因果自注意力

这是 Transformer Block 中比较关键的一个部分，其中包含两个线性层和两个 Dropout 层，还包含多头注意力层。

一些需要预先知道的参数:

- context length: 最大上下文长度，比如 GPT2-small 为 1024。
- heads: 分成多少个头计算，比如 GPT2-small，有 12 个头。
- head_dim (D): 假如模型第一层的嵌入维度为 768，那么每个头的嵌入维度就是 $D = \frac{768}{12} = 64$
- 缩放因子 (scale): 缩放因子用来将 $Y = Q \cdot K^T$ 得到的 Y 进行归一化:

$$scale = \frac{1.0}{\sqrt{D}}$$

- 掩码矩阵 (mask): 是一个 N=context length 的下三角方阵，类似这样:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- Q_w,K_w,V_w 三个权重矩阵，其各自尺寸一般都是N=嵌入维度的方阵，在 GPT2 官方的权重参数中，一般将 Q_w,K_w,V_w 三个权重矩阵合并到一起，形成一个 [768,2304] 的大矩阵。

因果注意力公式:

$$CausalAttention(Q, K, V) = Softmax\left(\frac{QK^T + M}{\sqrt{D_k}}\right)V$$

前向传播

前向传播输入形状为 [B,T,C]:

- B: 批次
- T: token 数量
- C: 嵌入维度，比如 768

注意力权重:

- Q: 查询向量。它代表了当前正在处理的这个词的“意图”，或者它“想要查找什么信息”。
- K: 键向量。它代表了序列中所有其他词的“内容摘要”，或者它们“能提供什么信息”。
- V: 值向量。它包含了序列中所有词的实际信息，一旦注意力分数被计算出来，这些信息就会被加权聚合。

输出形状与输入形状相同

输入与注意力权重矩阵相乘

设输入张量为 X，这里经过一个线形层，完成 X 与 Q_w,K_w,V_w 相乘，因为 Q_w,K_w,V_w 合并到一个大的张量，所以只需要将输入与这个大张量做乘法就可以了。

此步骤的输出形状为 [B,T,C*3]，其中:

$$Q = X \cdot Q_w$$

$$K = X \cdot K_w$$

$$V = X \cdot V_w$$

如果在训练场景下，我们需要缓存这个张量。

多头计算

将上述 [B,T,C*3] 的张量分为 12 个头(假设是 GPT2-small) 分别计算，每个头计算的方法完全一致。

对于每个头 Q_h , K_h , V_h 的形状为 [B, T, head_dim], 从理论上来说，就是上述步骤得到的 Q, K, V 按照当前头的顺序分割得到的分块矩阵。

每个头计算的中间结果会形成一个 [T,T] 的矩阵，我们可以实现创建这样一个名为 att 的注意力矩阵。

详细计算步骤:

1. $att = Q_h \cdot K_h^T$: 计算注意力分数。它衡量了 Q 和 K 向量之间的相似度。

   此刻，Q_h 的形状为 [T,head_dim], K_h 的形状为 [T,head_dim]，输出 att 形状为 [T,T]。

2. 注意力分数缩放。

   将上述步骤输出 att 逐个元素乘以缩放因子(scale)。

3. 因果掩码。

   对于 att 矩阵应用掩码矩阵，防止模型在生成当前词的时候看来未来的信息，
   对于序列中的第 t 个词，它只能看到第 1,2,...,t 个词，而不能看到第 t+1,...,n 个词。因为掩码矩阵是下三角矩阵，在掩码矩阵为零的位置，我们将 att 对应位置设置为负无穷（这是为了在下一步 Softmax 计算中使之结果接近为零。

4. 计算 att Softmax

   公式:

   $$p_i = \frac{e^{z_i - z_{max}}}{\sum_{k=1}^{T} e^{z_k - z_{max}}}$$

   其中：z_{max} 是 att 某个行向量的最大值。

5. 缓存第4步的结果（如果在训练模式下，推理场景不需要）。

6. 对 att 应用 dropout，只在训练模式下有效。

7. 缓存第6步的结果（如果在训练模式下，推理场景不需要）。

8. 计算 $att \cdot V_h$。

   这一步可以直接将输出结果映射到输出张量指定的头位置上。

投影变换

主要作用是信息融合，多个注意力头可以同时关注输入序列的不同方面。例如，一个头可能关注语法关系，另一个头可能关注语义关系。投影层的作用是将来自所有注意力头的不同“视角”和“特征”进行有效的融合。它通过学习参数来决定如何组合这些信息，从而生成一个更丰富、更全面的序列表示。

dropout（推理无效）

防止过拟合。多头注意力机制中的每个头都可能学习到输入序列的不同方面。如果这些头总是以相同的方式协同工作，它们可能会形成一种不健康的依赖关系，导致模型过度适应训练数据中的某些特定模式。投影层之后的 Dropout 随机地“关闭”一些输出神经元，这迫使模型不能过度依赖任何特定的注意力头或其组合。

前面的投影层本身是一个具有可训练参数的线性层。Dropout 应用在其输出上，为这个层的学习过程增加了正则化，有助于防止投影层本身过拟合。

反向传播

反向传播严格按照前向传播的逆过程进行操作。输入梯度为 grad_output,其形状为 [B,T,C]，如同前向传播的输出张量。

dropout 层反向传播

这个在另外的文档有较为详细的说明。

投影层反向传播

这个在Linear 文档有详细的推导过程。

多头反向传播

此刻也需要根据头的数量分别计算每个头的反向传播，但是计算过程完全一致，只是用到的 Q_h, K_h, V_h 是当前头的分块矩阵。

我们需要准备一个形如 Q, K, V 的梯度矩阵 grad_qkv，用于保存梯度计算结果。还需要一个 [T,T] 的 grad_att 的注意力头的梯度矩阵，用于缓存每个头的梯度传播中间结果。最终的结果和合并到 grad_qkv 矩阵中。

经过前面两层的反向梯度计算的输出形状依然是 [B,T,C], 这个是我们多头反向传播的输入梯度。我们称为 G_{in}

具体步骤：

1. 计算输入梯度对注意力的梯度。

$$grad_{att} = G_{in}^h \cdot V_h^T$$

2. 计算输入梯度对 V_h 的梯度。

$$grad_v = att_{(after-dropout)} \cdot G_{in}^h$$

这里的 $att_{(after-dropout)}$ 是前向传播缓存的中间结果。

3. dropout 层反向传播

4. Softmax 层反向传播

5. 因果掩码反向传播

   就是将掩码矩阵为零的地方，梯度设置为零即可。

6. 缩放因子

   将梯度逐个元素乘以缩放因子。

7. 计算对 Q_h 的梯度。

$$grad_q = grad_{att} \cdot K_h$$

8. 计算对 K_h 的梯度。

$$grad_k = grad_{att}^T \cdot Q_h$$

计算对 Q, K, V 权重的梯度

这是个线性层，可以根据线性层的反向传播直接计算。

最终的梯度输出形状为 [B,T,C]，与前向传播输入形状一致。