Attention is all your need学习笔记

1. key,value,query：

（1）键（key）：

键是一个向量，表示输入序列中特定位置的信息特征。每个键是用来和查询 (Query) 进行匹配的。

键用于衡量查询和每个位置的信息相关性（即计算注意力权重）

通过输入序列 X 与学习的权重矩阵 W_k 相乘，生成键向量： $K = XW_k$

(2)值（value）：

值也是一个向量，表示输入序列中特定位置的具体内容信息。每个值通常参与最终的输出加权。

值用来计算注意力机制的加权输出，表示与查询相关的实际内容。

通过输入序列 X 与学习的权重矩阵 W_v 相乘，生成值向量： $V = XW_v$

(3)查询（query):

查询 (Query) 是代表目标序列中需要匹配的信息，类似于对整个键值对数据库进行查找的查询向量。

查询向量通过和键向量计算点积，生成注意力分数，随后经过 softmax 操作得到注意力权重：

$$(K, V) = [(k_1, v_1), \ldots, (k_N, v_N)] \quad q, k_i \in R^m \quad v_i \in R^v$$

$$a(q, k_i) = \frac{q^T k_i}{\sqrt{D}} \in R$$

$$D = m$$

$$\alpha(q, k_i) = softmax(a(q, k_i)) = \frac{exp(a(q, k_i))}{\sum_{j=1}^{n} exp(a(q, k_j))} \in R$$

$$Attention((K, V), q) = \sum_{i=1}^{n} a(q, k_i) v_i \in R^v$$

相似度计算：计算query与每个key之间的相似度或相关性

权重计算：将相似度通过softmax函数转换为概率分布，这些概率分布就是分配给每个值的权重。

加权求和：将权重与对应的值相乘并求和，得到最终的注意力输出，这个输出是输入序列的加权和，反映了模型当前关注的重点。

2.基础的编码器与解码器：

1.编码器：

是一个RNN，读取输入句子，可以是双向

编码器是没有输出的RNN

2.解码器：

解码器使用另外一个RNN来输出

seq2seq中的应用：

编码器通过逐步读取输入序列中的每个单词（或者词向量），更新隐藏状态。当编码器处理完输入序列后，最后的隐藏状态（通常包含了整个输入序列的上下文信息）会被传递到解码器，编码器最后时间步的隐状态用作解码器的初始隐状态。

例如，假设编码器正在翻译一个句子，每处理一个单词，它都会生成一个新的隐藏状态，最终隐藏状态就成了该句子的“表示”或“摘要”。

解码器使用编码器传递来的最终隐藏状态作为初始化条件，开始生成目标序列。在生成每个新单词时，解码器会利用当前的隐藏状态和已经生成的单词来计算下一个单词。

3.注意力机制：

注意力机制的关键是通过计算**注意力权重**，让解码器能够在生成每个目标单词时，动态地“选择”关注输入序列的不同部分，而不是只依赖于编码器的最后一个隐藏状态。实际上，解码器每次生成的单词都不是仅仅依赖于某个固定的隐藏状态，而是综合了多个隐藏状态的加权和（即加权上下文向量），这个加权和的权重（即注意力分数）是根据解码器当前的状态与输入序列中每个时间步的隐藏状态之间的相关性动态计算的。

隐藏状态：

1.记忆信息：每当一个新的输入（如一个单词）传入网络时，隐藏状态会更新，以记住输入序列的相关信息。这个状态就像是一个“记忆单元”，帮助模型捕捉输入序列中之前的信息。

2.对序列的建模：在处理每个输入单词时，隐藏状态不仅仅存储当前单词的表示，还会结合之前单词的信息。这使得RNN能够处理顺序数据，保留上下文信息。

4.自注意力机制：

自注意力机制可以理解为一种内部交流的方式，让句子中的每个词（或者图像中的每个区域）在处理自己时，能与其他词或区域进行“交流”，根据它们之间的关系来调整自己的表示。通过这种方式，每个元素都能与自己组内的其他元素进行”沟通“。自注意力机制的核心思想就是在同一组元素内部计算元素之间的相互关系和相关性。

举个例子，对于“我喜欢机器学习”这句话，“喜欢”这个词的输出是由“我”、“喜欢”、“机器学习”这三个词的表示乘以相应的注意力权重后加权求和的结果，具体来说，模型会先计算出每个词对其他词的注意力权重，然后根据这些权重来加权求和，从而得到每个词的新的表示或输出。

自注意力机制是一种特殊的注意力机制，指的是在同一组元素之间计算注意力权重，即元素之间相互之间进行“注意”。在传统的 注意力机制 中，查询（Query）来自一个序列，键（Key）和值（Value）来自另一个序列。而在自注意力中，查询、键和值都来自同一组元素。

具体过程：

第一步，我们首先用输入的词向量x来生成三个新的向量qkv。每次一个输入的词向量都要生成三个新的向量。这三个新的向量分别叫做，查询向量（Query），键向量（Key），值向量（Value）。**这三个向量是通过输入的词向量x和三个矩阵做点乘得到的。这三个矩阵的权重先随机初始化，之后会在训练过程中调整。**

第二步，用查询向量和键向量做点乘得到注意力score，score的值决定了我们需要放多少注意力在相应的其他的输入单词上（即进行相似度计算）。

第三步，将score除以键向量的维度开方，score=Q·KT，如果查询和键的维度很大，即D很大，那么它们的点积值也会变得非常大，如果点积过大，后续转化成概率分布时，softmax输出会变得非常尖锐，导致模型在训练过程中产生数值不稳定或梯度爆炸的问题。

第四步，对第三步中得到的结果进行softmax处理。softmax处理使得第三步的结果都为正值且和为一。

第五步，用每个单词得到的soft score和其值向量的各个元素相乘。在这一步，我们保持那些需要注意的单词的值的完整性。并且，冲淡了那些与当前单词关联性不强的单词，例如softmax socre为0.001的单词。

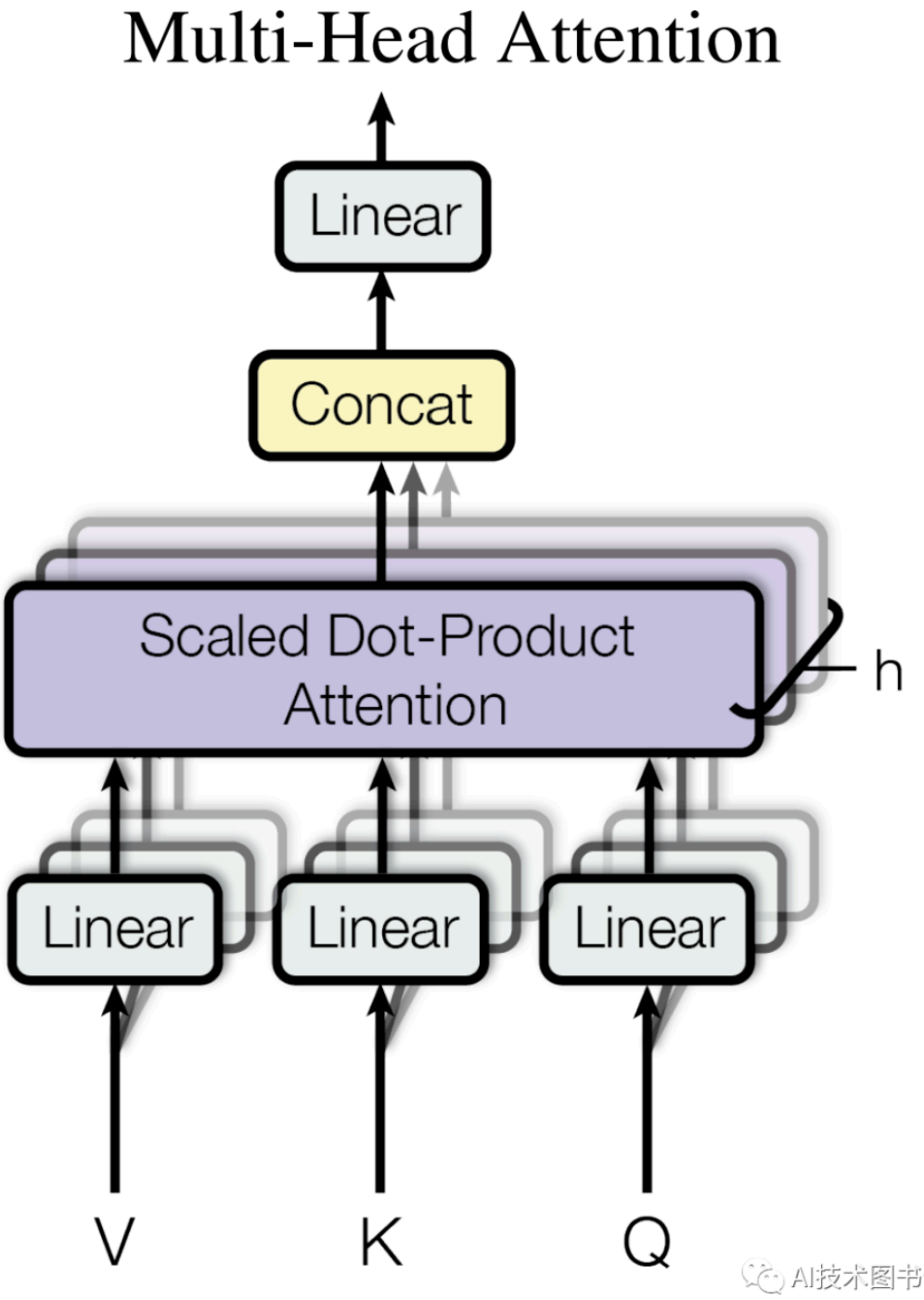
第六步，把第五步中得到的向量相加后得到最后输出。

5.多头注意力机制：

因为一段文字可能蕴含了比如情感维度、时间维度、逻辑维度等很多维度的特征，为了能从不同的维度抓住输入信息的重点，所以有了多头注意力机制。

而所谓多头注意力，简单说就是把输入序列投影为多组不同的Query，Key，Value，并行分别计算后，再把各组计算的结果合并作为最终的结果，通过使用多头注意力机制，模型可以更好地捕获来自输入的多维度特征，提高模型的表达能力和泛化能力，并减少过拟合的风险。

在论文原文中，（V,K,Q）三个矩阵通过h个线性变换（Linear），分别得到h组（V,K,Q）矩阵，每一组（V,K,Q）经过Attention计算，得到h个Attention Score并进行拼接（Concat），最后通过一个线性变换得到输出，其维度与输入词向量的维度一致，其中h就是多头注意力机制的“头数”。



6.位置编码（Positional Encoding）：

1.为什么需要位置编码：

在Transformer中，由于没用递归结构（如RNN）或卷积结构（如CNN），它无法自动捕捉输入序列中元素的顺序信息。为了让Transformer模型理解输入序列中元素的顺序，我们需要显式地为每个位置添加位置信息。这就是位置编码的作用。

2.位置编码的具体原理：

位置编码使用的是正余弦函数。

正弦和余弦函数的频率变化由公式中的 $\frac{1}{10000^{\frac{2i}{d_{\text{model}}}}}$ 控制：

当i较小时，对应的频率较高，位置编码变化迅速，适合捕捉局部的顺序信息。

当i较大时，对应的频率较低，位置编码变化较慢，适合捕捉全局的顺序信息。

正余弦函数的值在[-1,1]范围内平滑变化，使得每个位置之间的编码差异非常自然，模型可以更容易地学习这些变化关系。

两个位置的编码可以通过相对偏移量推导出来，因此模型可以从编码中直接推导相对位置信息。

数学公式：

设输入序列的长度为 L，每个词的位置为 pos，位置编码的维度为 d_{model}（即词嵌入的维度）。对于输入序列中的每个位置 pos，我们可以通过正余弦函数来计算它的编码：

(1)对于位置pos和维度索引$2i$(偶数维度)：

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

(2)对于位置 (pos) 和维度索引 (2i+1) （奇数维度）：

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

注：

pos是当前位置的索引，表示该词在序列中的位置。

i是维度的索引，表示在位置编码中，每个位置的编码会被拆分为多个维度，每个维度通过不同的频率变化。

$dmodel$是模型的嵌入维度（例如，Transformer中每个词的词向量的维度）。

7.残差连接：

1.原因：

残差连接的引入是为了应对梯度消失和梯度爆炸等问题，帮助网络保持梯度的流动，使得训练变得更加稳定，且不容易出现梯度消失或爆炸的情况。

让网络跳过不必要的复杂计算，保留原有的信息，让训练更加高效且稳定。

2.原理：

在传统的神经网络中，假设你有一个层，它的输入为 x，输出为 y，通过某种变换 $F(x)$ 得到：

y=F(x)

而在加入残差连接后，网络不仅仅是将输入 x 转换为输出 y，而是将输入 x 与变换 $F(x)$ 相加。这样，输出变成了：

y=F(x)+x

这个F(x)+x就是典型的残差连接。

其中x是输入数据，F(x)是经过某些变换（如卷积、激活函数等）后的结果。

3.加上x的作用：

（1）信息传递不丢失：即使某些层（如卷积或全连接层）学不到有效的特征，网络也能通过残差连接保留原始输入的信息，防止信息在深层次的传播中消失或变形。

（2）避免训练困难（防止梯度消失或爆炸）：在深层网络中，直接训练很容易导致梯度消失或梯度爆炸。残差连接通过在每层增加输入，保持了梯度流动的路径，使得网络训练更加容易。

（3）引导网络学习变化量：

残差连接的引入，改变了这个学习过程。通过残差连接， 网络不仅学习从x到y的映射F(x)，而是学习输入x和输出y之间的"差异"，即残差。

也就是说，网络不再从零开始学习整个映射，而是学习如何在输入x的基础上进行微小的调整（残差）。这个调整是F(x)函数所计算出的差异。

8.Transformer中的编码器：

在transformer中，编码器由多层堆叠的结构组成，每一层的基本结构为：

输入嵌入和位置编码：首先，输入的每个词都要经过嵌入(embedding)层转换为一个高维向量表示。这个过程就是将离散的词转换为连续的向量，这样神经网络就能处理这些词了。

（1）多头自注意力机制：

多头表示这一过程是同时进行的，多个不同的注意力“头”可以并行地从不同的子空间中学习相关性。通过这种方式，网络能够在不同的层次上关注不同的特征信息（例如语法、语义等）。

（2）前馈神经网络

（3）残差连接和层归一化

前馈神经网络：

在自注意力机制之后，每一层还包括一个前馈神经网络，它通常由两层全连接网络组成。前馈神经网络对每个位置的表示进行独立的非线性变换，增强了模型的表达能力。这部分作用是增加模型的非线性特征，使得网络能够学习更复杂的函数。

多层堆叠增强表示能力：

堆叠多个编码器层能够使模型学习到更复杂的特征。这一过程类似于逐步加深对输入数据的理解，从最初的局部特征到最终的全局抽象。在每一层，输入都被转化为不同的特征表示，捕获到越来越深层的语法和语义信息。通过多层的堆叠，模型的表示能力逐渐增强，能够从更细粒度的特征到更宏观的全局结构上进行学习。

堆叠多个编码器层，模型能够在不同的层次上捕获输入序列的丰富信息和全局依赖，从而提高了表示能力。

举个例子，第一层可能主要关注局部的词语关系（如单词与相邻单词的关系），第二层可能能够理解更大范围的上下文（如句子或段落中的关系），更深层的层次可能能够捕捉到复杂的语法和语义模式。

编码器的目的：

编码器的最终目标是生成一个上下文相关的表示（也叫做上下文向量或上下文嵌入），这些表示将会传递到解码器。

9.Transformer中的解码器：

Transformer中的解码器负责根据编码器的输出逐步生成目标序列。每一层的基本结构为：

输入嵌入和位置编码

(1)掩码多头自注意力机制

(2)编码器-解码器注意力机制

(3)前馈神经网络

(4)残差连接+层归一化

输出层：解码器的最终目标是生成目标词的概率分布，通常通过一个 softmax 层来实现。这个概率分布会决定下一个最可能的目标词。

解码器的目标：

解码器的目标是生成完整的目标序列，通常是通过逐步生成每个词直到结束标志为止。

掩码多头自注意力机制：

原因：

在解码任务过程中，目标序列是逐步生成的。为了保证解码器在每一步只能利用之前生成的词的信息，而看不到未来的词，需要对自注意力机制进行掩码。

原理：

掩码操作通过引入一个上三角矩阵的掩码，让注意力权重对未来词的关注度被设置为负无穷，从而在softmax中这些权重为0。

公式：

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

M是掩码矩阵，当j>i时，Mij等于负无穷，否则Mij等于0。

编码器-解码器注意力机制：

与自注意力机制不同，编码器—解码器注意力机制的Q,K,V来自不同的地方：

Q：来自解码器当前的状态（即解码器当前生成的部分），表示解码器当前关心的信息。

K和V：来自编码器的输出，表示输入序列的全局表示。

解码器会生成当前步骤的查询向量Q，这些查询向量将与编码器的键向量K进行点积计算，得到注意力权重，通过将这些权重应用到编码器的值向量V上，得到加权的编码器输出，从而为编码器提供与输入相关的信息。最终，解码器根据这些加权后的信息生成下一个输出词。

通过这种注意力机制，解码器能够借助编码器的上下文信息，动态地决定在生成下一个词时，输入序列中哪些部分对当前输出最为相关。