llm基础知识1 - transformer学习

本次作业主要是针对transformer的学习，题目主要分为简答题以及代码复现题，作业要求使用ipynb格式，对于代码复现题需要自行构建简单的测试模块，同时在作业最后需要提供完成作业所参考的材料

一、数据预处理

语言模型是对文本进行推理。由于文本是字符串，但对模型来说，输入只能是数字，所以就需要将文本转成用数字来表达，这便是tokenizer的作用，这通常都是nlp输入到model前的第一步，从粒度出发，主要分为以下几种类型：

- **word（词）**

词是最简单的方式，例如英文可以按单词切分。缺点就是词汇表要包含所有词，词汇表比较大；还有比如“have”,"had"其实是有关系的，直接分词没有体现二者的关系；且容易产生oov问题（Out-Of-Vocabulary，出现没有见过的词）

- **char（字符）**

用基础字符表示，比如英文用26个字母表示。比如 "China"拆分为"C","h","i","n","a"，这样降低了内存和复杂度，但增加了任务的复杂度，一个字母没有任何语义意义，单纯使用字符可能导致模型性能的下降。

- **subword（子词）**

结合上述2个的优缺点，遵循”**尽量不分解常用词，将不常用词分解为常用的子词**”的原则。例如"unbelievable"在英文中是un+形容词的组合，表否定的意思，可以分解成un”+"believable"。通过这种形式，词汇量大小不会特别大，也能学到词的关系，同时还能缓解oov问题。

subword分词主要有**BPE,WorkdPiece,Unigram**等方法。

q1：请自行查阅资料学习BPE,WorkdPiece,Unigram方法的具体实现，并选择一个算法详细介绍

q2：SentencePiece是一种无监督的文本 tokenizer 和 detokenizer，主要用于基于神经网络的文本生成系统，请自行学习该库的使用，并完成分词任务

google/sentencepiece: Unsupervised text tokenizer for Neural Network-based text generation. (github.com)

使用SentencePiece在西游记数据集上训练分词器，西游记.txt_数据集-阿里云天池 (aliyun.com)，并进行测试（当然中文数据集一般可以用jieba库来进行分词，可自行选择或者都尝试一下）

二、架构学习

参考文献阅读：

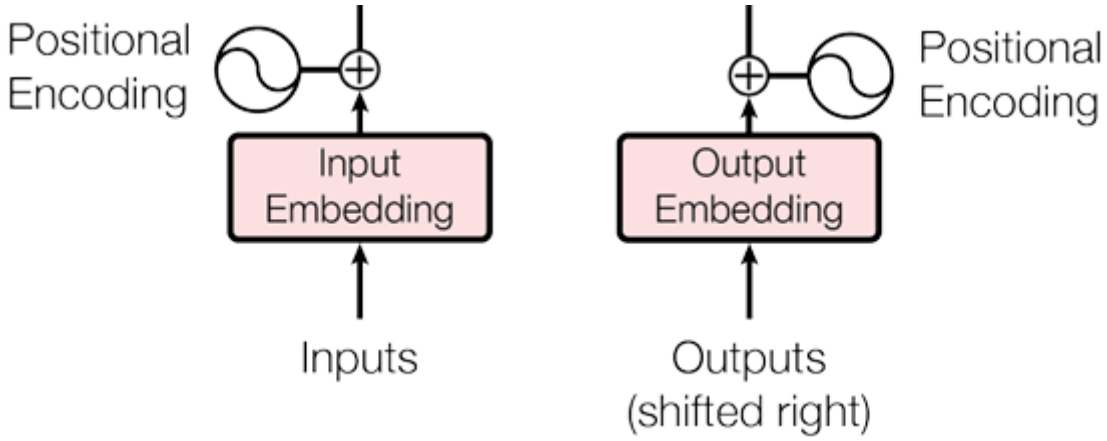
https://arxiv.org/pdf/1706.03762

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

https://jalammar.github.io/illustrated-gpt2/

1. embedding层

在初代的transformer种embedding层的操作可以总结为word embedding + positional embedding，如图所示：



q3：请解释embedding层的作用，重点介绍为什么使用positional embedding以及positional embedding的原理

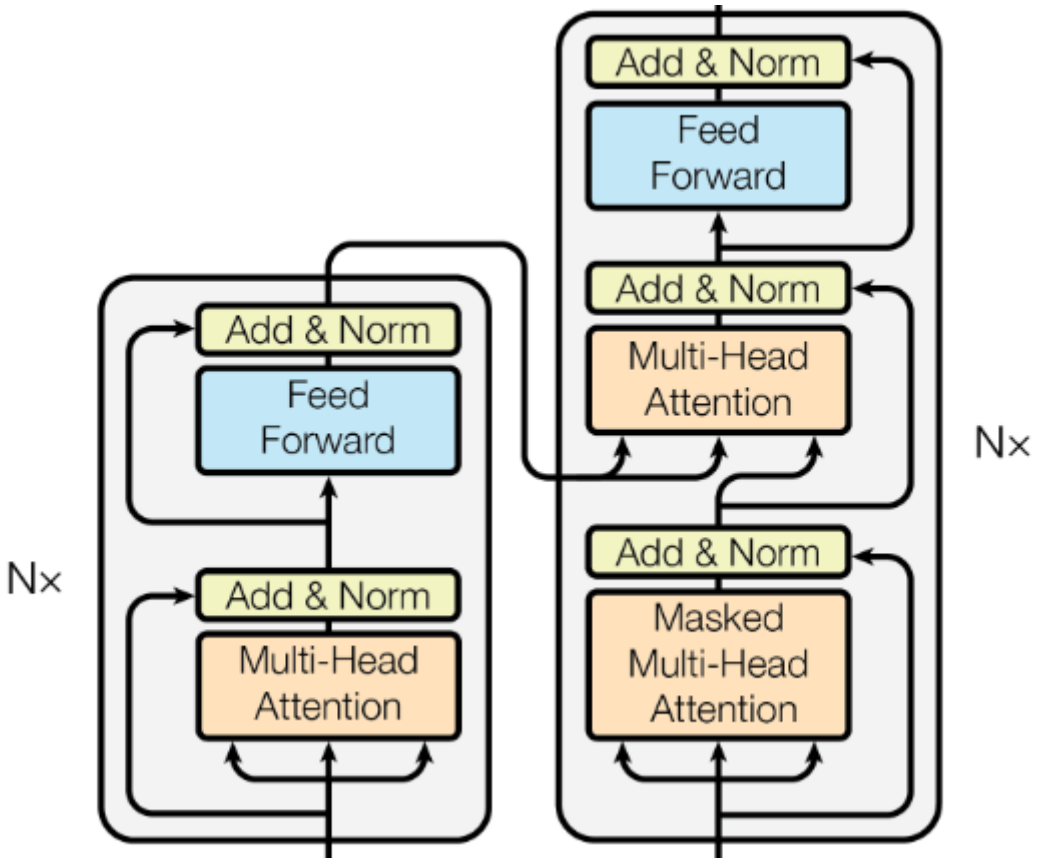
q4（选做）：完成下列positional embedding接口的实现

```
class PositionalEncoding(nn.Module):
    def __init__(self, input_dim, max_len=1000):
        super(PositionalEncoding, self).__init__()
        # your code

    def forward(self, X):
        # your code
```

2. attention

注意力机制允许model关注到长序列的文本信息，多头注意力通过并行执行多个注意力机制，并将结果合并，可以捕获不同子空间的信息，请自行学习注意力机制，回答以下问题：



q5：如何理解注意力机制中的key、value、query的作用，试着从矩阵变化的角度来解释其输入X维度的变化

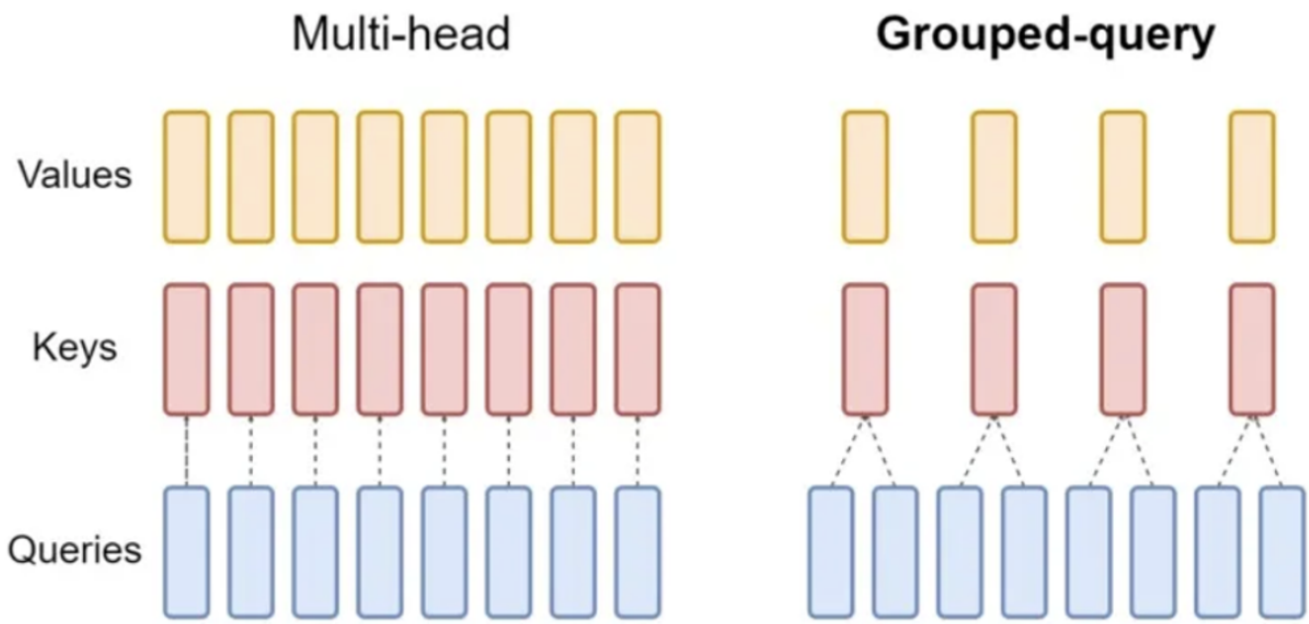
q6：完成简单版本的多头注意力的实现，接口如下：

提示：transformer中的多头注意力分为编码阶段和解码阶段，编码阶段默认是可以看到整个序列的信息，因此mask此时没有作用，只有在解码阶段才使用mask；

```
class SimpleMultiHeadAttention(nn.Module):
    def __init__(self, embed_dim, num_heads):
        super().__init__()
        self.embed_dim = embed_dim
        self.num_heads = num_heads
        # 假设kqv都不改变原先的维度大小，也就是其输入输出维度都为embed_dim
                # your code
                self.out = nn.Linear(self.embed_dim,self.embed_dim)

        # 简单起见，mask定义为一个下三角矩阵，为 1 时代表token有效，为 0 时无效
    def forward(self,key,query,value,mask):
        # your code
        return output, attention_weight
```

在llama系列中对注意力机制做了相应的变化，如下图：



GQA的具体思想是，不是所有 queries 头共享一组 key和value，而是分组一定头数 query 共享一组key和value，比如上面图片就是两组 query共享一组 key和value。

q7：请解释这样做的好处

q8：完成GQA的具体实现，接口与`SimpleMultiHeadAttention` 一致

3. Feed-Forward

Feed-Forward层作为前馈网络与自注意力机制相结合，使得Transformer能够捕捉长距离依赖关系，下面是他的公式实现：

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

q9：完成Feed-Forward层的代码实现：

```
class PositionwiseFeedForward(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(PositionwiseFeedForward, self).__init__()
        # your code

    def forward(self, x):
        # your code
```

4. **Layer Normalization**

Layer Normalization（层归一化）在计算归一化时不是在整个 mini-batch 上，而是在单个数据点的特征上进行，请自行学习其原理，并回答以下问题：

q10： 介绍Layer Normalization的原理，并分析它与batch normalization的区别，以及为什么Transformer用Layer Normalization而不用batch normalization?

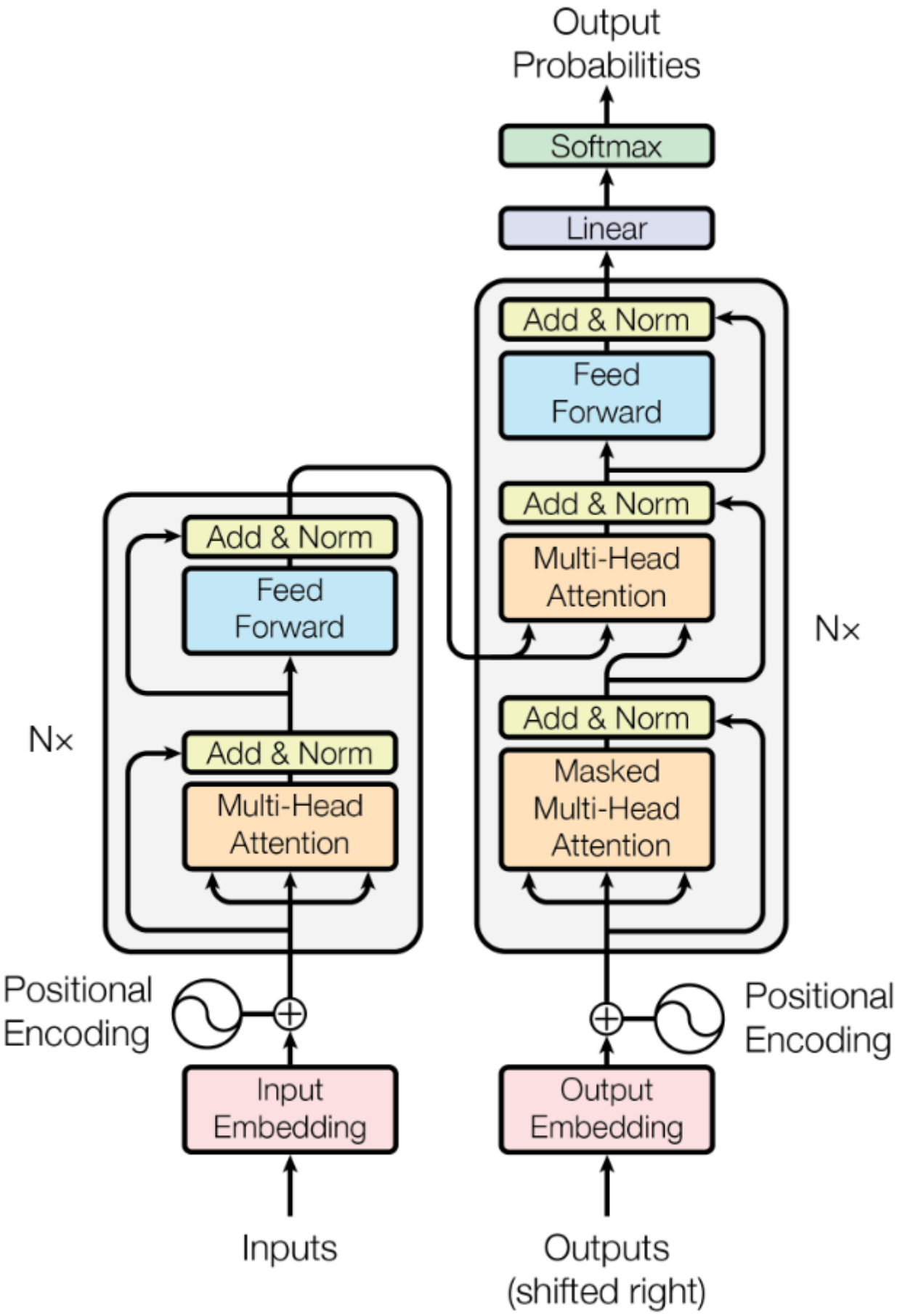
q11（选做）：实现Layer Normalization

```
class LayerNorm(nn.Module):
    def __init__(self, dim, eps=1e-6):
        super().__init__()
              # your code

    def forward(self, x):
                # your code
        return out
```

5. 模型整体架构搭建

q12：现在你应该可以完成整个transformer的搭建工作，给定相应接口完成transformer的构建：




```
# 基础的TransformerBlock模块
class TransformerBlock(nn.Module):
```

```
        ## 参数自行定义，你也可以单独构建一个ModelArg类作为参数传递类
    def __init__(self, layer_id: int, **kwargs):
        super().__init__()

    def forward(self,x: torch.Tensor,mask: Optional[torch.Tensor]):
            # your code
        return out
```

```
# 你可能需要自行构建encoder和decorder类
class Transformer(nn.Module):
        # 同样参数自行定义
    def __init__(self, params: ModelArgs):
        super().__init__()
        # your code

        # input_tokens.shape == (batch_size,seq_len)
    def forward(self, input_tokens: torch.Tensor):
        # your code
        return output
```

q13：尝试在你构建的transformer上训练数据，可以参考d2l中的做法

三、推理接口构建

transformer类模型的推理过程是一个自回归的过程，也就是每次都把输出的包括prompt序列放入model中生成一个token，其目标就是使得输出序列的每一步的条件概率相乘最大：

$$P(\text{“its water is so transparent”}) = P(\text{its}) \times P(\text{water}|\text{its}) \times P(\text{is}|\text{its water}) \times P(\text{so}|\text{its water is}) \times P(\text{transparent}|\text{its water is so})$$

我们在transformer最后一层使用softmax之后会得到一个形状为`(batch_size, seq_len, vocab_size)`的概率分布，如何选取合适的token便是推理时需要重点考虑的，目前有以下几种方式：

- greedy search：每步取概率最大的输出，然后将从开始到当前步的输出作为输入，取预测下一步，直到句子结束。
- beam search：对贪心算法做了优化，在每个step取beam num个最优的tokens。

q14：详细介绍beam search的做法，解释`num_beams`、`top_k`、`top_p`、`temperature`参数的作用

q15（选做）：下列代码节选自某框架，请根据上一题的学习完成代码补全：

```
# logits是model生成的概率分布，其经过了softmax层，logits.shape ==
(bs,seqlen,vocab_size)
logits = self.model.forward(tokens[:, prev_pos:cur_pos], prev_pos)
```

```
        if temperature > 0:
                    # your code

        # 贪婪搜索
        else:
            # your code

    def sample_top_p(probs, p):
        """
        Perform top-p (nucleus) sampling on a probability distribution.

        Args:
            probs (torch.Tensor): Probability distribution tensor.
            p (float): Probability threshold for top-p sampling.

        Returns:
            torch.Tensor: Sampled token indices.

        Note:
            Top-p sampling selects the smallest set of tokens whose cumulative
            probability mass exceeds the threshold p. The distribution is renormalized
            based on the selected tokens.

        """
            # your code
        return next_token
```

四、（选做）llama2源码阅读

llama系列由meta ai开发，其在transformer基础上改进了例如 **RoPE（相对位置编码）**、**GQA**以及**RMSNorm**的内容，请自行阅读llama2的源码，并回答以下问题：

- 介绍RoPE（相对位置编码），说明他的原理以及优势
- RMSNorm是什么，相对于transformer的计算有何优势
- llama使用only-decoder的架构，相比encoder-decoder架构有何优势
- 源码中的`cache_k`、`cache_v`是什么，涉及到什么知识点并介绍他的作用
- 相较于transformer前馈层使用`relu`，llama2使用`silu`，两者有什么区别
- forward中有一个参数`start_pos`，它的作用是什么，介绍`generation.py`中生成token的流程