Transformer 学习笔记

为什么需要Transformer?

此前，循环神经网络（RNN）及其改进版本（如LSTM、GRU）是序列建模的主流方法，但它们存在固有缺陷：

RNN系列模型的局限性：

1. **长距离依赖建模困难**：随着序列长度增加，RNN在捕捉远距离元素间依赖关系时效果不佳，存在梯度消失/爆炸问题
2. **串行计算的限制**：每一步计算都严格依赖前一步结果，无法充分利用并行计算资源

也有人研究利用CNN进行序列建模，但仍有以下问题难以得到解决：

CNN在序列建模中的挑战：

1. **局部感受野的局限**：需要堆叠深层次网络才能建模长距离依赖，增加参数量和计算复杂度
2. **信息流动效率低**：序列起始位置信息需要多层卷积才能传递到尾部，可能导致信息衰减

针对此，Transformer首次提出完全基于**自注意力机制**构建的编码器-解码器架构，核心优势：

1. **全局信息交互**：自注意力机制允许序列中任意两个位置直接建立联系，从根本上解决长距离依赖问题
2. **强大的并行计算能力**：自注意力层计算可在序列长度维度上完全并行化，充分利用GPU硬件优势

对transformer架构的基本解释

embedding

Word embedding

将输入词转换为向量嵌入。

Position encoding
allow transformer to keep track of the word.

Attention

- Self-Attention（自注意力）

  针对序列本身内部的token进行

  - single-head attention
  - multi-head attention
- Cross-Attention（交叉注意力）

Basic concept

Query

代表“当前需要关注什么”的向量。可以理解为当前需要生成输出的那个位置（比如要翻译的当前目标词）

Key

代表序列中**所有位置**所包含的“标识信息”的向量。用于与Query计算相似度

Value

代表序列中**所有位置**所包含的“实际内容信息”的向量。最终加权求和的来源

核心思想

对于给定的Query，计算它与序列中**所有位置**的Key的**相似度**（或相关性）。这个相似度分数决定了该位置的Value在最终输出中应占的**权重**。权重高的Value对最终输出的贡献大

single head of attention

Query matrix

将 **嵌入空间** 中的名词映射到较小的 **查询空间** 中的某个方向，用向量来编码 **寻找前置形容词** 的概念

Key matrix

从概念上讲，可以把 **键** 视为想要回答 **查询**。和查询矩阵一样，它也会将嵌入向量映射到相同的低维空间

Value matrix

输入和输出都存在于高维的嵌入空间

值向量

如果这个词需要调整目标词的含义，那么要反映这一点得对目标词的嵌入加上什么向量呢?

值向量与嵌入向量处于同一个高维空间

更新嵌入向量

对于网格中的每一列，你需要给每个值向量乘上该列的对应权重，然后将该列中所有带权值向量加和，得到想要引入的变化量，最后把它加入到原始嵌入向量上，预期是能得到一个更精准的向量，编码了更丰富的上下文信息

dot production

为了衡量每个键与每个查询的匹配（对齐）程度，我们计算所有可能的**键-查询对**之间的点积。
这个值可以是$-inf$到inf的任何实数，它代表 **每个词与更新其它词含义有多相关**
key words 和 query words 的点积越大，可以理解为前者的嵌入 **attend to** 后者的嵌入，反过来若点积是较小的正数或负值，可以认为两个词互不相关

softmax

归一化后每一列的数值介于 0 和 1 之间，且每列总和为 1。此时每一列可看作权重，表示左侧的键与顶部的查询的相关度

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

这里将所有点积除以 键—查询 空间维度的平方根即$\sqrt{d_k}$是为了提高数值的稳定性，使注意力logits的一组值也符合均值为0，方差为1的分布。

多头注意力将每个头计算所得的结果矩阵拼接在一起即可。

$$Attention(Q, K, V) = concat($$

$$Attention(Q_{h1}, K_{h1}, V_{h1}),$$

$$Attention(Q_{h2}, K_{h2}, V_{h2}),$$

$$Attention(Q_{h3}, K_{h3}, V_{h3})$$

$$)$$

masking（掩码）

由于 **self attention** 机制，Transformer 模型在处理单词时，会同时“看到”句子里的所有单词。为了不让后词影响前词（泄露 **答案**），我们希望在**attention patten** 的矩阵里左下方这些代表着后方 token 影响前方的位置能被强制变为 0

Question 可以直接将它们设为0吗?

Answer 显然不行，这样一来每列总和就不再是 1 了，不再是归一化的了。常见方法是在应用 *softmax* 之前，先将它们设为$-inf$，这样在应用 *softmax* 之后，它们就都会接近于0，但列仍保持归一化。

> $\mathrm{Softmax}(x) = \frac{e^{x_i}}{\sum_i e^{x_i}}$

分类

- Look-ahead Mask / Subsequent Mask
- Padding Mask
  - 用于处理不同长度的序列。为了批量处理，把短句子用特殊符号`[PAD]`补齐到相同长度，不让它们参与注意力计算

上下文窗口大小

其大小等于上下文长度的平方，扩大上下文长度是大语言模型的巨大瓶颈

Encoder结构

- **Input Encoding**
  - word embedding
  - positional encoding

    allow transformer to keep track of the word
- **Multi-Head Attention**

  Self-Attention
- **Add & Norm**
  - residual connection
  - layer normalization
- **Feed Forward**

关于参数的一点小小改进

设计上值矩阵的参数量可以比键矩阵和查询矩阵多几个数量级，但实际上更为高效的做法是，让值矩阵所需要的参数量等于键矩阵和查询矩阵的参数量之和

具体做法是将值矩阵分解为两个小矩阵相乘。本质上是对大矩阵进行 **低秩分解**
Value down matrix
第一个矩阵行数较少，通常等于键—查询空间的维度，可以看作是将较大的嵌入向量**降维**到较小的空间
Value up matrix
第二个矩阵则是从小空间映射回嵌入空间，得到的是用于实际更新的向量

Cross-attention

交叉注意力和自注意力几乎相同，唯一的区别是 **键和查询矩阵作用于不同的数据集**，例如，文本翻译模型中，键可能来自一种语言而查询则来自另一种语言，这个注意力模式就可以描述一种语言中的哪些词对应另一种语言中的哪些词，在这种情况下通常不会用到掩码，因为不存在后面 token 影响前面 token 的问题

multi-head of attention

我们此前已描述了单头注意力，而 Transformer 内完整的注意力模块由多头注意力组成，大量并行地执行这些操作，而每个头都有不同的键、查询、值矩阵

96 个不同的键和查询矩阵将产生 96 种不同的注意力模式，每个注意力头都有独特的值矩阵，用来产生 96 个值向量序列，全部都将以对应注意力模式作为权重分别进行加权求和

对于每个 token，**每个头都会给出一个要加入到该位置的嵌入中的变化量**，而把各个头给出的变化量加起来，然后给该位置的初始嵌入加上这个加和，这个总和就是多头注意力模块输出的一列，也就是一个更精准的嵌入

为什么需要多头?
类比于卷积神经网络中的多个卷积核可以检测不同特征。通过并行多头计算，模型能学习到根据上下文来改变语义的多种方式，捕获更丰富的关系

补充

output matrix

在上文，我们提到过 *value up matrix* 和 *value down matirx*，但在实际的论文写法中往往有所不同。这些 *value up matirx* 会合在一起形成**输出矩阵**（*output matrix*），与整个多头注意力模块相连。

而单个注意力头的值单指 *value down matirx*

Layer Norm

层归一化会按照每个token分别统计每个token所有特征的均值和方差。所有token共享相同数量（与embed_dim相同）的 γ 和 β 参数。

$y = \frac{x - E[x]}{Std[x] + \epsilon} * \gamma + \beta$

通过Layer Norm，它保证了：

- 每个token的特征大致分布为均值为0，方差为1。因此后边进行注意力计算时点积值的稳定性，不会落入softmax的失活区域

- Layer Norm因为是对输入每个token embedding的维度进行均值和方差的统计。对于推理状态也可以进行，所以不区分训练状态和推理状态

Questions

1. 为什么序列数据归一化的方式选择采用层归一化（Layer Normalization）而不是Batch Norm?

   因为序列可能很长，导致Batch Size很小，不同Batch 之间的均值和方差差距会非常大。

   同时，大量的 `<pad>` token的特征也会影响正常token的均值和方差计算。

   另外，BatchNorm本身还有一个问题，那就是区分训练和推理状态，训练时会动量更新维护每个特征的均值和方差。推理时需要用训练时保存的均值和方差。这样比较麻烦。

2. Pre-LN和Post-LN的区别是什么?

   Post-LN是*《Attention is all you need》*原始论文给出的方案，目前绝大多数主流模型已优先选择Pre-LN结构。两者之间的核心区别：**LayerNorm 是放在残差连接之前还是之后。**

   ```
   # Post-LN
   ...
   return self.dropout(self.norm(sublayer(x) + x))
   # Pre-LN
   ...
   return x + self.dropout(sublayer(self.norm(x)))
   ```

   相较于Post-LN，Pre-LN的训练稳定性更好，收敛速度也更快，对深层网络训练也更加友好。

Conclusion

注意力机制成功的主要原因：能用 GPU 在短时间内运行大量并行计算

Reference

- 直观解释注意力机制，Transformer的核心
- 把这一切组装起来 · GitBook