer & PCW & NBCE