Transformer相关——（7）Mask机制

发表于 2021-08-17 | 更新于 2021-08-19 | 深度学习
| 字数总计: 1,759 | 阅读时长: 7分钟 | 阅读量: 17070

Transformer相关——（7）Mask机制

引言

上一篇结束Transformer中Encoder内部的小模块差不多都拆解完毕了，Decoder内部的小模块与Encoder的看上去差不多，但实际上运行方式差别很大，小模块之间的连接和运行方式下一篇再说，这里我们先来看一下Decoder内部多头注意力机制中的一个特别的机制——Mask（掩膜）机制。

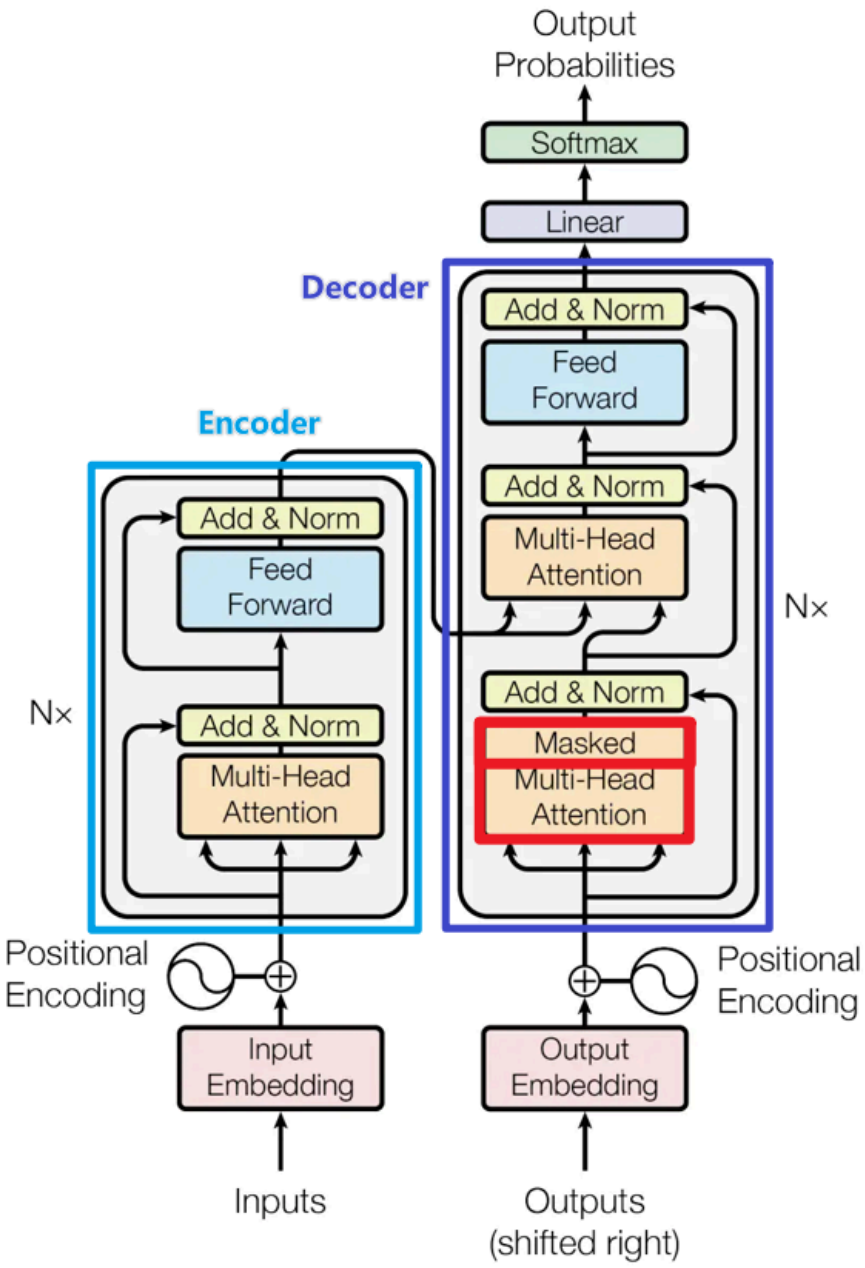


Figure 1: The Transformer - model architecture.

image-20210818091913204

Mask机制

Mask机制经常被用于NLP任务中，按照作用总体来说可以分成两类：

1. 用于**处理非定长序列**的padding mask（非官方命名）；
2. 用于**防止标签泄露**的sequence mask（非官方命名）。

Transformer中同时用到了这两种Mask机制。

padding mask

在NLP任务中，文本通常是不定长的，所以在输入一个样本长短不一的batch到网络前，要对batch中的样本进行truncating截断/padding补齐操作，以便能**形成一个张量的形式输入网络**，如下图所示。对于一个batch中过长的样本，进行截断操作，而对于一个长度不足的样本，往往采用特殊字符"<PAD>"进行padding（也可以是其他特殊字符，但是pad的字符要统一）。Mask矩阵中可以用1表示有效字，0代表无效字（也可以用True/False）。

padding mask生成和使用

1. **padding（补齐）操作在batch输入网络前完成**，同步生成padding mask矩阵；
2. padding mask矩阵常常**用在最终结果输出、损失函数计算等等，一切受样本实际长度影响的计算中，或者说不需要无用的padding参与计算时候。**

padding mask使用案例

RNN中的Mask

> 在 pytorch 中，对 mask 的具体实现形式不是mask矩阵，而是通过一个句子长度列表来实现的，但本质一样。实现如下，sentence_lens 表示的是这个batch中每一个句子的实际长度。
>
> ```
> 1    embed_input_x_packed = pack_padded_sequence(embed_input_x, sentence_lens, batch_first=True)
> 2    encoder_outputs_packed, (h_last, c_last) = self.lstm(embed_input_x_packed)
> 3    encoder_outputs, _ =
> ```

```
pad_packed_sequence(encoder_outputs_packed,
batch_first=True)
```

在 pytorch 的 Embedding 和 Loss 中也有对 padding 值的设置：

```
# padding_idx (int, optional): If given,
pads the output with the embedding vector at
# `padding_idx` (initialized to zeros)
whenever it encounters the index.
embedding = nn.Embedding(vocab_size,
embed_dim, padding_idx=0)

# ignore_index (int, optional): Specifies
a target value that is ignored
# and does not contribute to the input
gradient.
criterion =
nn.CrossEntropyLoss(ignore_index=0)
```

self-attention中的padding mask

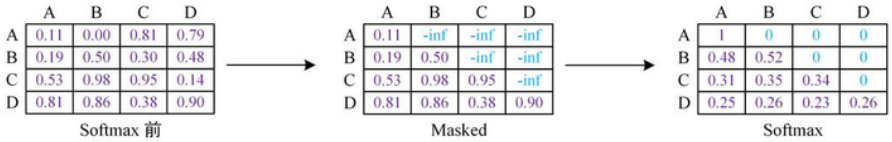
Q和K在点积之后，需要先经过mask再进行softmax，因此，对于要屏蔽的部分，mask之后的输出需要为负无穷，这样softmax之后输出才为0。

```
def attention(query, key, value,
mask=None, dropout=None):
    "Compute 'Scaled Dot Product
Attention'"
    d_k = query.size(-1)
    scores = torch.matmul(query,
key.transpose(-2, -1)) / math.sqrt(d_k)
    if mask is not None:
        scores = scores.masked_fill(mask
== 0, -1e9) # mask步骤，用 -1e9 代表负无穷
    p_attn = F.softmax(scores, dim = -1)
    if dropout is not None:
        p_attn = dropout(p_attn)
```

```
10        return torch.matmul(p_attn, value),
p_attn
```

sequence mask

sequence mask有各种各样的形式和设计，最常见的应用场景是在需要一个词预测下一个词的时候，如果**用self attention 或者是其他同时使用上下文信息的机制，会导致模型"提前看到"待预测的内容**，这显然不行，所以**为了不泄露要预测的标签信息，就需要 mask 来"遮盖"它。**如下图所示，这也是**Transformer中Decoder的Masked Multi-Head self-attention使用的Mask机制**。



除了在decoder部分加入mask防止标签泄露以外，还有模型利用这种填空机制帮助模型学的更好，比如说BERT和ERNIE模型中利用到的**Masked LM**（MLM）。（注意：**BERT模型只有Transformer的Encoder层，是可以学习上下文信息的**）

BERT中的Mask

> **Masked LM随机掩盖部分输入词，然后对那些被掩盖的词进行预测。**在训练的过程中，BERT随机地掩盖每个序列中15%的token，并不是像word2vec中的cbow那样去对每一个词都进行预测。MLM从输入中随机地掩盖一些词，其目标是基于其上下文来预测被掩盖单词的原始词汇。

而ERNIE不是在token级进行掩码，而是在短语级进行掩码。

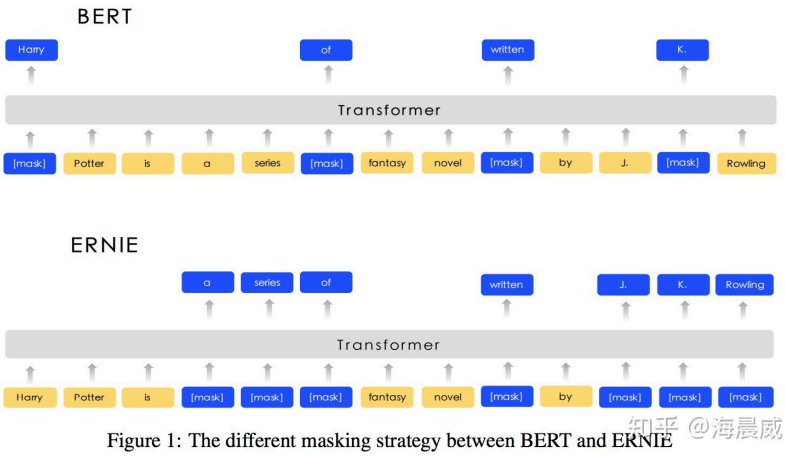


Figure 1: The different masking strategy between BERT and ERNIE

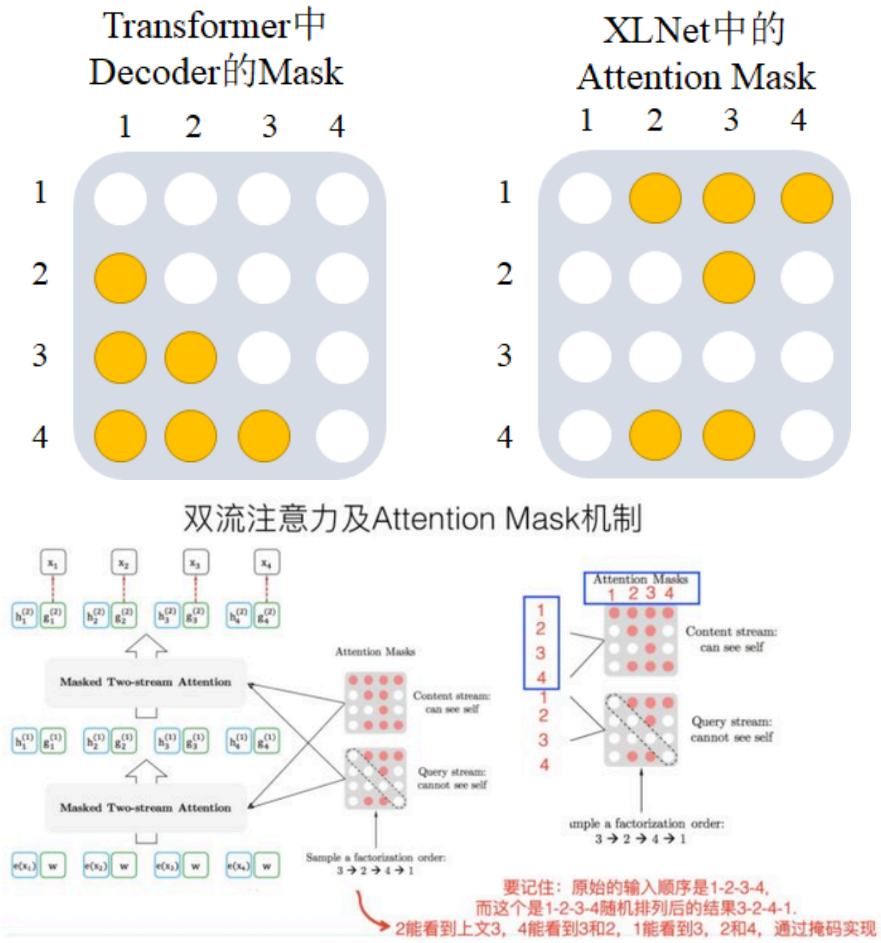
XLNet中的Mask

另外一个经常被提到的**XLNet中的Mask**机制在这里也简单总结一下，为什么要如此设计请参考：XLNet:运行机制及和Bert的异同比较

上面介绍了BERT中的Mask机制是通过加入随机将文本中的词替换成"[MASK]"标记，通过上下文预测该掩码位置的词。而XLNet并没有利用这种方法，而是通过 **Permutation Language Modeling (PLM)** 重排输入文本的想法，而实际实现中，使用的是**Attention Mask 机制（对attention层进行掩码）**来完成重排输入文本这一操作。

比如序列$1->2->3->4$，假设我们要对第3个位置进行掩码和预测，那么我们首先先确定输出在位置3，然后对所有词进行重排序，其重排序列有很多种，比如
$3->2->4->1;\ 2->1->4->3;\ 2->4->3->1$
等等，XLNet的思想就是从这些重排序的序列中选一部分出来做训练，这样看上去模型还是从左到右的顺序，但位置3却可以学习到上下文的信息（包括1、2和4）。

但XLNet中不是直接通过固定位置重排序实现的，而是通过attention mask机制，以$3->2->4->1$这一重排序列为例的attention mask如下图所示，白色为0，黄色为1，那么2能看到3，4能看到3、2，1能看到3、2、4。



UniLM中的Mask

> UniLM (Unified Language Model)，仅用Mask，就让BERT可以同时构建双向语言模型，单向语言模型和seq2seq语言模型。
>
> 上面三种语言模型的差异，就在于训练时能利用哪些信息，具体实现上，UniLM就通过Mask来控制信息的利用，语言模型的学习上还是和BERT一样，通过完形填空，预测被mask位去学习。
>
> 下图展示了上述三种语言模型是如何通过设计不同Mask实现的：

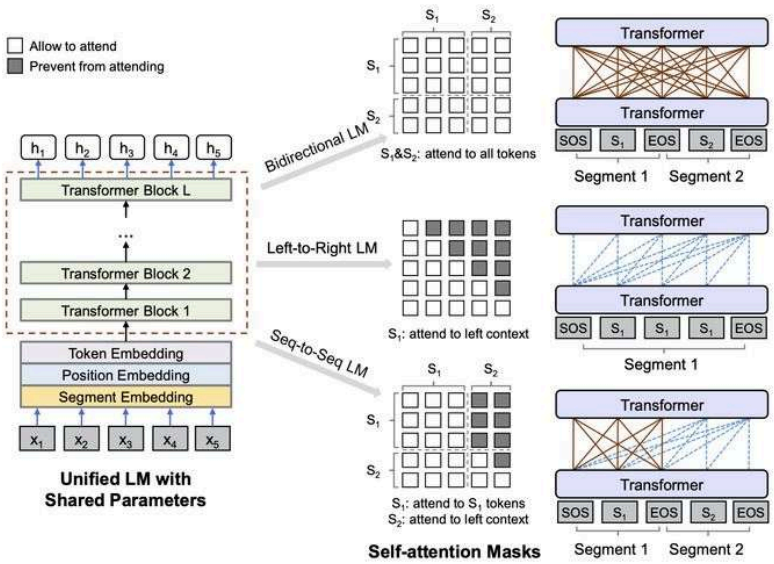


Figure 1: Overview of unified LM pre-training. The model parameters are shared across the LM objectives (i.e., bidirectional LM, unidirectional LM, and sequence-to-sequence LM). We use different self-attention masks to control the access to context for each word token. The right-to-left LM is similar to the left-to-right one, which is omitted in the figure for brevity.

上图的中间展示的就是不同语言模型的Self-Attention Mask 矩阵（没有考虑 padding mask，仅有sequence mask，假设被mask位为0)：

双向语言模型：和BERT一样，可以是一句或两句话，全1的mask矩阵，即可以看到上下文所有信息

单向语言模型：仅一句话，从左到右预测的话是上三角为0的mask矩阵，仅能看到上文。

seq2seq语言模型：需要两句话，第一句话可以看到本身所有信息，但看不到第二句话任何信息；第二句话可以看到第一句话所有信息，本身只能看到上文。所以它的mask矩阵包括四个部分，从左到右，从上到下分别是：全0，全1，全0，上三角为1。

参考文献

NLP 中的Mask全解

BERT算法原理解析

XLNet:运行机制及和Bert的异同比较

XLNet 中神奇的 Attention Mask