Transformer

七、Transformer 的理解（自注意力机制、多头注意力）

transformer 输入数据：

```python
input_text = 'あなたの人生がつまらないと思うんなら、それはあなた自身がつまらなくしているんだぜ。'
```

文本数据需要转为数值数据，先分词。

```python
['あなた', 'の', '人生', 'が', 'つま', '##ら', 'ない', 'と', '思う', 'ん', 'なら', '、', 'それ', 'は', 'あなた', '自身', 'が', 'つま', '##ら', 'なく', 'し', 'て', 'いる', 'ん', 'だ', 'ぜ', '。']
```

文本词 ->ID

```python
[6968, 5, 5386, 14, 16014, 28469, 80, 13, 7105, 1058, 737, 6, 218, 9, 6968, 901, 14, 16014, 28469, 332, 15, 16, 33, 1058, 75, 4403, 8]
```

将文本数据输入到 Transformer 中，先把数据和转换为**词嵌入，即将每个关键词的含义转换为表示连续值的固定长度向量。**

> 注：虽然 ID 转换将文本转换为数字，但这些数字仅仅是代表每个关键词的符号（ID），与其他数字进行计算时没有任何意义。假设两个关键词“我”和“猫”分别被分配了 ID“2768”和“3200”。然而，2768 和 3200 之间没有任何数值上的意义，即使 3199 代表“狗”，也不能说“狗和猫的含义比我和猫更接近，因为它们的数值更接近”。**词嵌入是将词ID转化为连续值的固定长度向量，就是为了让相近的向量表示他们两个词的含义相近。计算词汇的相似度就可以等同为计算向量的相似度了。**

每个词嵌入的特征维度取决于模型和语言。例如，**Transformer 的源文本是英文，基础维度为 512 个维度。Transformer 的源文本也包含 Big 模型，其维度为 1024 个维度。**

> 为啥所有的词嵌入向量维度相同呢?
> （一）第一个原因是“可以变为矩阵运算，从而可以并行计算”。在 Transformer 出现之前，自然语言处理模型就已经存在，其中最著名的是 LSTM。然而，LSTM 需要顺序计算，例如计算一个词，生成该词的状态向量，然后使用该状态向量计算下一个相邻词。由于这种顺序性，输入文本越长，计算时间就越长。实际上，它无法处理目前生成式人工智能所能处理的长文本。但是，固定长度向量的概念使得矩阵运算成为可能。换句话说，每个关键词都可以表示为一行，从而可以创建一个包含固定长度向量特征的矩阵来表示其含义。这使得可以一次性对所有关键词进行矩阵计算（并行计算）。因此，并行计算显著提高了计算速度。
> （二）第二个原因是它支持长期记忆。这与第一个原因有些重叠，但顺序处理的问题在于，早期学习的关键词信息在计算后续词语时很少被反映出来（即被遗忘）（长期依赖问题）。**这个问题也可以通过使用固定长度向量进行矩阵运算来解决，从而实现长期记忆。**然而，仅仅能够处理固定长度向量并不能完全解释这一点；需要注意的是，这需要结合注意力机制（Transformer 的关键特性）才能实现长期记忆（注意力机制将在后面讨论）。

Transformer 有一个叫做嵌入矩阵的矩阵（该矩阵是预训练得到的）。最初的英文 Transformer 模型的嵌入矩阵是 A:37000（单词数）×512。而日语 BERT 模型的嵌入矩阵是 B:32000×**768**。这个矩阵是一个字典，用固定长度的向量表示 37,000 个单词的含义。如关键词“cat”的 ID 是 3200，提取 A[ID,:]为该词的嵌入表示向量。如果矩阵中没有某个关键词，则会返回一个与关键字“UNK”关联的向量。

如果我们用序列长度为 27 的示例句子进行实际测试，我们会得到以下向量：

```python
tensor([-2.6980e-02, -3.7995e-02,  1.0123e-01, -2.4440e-01,  2.9017e-01,
         1.9625e-01, -1.0889e-01, -5.2488e-01,  3.0080e-02, -3.3186e-01,
        -1.5680e-01, -4.0182e-01,  6.3099e-02,  2.6108e-01, -4.8502e-01,
        -4.6710e-01, -1.3623e-01, -3.6355e-01,  3.6669e-01,  3.0299e-01,
        -3.4183e-01, -3.4081e-01,  2.0605e-01, -3.4518e-01,  3.6032e-01,
         1.0690e-01, -1.9173e-01, -7.0575e-02,  1.9683e-01,  2.9114e-01,...
```

现在转换为 embedding（词嵌入）后，为了让模型能够理解位置信息，比如“我爱你”和“你爱我”含义是不一样的，需要加入**位置编码**。用一个相同维度的向量表示每个关键词的词序。

embedding 有次序的表示 == 位置编码向量 + 输入嵌入输出的每个词元向量。

🔔 如何生成位置编码向量?

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

pos 表示关键词在序列中的位置。例如，在“私は猫です。”这句话中，“私”是句子（序列）中的第一个关键词，因此 pos 为 1。

i 表示嵌入维度的索引。对于日语，i 的取值范围为 1 到 768。（日语每个词的 embedding 长度为 768）

d_model 是 768。

数据处理完成后，将输入 transformer 的编码器中。

下面是编码器输入。权重矩阵 W_Q、W_K 和 W_V（分别对应 Q、K 和 V）均使用随机数进行初始化。这些随机数会根据所使用的初始化方法而有所不同，但在许多情

况下，基于 Transformer 的模型会采用 Xavier 初始化（Glorot 初始化）或 Kaiming（He）初始化。

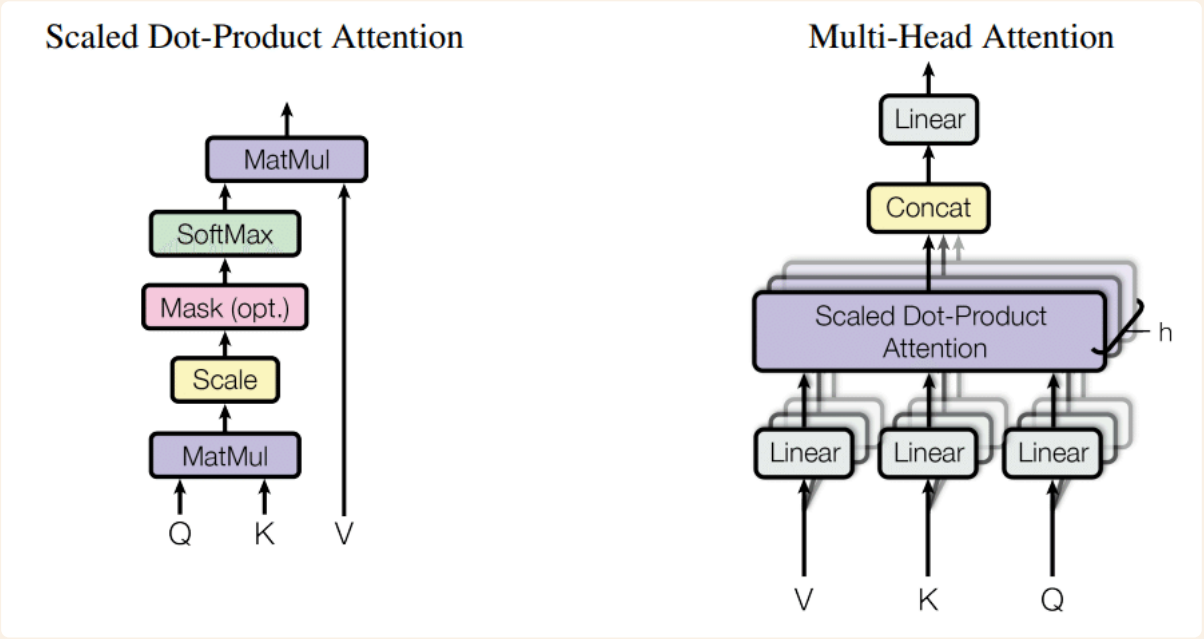
transformer 的编码器

编码器分为几个层，每一层都由以下几个函数组合而成

⭐ Multi-Head Attention 多头注意力

量化输入关键词之间关联性的技术

多头注意力机制是由缩放点积注意力机制（Scaled Dot-Product Attention）和多头注意力机制组合而成



缩放点积注意力（Scaled Dot-Product Attention）

实际上，缩放点积注意力机制的初始输入数据应该是前一步的输入嵌入 + 位置编码，而不是 Q、K 和 V。

下面介绍如何将前一步的**输入嵌入 + 位置编码转换为 QKV。**

输入维度大小：**T*d_model**

权重矩阵 **W_Q、W_K 、 W_V**（分别对应 Q、K 和 V）均使用随机数进行初始化。基于 Transformer 的模型会采用 Xavier 初始化或 He 初始化。

W_Q：d_model*dk

W_K：d_model*dk

W_V：d_model*dv

Q = X * W_Q shape:T*dk

K = X * W_K shape:T*dk

V = X * W_V shape:T*dv

dk==dv==d_model/head

Score = Q * K_transpose shape:T*T(每一个词和另外一个词之间关联性的矩阵，因此是一个方阵)

A == softmax{Score/sqrt(d_k)} shape:T*T，A[i,j]表示第 i 个单词对第 j 个单词的关系程度。

为什么计算点积就可以代表计算相似度呢?

余弦相似度。

实际上的 transformer 使用的是缩放点积注意力，而不是余弦相似度。

是的，有多篇相关论文和研究讨论了 Transformer 中的位置编码（Positional Encoding）和注意力机制中为什么使用缩放点积（Scaled Dot-Product）而非余弦相似度。下面我结合对话上下文（位置编码公式和点积 vs 余弦）列出关键论文，并简要解释其相关性。注意，原 Transformer 论文是核心来源，其他是扩展或分析。

> 1. **核心论文：Attention Is All You Need (Vaswani et al., 2017)**
> **相关性**：这是 Transformer 模型的原创论文，首次引入了你提到的位置编码公式（sin/cos 函数，10000 作为基数）。论文 Section 3.5 详细解释了为什么用 sin/cos 生成 PE（固定、非学习，确保不同位置独特，避免学习顺序依赖）。同时，Section 3.2 描述了注意力机制中的缩放点积（Q * K^T / √d_k），并在 ablation study 中比较了点积 vs 其他相似度（如 additive attention），解释点积高效且效果好（而非 cosine，因为点积 + 缩放已足够，且计算更快）。论文提到尝试过类似 cosine 的变体，但点积在实践中更好。
> **链接**：arXiv:1706.03762 或 NeurIPS 2017 论文。
> 2. **相关扩展论文：On Layer Normalization in the Transformer Architecture (Xiong et al., 2020)**
> **相关性**：讨论位置编码在 Transformer 中的作用，以及为什么不需额外余弦相似度归一化（因为 LayerNorm 已隐式处理模长）。它分析点积在高维下的方差问题，并确认缩放因子 √d_k 足以替代完整 cosine。
> **链接**：arXiv:2002.04745。
> 3. **分析论文：Understanding and Improving Transformer From a Multi-Particle Dynamic System Point of View (Lu et al., 2019)**
> **相关性**：解释注意力中点积 vs cosine 的理论基础，指出点积允许模型学习模长作为额外信号（而 cosine 强制归一化，可能丢失信息）。这与对话中“点积大表示高相关，但受模长影

> 响”的讨论一致。
> **链接**：arXiv:1906.02762。

假设你使用 Transformer 进行日语到英语的机器翻译。当你输入日语时，Transformer 会预测对应的英语翻译。当你查看预测的英语翻译和正确的英语翻译时，你可以计算出它们之间的差异。如果这个差异消失了，就说明模型运行良好。那么，我们如何消除这个差异呢？我们通过更新由随机数生成的权重矩阵 Q、K 以及最终整个 Transformer 的值来实现这一点，从而改进预测结果并减少差异。

Attention == A * V shape:T*d

矩阵变换为融合了上下文的信息的词嵌入?

> ⭐ 融合上下文的含义：
>
> - V 来自 X（原始嵌入），但加权后，每个输出行是“上下文增强”的嵌入。
> - 例子：位置 i (“人生”) 的 A[i,:] 高于相关词（如 “つまらない”），新向量 _i 融入 “无聊” 的语义，变成“无聊的人生”的表示

多头注意力机制：

权重矩阵 Q、K 和 V 在初始阶段使用随机数进行初始化。换句话说，如果初始值发生变化，权重矩阵的值也会随之改变。反之，这意味着如果我们使用多个用不同随机数初始化的权重矩阵来生成分数，我们应该能够生成一个比单个注意力机制考虑更多信息的分数。

并行多头后，最后拼接完毕之后，需要使用随机数初始化一个新的权重矩阵 Wo，并与水平连接的 Heads 进行内积运算，使得最终输出的大小与 d_model 的大小相同，即多头注意力机制的输入大小。

$$\text{output} = \text{concat}(H) \cdot W_O$$

解码器

和编码器的唯一区别：多头注意力机制被掩码多头注意力机制所取代。

解码器的作用是预测下一个关键词的概率。

> 考虑训练一个日语到英语的机器翻译任务。假设其中一个训练数据集包含“I am a cat”和“I am a cat”。编码器使用日语作为输入，解码器使用英语作为输入。在这种情况下，当解码器学习到“I am a cat”时，它首先输入“I”，并预测下一个关键词（“am”是正确答案）。然后，“I am”成为解码器的下一个输入，它再次预测下一个关键词（这次，“a”是正确答案）。

观察这张图，你能看出缩放点积注意力机制的掩蔽步骤发生在哪个阶段吗？如图所示，掩蔽操作使用掩蔽矩阵 (M) 进行。具体来说，首先计算 Q 和 K 的转置矩阵的点积，然后乘以 dk 进行缩放，最后在执行 Softmax 函数之前加上掩蔽矩阵 M。M 的矩阵形式如下：

$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

以“我是一只猫”为例，矩阵变为如上所示的 4x4 矩阵。观察第一行，只有第一列为 0，其余列均为负无穷大。这表明关键词“我”仅指代它自身。接下来，在第二行，前两列均为 0，因此“是”仅指代它前面的“我”以及它自身。（使用 -∞因为 Softmax 函数会将接近负无穷大的值转换为 0。通过转置 Q 和 K 计算出的未来分数无法被参考。）

另一方面，我们还没有解释它的工作原理，所以你可能还会有诸如“它是如何进行预测的？”或“它是如何学习的？”之类的问题。我们将在下一节中探讨这些问题。