Transformer 深入挖掘

MLE 算法指北

2025 年 2 月 3 日

1 multi-head attention 计算公式常见问题

1.1 Transformer 使用多头注意力机制的优势？

回答： 多头注意力机制（Multi-Head Attention）可以让模型从不同的表示子空间中学习不同的特征，增强模型的表达能力。使用多个注意力头有如下优点：

- 提高模型的鲁棒性，减少单个注意力头可能带来的信息丢失。
- 允许不同的注意力头关注输入序列的不同部分，使得模型能够捕捉更丰富的上下文信息。
- 增强模型的能力，避免单一注意力模式可能导致的局部最优问题。

1.2 为什么 Q 和 K 需要使用不同的权重矩阵？

回答： 如果 Q 和 K 使用相同的权重矩阵，则 QK^T 变成了一个对称矩阵，这样会导致注意力分数的计算缺乏灵活性，无法区分不同的输入关系。因此，通常使用不同的权重矩阵来生成 Q 和 K：

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

其中 W_Q, W_K, W_V 为不同的投影矩阵，以提供不同的信息表示。

1.3 计算注意力使用点积计算优势

回答： 在 Transformer 中，点积注意力（Dot-product Attention）是主要的注意力计算方式。相比于加法注意力（Additive Attention），点积计算方式的优点包括：

- 计算效率更高，矩阵运算可以使用并行计算加速。
- 避免了额外的可训练参数，而加法注意力需要额外的权重矩阵和非线性变换。
- 适用于高维度的向量，尤其是使用多头注意力时，点积计算更为高效。

1.4 为什么对 QK^T 进行 scaled 处理？

回答： 在计算注意力分数时，我们使用了 QK^T 计算相似度：

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

其中 d_k 是 K 的维度。如果不进行缩放，当 d_k 较大时，QK^T 的数值会变得过大，导致 softmax 的梯度消失问题。因此，除以 $\sqrt{d_k}$ 可以使得分布更加平稳，提高训练稳定性。

1.5 为什么在进行多头注意力时需要先 concat 再降维？

回答： 如果不进行降维，每个头的计算量会变得很大，因此一般将 d_{model} 维的输入向量映射到一个较低维度的子空间，使得每个注意力头的计算量减少：如果我们直接相加：所有注意力头的输出都会累加到同一个维度上，而多头注意力的核心优势在于不同的头可以关注不同的特征。直接相加会让这些不同特征的表示混合在一起，可能导致模型无法区分不同的注意力模式，降低表达能力。

1.6 Masked multihead attention 如何实现

回答： 在计算注意力分数时，对于 padding 位置，我们可以使用一个 mask 矩阵 M，将这些位置的注意力分数设为 $-\infty$，从而在 softmax 计算后变为 0：

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T + M}{\sqrt{d_k}}\right)V$$

其中，M 的值为：

$$M_{ij} = \begin{cases} 0, & \text{如果} j \text{ 不是 padding 位置} \\ -\infty, & \text{如果} j \text{ 是 padding 位置} \end{cases}$$

2 Transofer 架构常见问题

2.1 简要讲一下 Transformer 的 Encoder 模块

回答： Transformer 的 Encoder 由 N 层堆叠而成，每一层包含以下主要部分：

- **多头自注意力（Multi-Head Self-Attention）**：捕捉输入序列中不同位置之间的依赖关系。
- **前馈神经网络（Feed Forward Network, FFN）**：用于对每个 token 进行非线性变换，提高模型的表达能力。
- **残差连接（Residual Connection）+ Layer Normalization**：保证梯度稳定，提高训练效果。

计算流程：1. 计算 ** 自注意力 **：

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

其中 Q, K, V 均由输入 X 经过投影得到：

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

2. 计算 ** 前馈神经网络（FFN）**：

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

3. 通过 ** 残差连接 + LayerNorm** 归一化输出：

$$X' = \text{LayerNorm}(X + \text{MultiHead}(X, X, X))$$

$$Y = \text{LayerNorm}(X' + \text{FFN}(X'))$$

2.2 讲一下 Transformer 的 Decoder 模块

回答 Transformer 的 Decoder 由 N 层相同的解码层堆叠而成，每层包含：

- **Masked 多头自注意力（Masked Multi-Head Self-Attention）**：防止当前 token 看到未来信息，保证自回归生成。
- **Encoder-Decoder 多头注意力（Multi-Head Attention over Encoder Outputs）**：让 Decoder 关注 Encoder 计算出的特征。
- **前馈神经网络（Feed Forward Network, FFN）**：增强表达能力。
- **残差连接（Residual Connection）+ Layer Normalization（LayerNorm）**：稳定训练，提高梯度传播能力。

2. 计算流程假设输入序列的表示为 X，编码器的输出为 Z，每一层的计算流程如下：

1. **Masked 多头自注意力**防止未来信息泄露，仅利用已生成的 token 计算注意力：

   $$\text{MaskedAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T + M}{\sqrt{d_k}}\right)V$$

   其中 M 是 mask 矩阵，屏蔽当前 token 之后的所有位置。

2. **Encoder-Decoder 多头注意力**让 Decoder 关注 Encoder 的信息：

   $$\text{CrossAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

   其中 Q 由 Decoder 生成，K 和 V 由 Encoder 计算得到。

3. **前馈神经网络（FFN）**对每个位置的特征进行独立的非线性变换：

   $$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

   其中 W_1, W_2, b_1, b_2 是可训练参数。

4. **残差连接 + LayerNorm** 每个子层后都加上残差连接：

   $$X' = \text{LayerNorm}(X + \text{SubLayer}(X))$$

3. Decoder 的最终输出 Decoder 最后的输出会经过一个线性变换 + Softmax 层，得到最终的预测分布：

$$P(y_t|y_{<t}, X) = \text{Softmax}(WO_t)$$

其中 O_t 是 Decoder 最后一层的输出。

4. 总结

- Transformer 的 Decoder 由多个解码层组成，每层包含：Masked 自注意力、Encoder-Decoder 注意力、前馈神经网络、残差连接和 LayerNorm。
- Masked 机制确保 Decoder 逐步生成输出，避免未来信息泄露。
- Encoder-Decoder 注意力使 Decoder 能够关注 Encoder 提供的特征。

2.3 残差连接的作用和意义

残差连接的主要作用如下：

- 缓解梯度消失问题：梯度可以沿着残差路径传播，使得深层网络的训练更加稳定。
- 加速收敛：通过跳跃连接，使得梯度流动更加顺畅，减少训练时间。
- 允许信息直接传播：输入信息可以直接通过残差路径流向更深层的网络，防止信息丢失。
- 提高模型泛化能力：残差连接结合 LayerNorm，可以防止模型过拟合，提高泛化能力。

2.4 为什么 Transformer 使用 LayerNorm 而不是 BatchNorm？

在 Transformer 中，LayerNorm 更适合于序列建模任务，主要原因如下：

- **适用于变长输入序列**：BatchNorm 依赖于 mini-batch 统计量，而 NLP 任务中的序列长度各不相同，BatchNorm 可能表现不稳定，而 LayerNorm 不受影响。
- **独立于 batch size**：BatchNorm 需要较大的 batch size 来获得稳定的均值和方差，而 Transformer 训练时 batch size 通常较小，LayerNorm 更稳定。
- **更适用于自注意力机制**：在 Transformer 结构中，每个 token 之间的计算是独立的，LayerNorm 可以为每个 token 进行归一化，而 BatchNorm 需要整个 batch 的统计信息，可能导致不稳定。
- **训练更稳定**：LayerNorm 适用于梯度传播，能够减少梯度消失/爆炸问题，提高训练收敛速度。

3 Transformer 的位置编码及其意义

3.1 为什么 Transformer 需要位置编码？

Transformer 不像 RNN 那样按顺序处理输入序列，而是并行计算，这使得模型无法直接获得位置信息。为了让 Transformer 了解输入序列中单词的顺序，我们需要引入 **位置编码（Positional Encoding）**，将其加入词嵌入（Embedding）中：

$$X' = X + PE$$

其中：- X 是词嵌入向量（Embedding），维度为 d_{model}。- PE 是位置编码，维度与 X 相同，使得模型能够学习到位置信息。

3.2 正弦-余弦位置编码（Sinusoidal Positional Encoding）

Transformer 论文提出了一种基于正弦和余弦函数的位置编码方法，其计算公式如下：

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

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

其中：- pos 表示序列中的位置（Position）。- i 表示词向量的维度索引。- d_{model} 是 Transformer 词嵌入的维度。

优点：- 位置编码对序列长度没有限制，适用于任意长的序列。- ** 不同位置之间的相对关系可以通过线性变换学习，即模型可以利用 $PE(pos)$ 来推断相对位置信息。

3.3 可训练位置编码（Learnable Positional Encoding）

除了正弦-余弦编码，Transformer 也可以使用 **可训练的位置编码**，即：

$$PE = W_{\text{pos}}$$

其中 W_{pos} 是可学习参数，模型在训练过程中自动调整其值。

优点：- 可学习的位置编码能够根据任务优化，但缺点是 **对序列长度有限制**，无法处理比训练时更长的输入序列。

3.4 位置编码的作用和意义

位置编码的核心作用是提供序列的位置信息，使得 Transformer 能够学习顺序信息：

- 保留顺序信息：解决 Transformer 无法捕捉词序的问题。
- 支持并行计算：相比 RNN，不需要逐步传递序列信息，而是一次性引入位置信息。
- 能学习相对位置信息：尤其是正弦-余弦编码，可以帮助模型学到相对距离，而不仅仅是绝对位置。

4 Transformer 并行计算

Transformer 相比于传统的 RNN 和 LSTM 具有 ** 更强的并行计算能力 **，主要体现在以下几个方面：

4.1 自注意力机制的并行计算

在 RNN 或 LSTM 中，序列数据需要按照时间步（time step）进行逐个处理，计算是 **串行**的，而 Transformer 采用自注意力机制（Self-Attention），可以对整个序列进行矩阵运算，从而实现并行计算。

自注意力的计算公式如下：

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

其中：- Q, K, V 是整个序列的表示矩阵（大小为 $n \times d_{\text{model}}$）。- 计算 QK^T 时，所有 token 之间的关系是同时计算的，不需要像 RNN 那样按顺序处理每个时间步。

由于该计算仅涉及矩阵乘法，可以使用 GPU/TPU 进行大规模并行计算，极大地加速了训练过程。

4.2 多头注意力的并行计算

多头注意力（Multi-Head Attention）通过多个不同的投影矩阵，对输入数据进行多个独立的注意力计算：

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O$$

其中：- 每个头的计算方式是：

$$\text{head}_i = \text{Attention}(QW_{Q_i}, KW_{K_i}, VW_{V_i})$$

- 不同头的计算是完全独立的，可以并行执行，并最终拼接（Concat）。

由于多个注意力头的计算可以同时进行，这使得 Transformer 具有更强的计算效率。

4.3 前馈神经网络（FFN）的并行计算

Transformer 中的每个位置都会通过相同的前馈神经网络（Feed Forward Network, FFN）** 进行处理：

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

由于 FFN 对每个 token 是独立的，所有 token 的 FFN 计算可以并行进行，进一步提高计算速度。

4.4 位置编码（Positional Encoding）无依赖性

相比 RNN 需要通过时间步依赖性传播信息，Transformer 使用位置编码（Positional Encoding）来表示顺序信息：

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

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

这种方式使得不同位置的信息可以同时输入 Transformer 进行处理，而不需要像 RNN 那样逐步传播。

4.5 训练过程的并行化

Transformer 采用 Teacher Forcing 训练方法，即：- **Encoder 部分**：输入序列 X 可以一次性全部处理，不需要按时间步展开。- **Decoder 部分**：训练时目标序列 Y 也是同时输入 Decoder，不需要逐步预测（预测时才是自回归）。

这种方法使得训练阶段的计算可以完全并行化，相比 RNN 需要逐步展开的计算方式，Transformer 训练更快，能充分利用 GPU/TPU 进行高效计算。

4.6 总结

Transformer 之所以比 RNN 更容易并行化，主要体现在：

- 自注意力计算是基于矩阵运算，不依赖时间步展开，可并行处理整个序列。
- 多头注意力计算是独立的，多个头可以同时计算注意力分数。
- 前馈神经网络对每个 token 是独立的，可以批量并行计算。
- 位置编码代替了循环信息传播，消除了对前一步状态的依赖。
- 训练时使用并行化的 Teacher Forcing，而 RNN 训练需要逐步展开，难以并行化。

由于这些特性，Transformer 可以高效利用 GPU/TPU 进行大规模并行计算，大幅提高训练速度，尤其适用于大规模数据和长序列任务。