3. Transformer基础

本章回顾了深度神经网络（DNN）在自然语言处理（NLP）中的应用，重点介绍了Transformer结构。内容包括分词（tokenization）、词向量（word2vec）、注意力机制（Attention Mechanism）、Transformer内部各组件的作用、Transformer结构的演变。并具体介绍了以GPT为代表的仅解码器模型（Decoder-only models）、仅编码器模型（Encoder-only models）和编码器-解码器模型（Encoder-Decoder Models）。此外，还提及了GLM和GPT结构上的差异。

2024版本：唐杰、杜晋华

统筹：**于竣丞**；初步成稿：**于竣丞，武文静**；修改完稿：**吴彦辰**；稿件审校：**吴彦辰**

## 3.1 引言

### 3.1.1 预备知识

#### 3.1.1.1 Token与Tokenizer

Token是大语言模型中输入、输出以及中间状态(hidden state)变量的基本单位，是词语进行离散化表示后的基本语言单位，根据分词器（tokenizer）所用的码本（codebook）的不同，一个词可能被离散化为不同长度的token sequence，比如"apple"在某个码本中也许被分词为['a', 'pp', 'l', 'e']，分词后长度为5个token，在另一个码本下则可能被分词为['app', 'le']，长度2个token。

Tokenizer是将词或者句子分割成由token所组成list的组件，一般来说获得一个tokenizer不需要训练神经网络，但有可能涉及到损失计算以及优化算法，如EM算法，也有基于词频的tokenizer如BPE算法，在后文中，我们会逐一介绍这两种算法。

#### 3.1.1.2 Codebook

Codebook是将token和向量对应起来的词典，给定一个token sequence，如上文的['a', 'pp', 'l', 'e']，这些token会分别对应着码本中的一个索引，则实际可以将token sequence表示为一个由索引组成的列表，如['100', '200', '300', '10004', '20003']，通过在codebook中索引对应位置可以获得每一个token对应的向量化表示，从而向量化token sequence。举例而言，一个codebook一般被定义为一个形状为[codebook\_len, token\_dim]的可学习参数矩阵（或者如果你熟悉深度学习，这里应该称为张量），则我们刚才索引出的向量化token sequence的形状为[5, token\_dim]，其中每一个token被长度为token\_dim的向量所描述，共有5个token。

#### 3.1.1.3 Query, Key, Value

在介绍Self Dot-Product Attention的部分中，我们会引用Query, Key以及Value的概念，比较重要的概念是Attention矩阵中每一个Query token 索引下的权重分布是一个“和为1”的分布，比如Query的token长度为10, Key的长度为20，则Attention score矩阵K的形状为[10, 20]，其中K[i, :].sum() = 1 对于 descript 均成立。定性的理解Attention score矩阵，其所表示的是Key的每一个token对Query的每一个token位置输出结果的贡献权重， 此外，经过attention操作后，输出矩阵的token len取决于query的token len。

### 3.1.2 大语言模型运行概括

大语言模型的推理过程可以简单概括为几个步骤：分词、向量化、通过transformer模型推理，分类，在分类头的最后，我们可以获得下一个token在codebook上的概率分布，之后通过预定义的temperature, topk等超参数，我们可以具体索引出下一个token，将其补充进输入中，并迭代上述过程，就是大语言模型在推理的过程了。

在本章中，我们将逐个介绍上面的每一个组件和步骤，从而对大语言模型的整体架构有基础的认知，需要注意的是本章不聚焦于训练大语言模型的过程，而是希望读者能先认识整个大语言模型的结构，大语言模型的训练过程分很多阶段，相对于传统深度学习方法的训练更加复杂，我们会在后面几个章节中进行介绍。

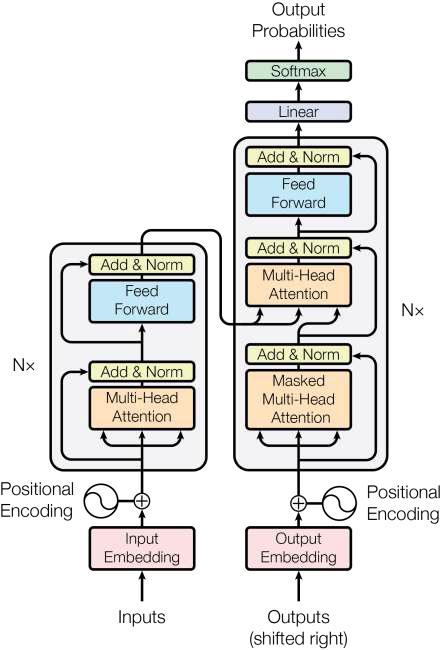


图1. Transformer结构示意图

## 3.2 分词（Tokenization）

首先我们考虑语言模型的定义，语言模型是一系列token的概率分布，其中每个token都来自同一个codebook。一个最简单的例子，假设我们的码本中只有['the', 'mouse', 'ate', 'the', 'cheese']这5个token（这也意味着我们的数据集经过分词后不能超出这五个token的类别），那么我们可以将next token的表达为一个shape为[5]的概率分布，其中第i个元素代表了在下一个token中，出现第i个token的概率，比如当我们获得一个离散概率分布p=[0.1, 0.1, 0.1, 0.7, 0]时，这就以为next token的位置更可能出现codebook中第四个token: 'the'。

然而，自然语言并不是以token sequence（token序列）的形式出现的，而只是一个字符串（更具体来说，是 Unicode 字符序列），比如"the mouse ate the cheese"，实际上，自然语言并不会天然被分割成离散的token，那么我们就需要一种算法（或者称一种组件）来将自然语言分割成离散的token，它就是tokenizer。一个tokenizer可以将任何在数据集中的字符串转换为token序列，当然在大语言模型的情境下，因为现代大语言模型几乎使用了互联网上所有的数据，这里的“在数据集中的字符串”也就几乎可以理解为所有字符串。

### 3.2.1 什么是好的tokenizer？

考虑最简单的tokenizer，就是将文本基于空格进行分割

这样给定自然语言"I went to the store"，它将会被划分为token sequence ['I', 'went', 'to', 'the', 'store']，在这种情况下，看起来我们可以通过定义一个和“英语词典”一样长的codebook来token化所有字符串。但现在回想使用大语言模型的场景，是不是几乎所有大语言模型都可以完成翻译的工作？也就是说，除了英文，其他语言的tokenizer也应该被考虑，于是基于空格分割的tokenizer就会有一定的问题，比如在中文中“*我今天去了商店*”，实际上我们无法基于空格给出分割；再比如德语中较长的复合词“*Abwasserbehandlungsanlange*”，实际上也很难用空格进行分割；即使在英文的语境中也有一些带连字符的单词（例如 father-in-law）和缩写（例如 don’t），这些单词应该分开。例如，Penn Treebank 将 don’t 拆分为 do 和 n’t，这是一个符合语言学原理但并不是可以说“显然”的选择。因此，用空格来分割单词来识别单词是相当困难的。

那么什么样的tokenizer是我们所希望的呢，也就是什么tokenizer是更好的呢？这里有几个原则可以参考：

我们不希望太多种token，比如极端情况中，我们把每个字节和字符作为独立的token，这样会让token序列将变得难以建模。

我们也不希望token太少，否则否则单词之间就不会有参数共享，也就是说在embedding空间（高维向量空间）中两个相似的词是独立的两个token，而不是由类似的tokens组成两个类似的token sequences，比如说，mother-in-law和father-in-law实际上不应该是完全独立的两个token，我们希望在tokenizer的过程中也保留他们之间一定的相关性。这件事情对语言学上形态丰富的语言（例如阿拉伯语、土耳其语等）来说尤其成问题。

我们还希望每个token都是具有语言学或统计学意义的单位。

这样的tokenizer实际上需要一定统计的方法获得，接下来我们将介绍几种常见的tokenizer。

### 3.2.2 BPE分词器（Byte Pair Encoding Algorithm）

Byte Pair Encoding算法（BPE算法）是由Sennrich等在2015年提出的一种数据压缩算法，是目前最常被用作tokenizer的算法之一。在得到BPE tokenizer的过程中，直观来讲，我们从每一个字母开始['a', 'b', 'c', ...]，让每一个字母自己成为一个token（初始化），随后合并那些常出现（词频更高）的token组合。整个算法流程可以概括为：

* 给定一个训练语料库作为输入，其中包含了非常多的字符序列（自然语言）作为数据。
* 我们初始化codebook为字符集合$V$，如在纯英文语料库中初始化['a', 'b', 'c', ..., 'z']为初始codebook。
* 随后我们迭代如下的过程
  + 找到在数据中出现次数最多的元素组合$x,x^' \in V$
  + 用一个新的符号 $x \text{ } x^'$代替所有数据集中出现的$x,x^'$
  + 将新符号$x \text{ } x^'$加入（append）到$V$中

举例而言，假设我们现在有一个语料库，包含三个字符序列,"the car", "the cat", "the rat"，则我们运行BPE算法如下：

初始化：  
1. [t, h, e, ␣, c, a, r], [t, h, e, ␣, c, a, t], [t, h, e, ␣, r, a, t]

我们不难发现，[t, h]的字符组合出现了3次，是字符组合中出现次数最多的，因此我们合并[t, h]为[th]

1. [th, e, ␣, c, a, r], [th, e, ␣, c, a, t], [th, e, ␣, r, a, t]

接下来，[th, e]的字符组合同样出现了3次，于是我们合并[th, e]为[the]

1. [the, ␣, c, a, r], [the, ␣, c, a, t], [the, ␣, r, a, t]

最后，我们发现[c, a]也出现了2次，于是合并[c, a]为[ca]，有

1. [the, ␣, ca, r], [the, ␣, ca, t], [the, ␣, r, a, t]

我们无法找到可以进一步合并的字符组合，因此迭代停止，我们因此获得了codebook：

[a, c, e, h, t, r, ca, th, the]

于是当有一个新的词"the ox"出现，这个tokenizer就可以将其tokenize为

[the, ␣, o, x]

如上的BPE算法还有一个问题，就是Unicode的字符有太多了（144，697个），这个问题在多语言的情境下尤为突出。我们几乎可以确定，不能在训练的数据中见到所有的字符。为了进一步降低数据稀疏性，Wang et al.在2019年提出可以对字节而不是 Unicode 字符运行 BPE。以中文为例，给定输入文本“今天”，其将会被tokenize为

[x62, x11, 4e, ca]

这种对bytes进行BPE的方法在目前更为常用。

### 3.2.3 SentencePiece工具

除了像BPE那样按照词频进行tokenize之外，一种更“有原则”的方法是通过定义一个目标函数来评估一个好的tokenizer是什么样子的。

给定一个序列$x\_{1:L}$，则tokenization T是一个集合，满足

\[

p(x\_{1:L}) = \underset{(i,j)\ in T}{\Pi} p(x\_{i:j})

\]

descript

举例而言，当我们的训练数据仅包括一个sequence：ababc，给定某种码本V={ab,c}该token sequence经过tokenize之后的sequence是T = {(1, 2), (3, 4), (5, 5)}，也即{ab, ab, c},则从似然概率考虑，其似然函数为

\[

p(x\_{1:L}= \frac{2}{3} \cdot \frac{2}{3} \cdot \frac{1}{3}) = \frac{4}{9}

\]

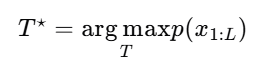
descript

则这种tokenizer希望去最大化似然函数，也就是

\[

T^\star = \underset{T}{argmax} p(x\_{1:L})

\]



Unigram model使用EM algorithm来优化上述过程，算法的流程可以概括为：

* 初始化一个相当大的codebook V
* 重复下述过程
  + 给定V，使用EM algorithm优化 $p(x)$和$T$
  + 计算loss(x)，也就是给定codebook V中的每一个$x \in V$，loss（x）评估了当该x被移除以后，似然函数会下降多少。定性来讲，如果该x移除后似然显著下降，则loss很大，反之如果影响不大，loss很小
  + 按照loss进行排序，保留V中loss较大的80%的token。定性来讲，这样去除了对likelihood影响不大的tokens。

### 3.2.4 两种tokenizer的对比

常见的大语言模型GPT-2和GPT-3使用的tokenizer是基于BPE的，其codebook size为50K个token，另外有一种大语言模型，叫做Jurassic，它使用的是基于SenstencePiece的tokenizer，其codebook的size是256K个token。

基于SentencePiece的tokenizer的主要优势在于，当给定同样的输入字符串，Jurassic比起GPT-3可以在token长度的层面少输入28%的token，所以对于同样的输入字符串，实际上他的速度可以达到1.4倍。

当考察同样最长上下文的GPT-3和Jurassic时，Jurassic可以输入相对GPT输入文本1.39倍长的文本，意味着其可以接受更长的提示词（prompt）。

以下给出GPT-3和Jurassic的tokenization的例子。

对于同样的输入"Abraham Lincoln lived at the White House."，他们tokenizer的输出分别是：

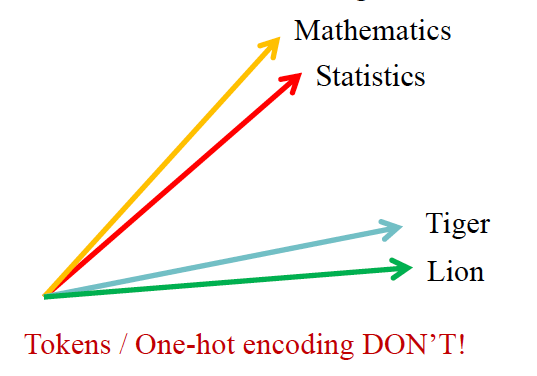
* GPT-3: [Ab, raham, ␣Lincoln, ␣lived, ␣at, ␣the, ␣White, ␣House, .]
* Jurassic: [Abraham␣Lincoln, ␣lived, ␣at␣the␣White␣House, .]

## 3.3 Token的向量化（Vector representations）

在将自然语言转化为离散的tokens以后，我们可以用[1, 3, 2, 40, 100]这样的token sequence来表达自然语言的字符串。下一个问题是，我们如何将这样的token sequence转换到以向量表达的表征空间中。

还是考虑最简单的一种表达方法，假设我们有一个大小为5的codebook，则最简单的表达该codebook中的每一个token的向量方法就是one-hot方法，也就是说每一个token是一个长度为5的向量，其中第一个token以[1, 0, 0, 0, 0]表示，第二个token以[0, 1, 0, 0, 0]表示，以此类推，也就是说向量的每一个维度表示词表中的一个词，对应于该词的维度是1，而其他维度是0。

one-hot当然是一种直观，简单的向量化方法，然而one-hot很难以表达词的一个很重要的性质，也就是有类似词义的词应该也有类似的表征，也就是相似度更高的向量表达。



比如在上图的场景中，实际上statistics和mathematics更加相似，而tiger和lion更加相似，然而在利用one-hot的方法表示他们时，statistics、mathematics、tiger、lion这四种token在向量的表达上并不体现出区别，彼此两两之间的区别也都相同——从向量余弦相似度上讲，都是0。如tiger的表示是[1, 0, 0, 0, 0]，而lion的表示是[0, 1, 0, 0, 0]，他们之间的余弦相似度为0，然而实际上他们具有语义上的共性，但在向量表示中没有表达出且不能在训练过程中逐渐学习出这样的共性。

在深度学习领域中，研究者们更喜欢分布式的表征，也就是概念和神经元之间存在着多对多的对应关系，其中的关键主要有两点：其一是一种概念可以被多种神经元的激活表示，另外是一种神经元可以表达多种概念。依照于这样的原则，在大语言模型中，每一个词往往被表达为一种dense的且以实数表达的低维向量表示。这里的低维是一种相对的概念，一般并不是在向量长度上绝对值低。

于是我们希望依照上面所述的原则学习出词表中每一个词的分布式表征，其中的训练数据往往是没有经过标注的文本，比如维基百科的文本。也就是说，这里的训练数据一般不具有对应关系，没有一定要求一问一答的特性。主流的方法有两种，连续词袋模型和Skip-gram模型，对于每一种模型，又有两种学习的方法，一种是hierachical softmax，另一种是negative sampling。尽管当下的大模型一般不会利用如word2vec那些早期向量化方法对词表进行预训练，而是利用蕴含在预训练数据集的大量数据中的知识来训练codebook，早期的方法还是在理论方面有很深的影响，因此先介绍他们。

### 3.3.1 连续词袋模型（Continous Bag-of-Words Model）

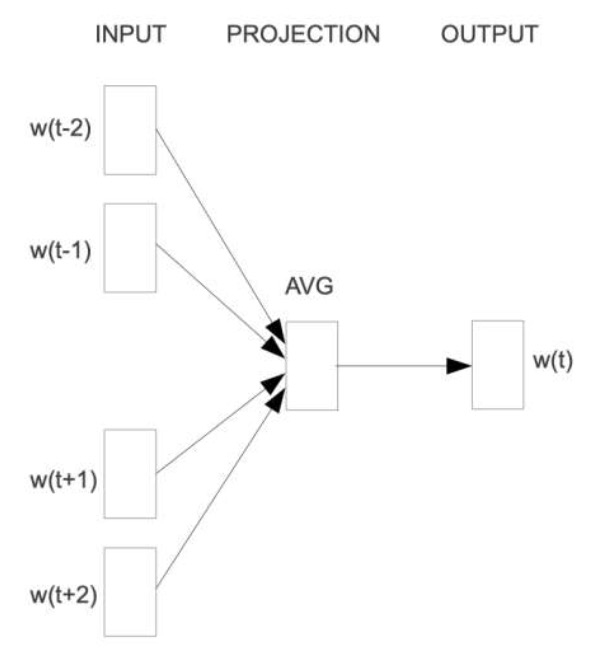
连续词袋模型基于上下文预测当前词，如下图表示的算法，给定一个输入的token sequence $\in \mathbb{R}^{L \times D}$，其中L是token sequence的长度，D为表示codebook中每一个token的向量的长度。于是考察一个L=5的token sequence，考察第三个token，我们通过取一定长度$K=4$上下文窗口中的上下文的平均值来表达该位置的上下文$C\_t$，也即

\[

C\_t = \frac{1}{K} \Pi\_{w\_i \in C\_t} u\_{w\_i}

\]

descript



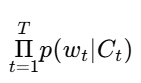
定性来考虑，这样的模型可以建模常出现在一句话中的token之间的相似度关系，令更常同时出现的token的向量表示具有更相似的表达。该模型被称为词袋模型，因为上下文中的单词顺序不会影响上下文表示的投影。

连续词袋模型的目标函数是最大化如下条件概率

\[

\underset{t=1}{\overset{T}{\Pi}}p(w\_t|C\_t)

\]



这里的条件概率可以用指数能量函数（softmax）表示，满足

\[

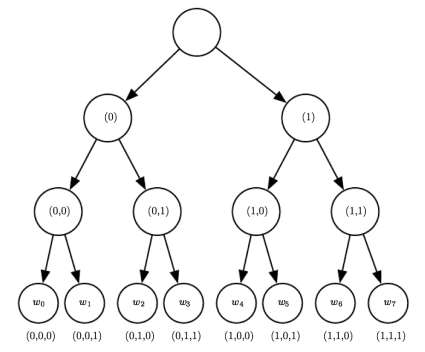
p(w\_t|C\_t) = \frac{exp(v\_{w\_t}\cdot C\_t)}{\Sigma\_{i=1}^{|V|}exp(v\_{w\_i} \cdot C\_t)}

\]

descript

连续词袋模型需要扫描词表中的所有词以得到概率。

我们也可以用hierarchical softmax来计算这里的条件概率，首先为词汇表中的所有单词构建一棵二叉树，以单词为叶节点，左子树编码为1，右子树编码为0，则每个单词都可以被赋予一个从根节点到叶节点的唯一编码。



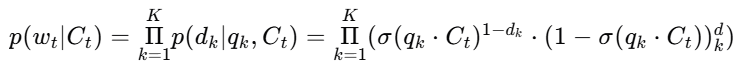
则这里的条件概率可以表达为：

\[

p(w\_t|C\_t) = \underset{k=1}{\overset{K}{\Pi}}p(d\_k|q\_k,C\_t) \\

= \underset{k=1}{\overset{K}{\Pi}}(\sigma(q\_k \cdot C\_t)^{1-d\_k} \cdot (1- \sigma(q\_k \cdot C\_t))^d\_k)

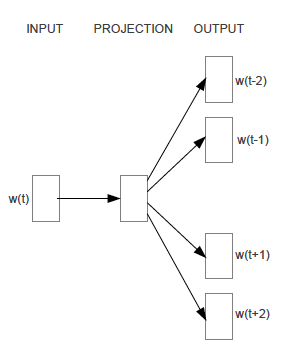
\]



其中$\sigma$为sigmoid函数，$q\_k$是从根节点到叶子节点路径上第k个节点的向量表示，$d\_k$是对应于$q\_k$的编码。

### 3.3.2 Skip-gram Model

Skip-gram 模型通过当前词来预测上下文，如下图表示

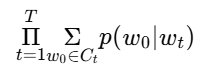


上下文同样是取一定长度上下文窗口中的所有词。在skip-gram模型中，目标函数是最大化如下条件概率。

\[

\underset{t=1}{\overset{T}{\Pi}}\underset{w\_0 \in C\_t}{\Sigma} p(w\_0 | w\_t)

\]



接下来我们讨论CBOW和skip-gram哪一个性能更好，在NLP领域中，一个普遍的共识是CBOW的word embedding不如skip-gram的embedding。但最近研究人员发现，这种观点也许应该归因于原始论文和标准软件库中错误的 CBOW 实现，而正确实施 CBOW 可以生成在各种任务上与 skip-gram 完全可以匹敌的word embedding，同时训练速度是 skip-gram 的三倍以上。

现代大语言模型中，codebook中的word embedding 往往通过在大规模的无标签数据（预训练数据集）中进行无监督学习来学习token的向量表示，一般不会先用如上的方式先进行embedding的初始化。

## 3.4 建模Token序列中的隐含关系（Sequence Modeling）

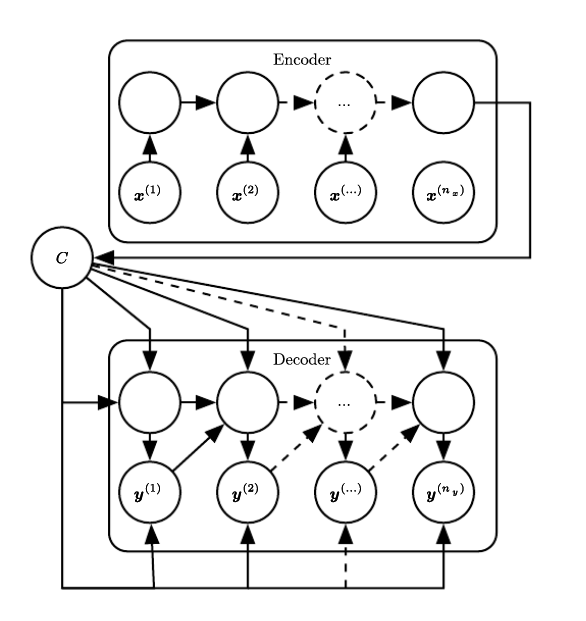
我们已经知道如何向量化一个token，使他们被映射到一个连续的表征空间中，从而使得表达token与token之间的语义相似性成为一种可能。进一步的，我们的每一个句子，或者词语，都往往由多个token组合而成，对于这样多个token组成的序列，要如何对他们进行建模是我们下一个要解决的问题。而由于我们无论如何都会将token按顺序排列为一维的序列，因此这里提到的序列建模方法都是隐式的，也就是说，我们不会再像token向量化那样把某个序列量化为某种数学形式，而是希望模型能够学习到序列中token与token之间的隐含关系，接下来我们来介绍这种隐含关系的表达是如何演进到如今的Multi-Head Attention的。

### 3.4.2 Encoder-Decoder结构的RNN

现在，考虑我们在日常生活中使用的大语言模型，定性来看，模型会接受用户的一段话（被称为提示词，prompt）作为输入，而输出一段话，如上面这样使用语言模型，给定一段话，输出另一段话，或者从token的角度说，输入一个token sequence，输出另一段token sequence，这种任务叫做Seq-to-Seq。

其中，可变长的输入和输出对Seq-to-Seq任务具有非常重要的意义，因为输入的内容和希望模型输出的内容往往具有**不一样的长度**，回想在使用大模型时的经历，是不是模型经常会输出比我们问题更长的回答？

由于我们在之前已经描述了一段话被分词并排列成token sequence的过程，这时我们应该以一个更加专业的角度看待这个问题，考虑一个机器翻译（machine translation）问题，假设我们有三个英文单词"Large Language Model"要被翻译成中文，则输出的中文应该是"大语言模型"。我们假设输入的英文"Large Language Model"被分为9个token，其对应的输出是UTF-8编码后的字节序列（GLM4中所使用的中文分词化方法）有3个token。则我们所期望的就是给定一个长度为9的输入，要求一个长度为3的输出，这样的任务要求我们有一个可接受变长输入又可变长输出的模型，于是有了Encoder-Decoder结构的RNN，整体结构如下图所示。

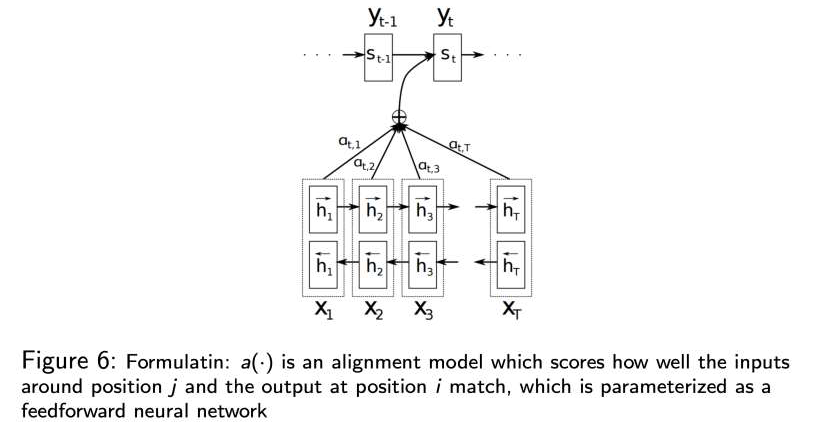


其中的bottle neck C是一个固定大小的向量，这样Encoder-Decoder的结构可以被看作为一个sequence到固定向量的RNN以及另一个固定向量到sequence的RNN，这样两个RNN的组合。

这样结构的问题在于表征能力有限，一个bottle neck的特征要去总结表达句子中的所有信息，这是一件很难的事情，于是有了让中间的bottle neck也是变长序列的方法，就是Bahadanau Attention。

### 3.4.3 Bahdanau Attention

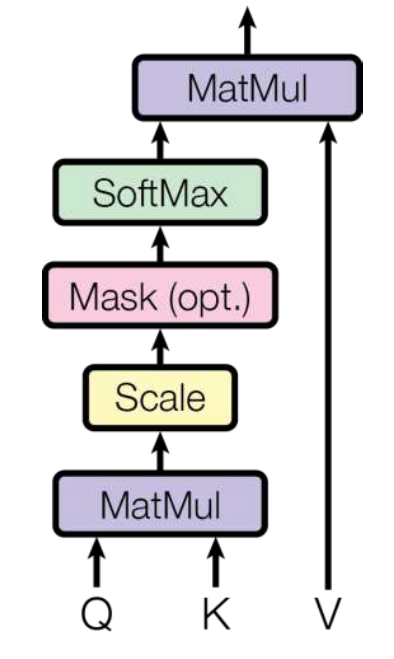
由于Encoder-Decoder结构RNN中的C经常很难总结一个很长的sequence，Bahdanau et al.提出将C建模为一个可变长的sequence，而不是一个定长的sequence。此外，他们也提出了第一种attention技术——Bahdanau Attention，也被称为Additive Attention。



他的问题在于并行化程度差，还是要串行处理，因此导致计算的效率非常低。问题到此还没有完全解决。

### 3.4.4 Self Dot-Product Attention

Self Dot-Product Attention在这样的背景出现，可以有效解决上面的并行问题。他也是现代大语言模型中几乎可以说最关键的组件，它的基本结构可以概括为下图。



Attention是求values的加权和，其中给每一个value的权重是由query和key经过一个函数进行计算得到。

\[

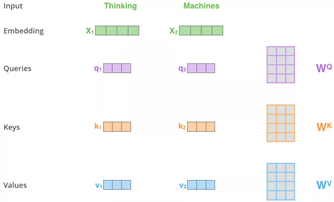
Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d\_k}})V

\]

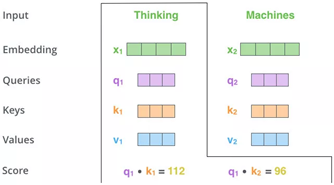
descript

其中$Q, K \in \mathbb{R}^{L \times d\_k}$, $V \in \mathbb{R}^{L \times d\_V}$，其中点积的尺度会随着维度的增大而增大，因此注意力得分会根据向量的大小进行缩放。

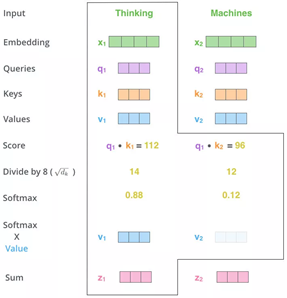
假设我们现在有两个词：thinking和machines，其中每一个词都由一个长度为d的向量来表达时，q,k,v可以由他们经过一个线性变换（Linear层）定义，也就是通过乘以三个独立的矩阵$W^Q, W^K, W^V$，



通过对$(q\_1,k\_1)$以及$(q\_2,k\_2)$求点积，我们可以得到attention score

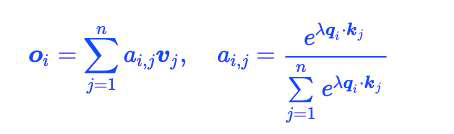


在进行normalizatin以及softmax以后，我们可以得到归一化之后、类似于权重分布的attention score，最后通过乘以values，我们可以得到z

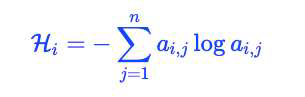


接下来我们详细的推导下为什么要在scale中除以$\sqrt{d\_k}$，在Self Dot-Product Attention中，我们可以表达第i个token位置的输出o\_i以及attention score矩阵中第i, j个位置的元素为

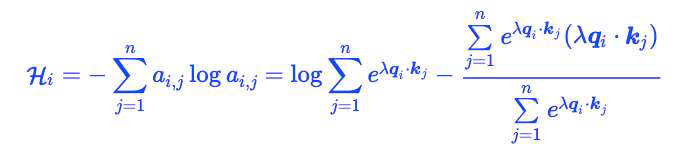
o\_i = \sum\_{j=1}^{n} a\_{i,j} v\_j, \quad a\_{i,j} = \frac{e^{\lambda q\_i \cdot k\_j}}{\sum\_{j=1}^{n} e^{\lambda q\_i \cdot k\_j}}



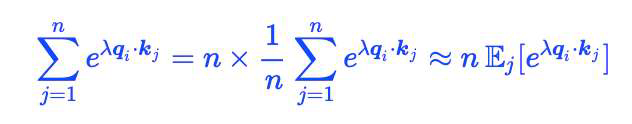
考察attention score矩阵的熵，应有

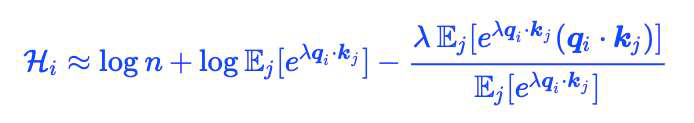


显然它对n是敏感的，会随n的变化而由尺度的变化。然而实际上，$H\_i$是不该对n敏感的，因为在加入新的token（此时n上升）时，现存的token还是应该被同样关注的。因此，我们需要找到一种$a\_{i, j}$，使得$H\_i$对n不在敏感。我们展开熵的表达式，可以得到如下公式。

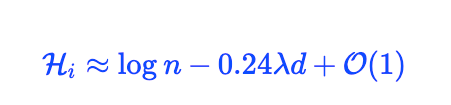


我们需要确定一个合适的$\lambda$来抵消长度带来的影响，于是继续计算上式，有





使用拉普拉斯方法进行估计，有



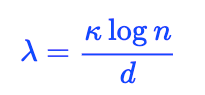
为了取抵消n的影响，我们希望

descript

于是有

descript

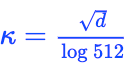
这里引入超参数k用于估计，有



则attention score可以表达为



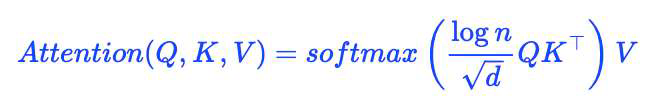
假设n=512，我们令k满足



于是有



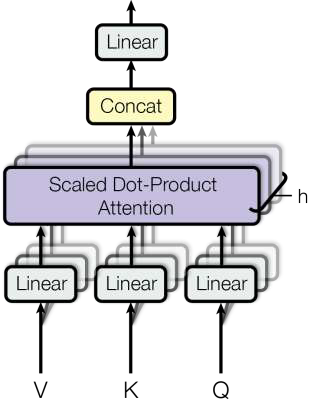
实际上，除以$\sqrt{d\_k}$并不是必要的，在ACL 2022的一篇研究中（Overcoming a Theoretical Limitation of Self- Attention）给出了如下公式



但是Self Dot-Product Attention可以看作是一种给定权重的平均，和卷积操作相比，他不能对不同的位置有不同的线性变换，这会限制它的表征能力。



因此，Multi-Head Attention被提出。



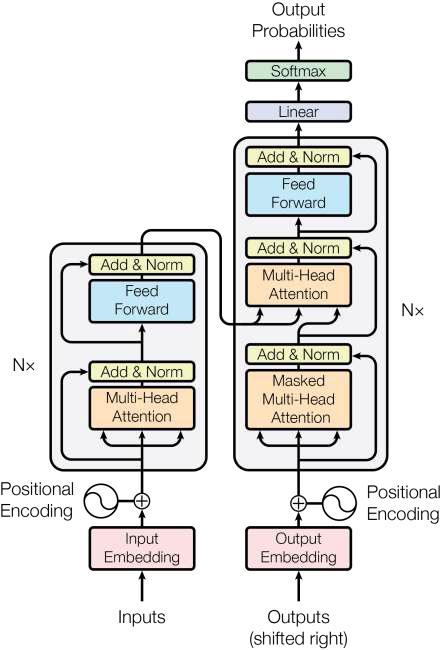
其中多个attention层（head）是并行计算的，每一个head都表示一种独立的线性变换，因此不同的head可以学习到不同的关系。

## descript

## 3.5 Transformer的基本结构

在3.4节中，我们介绍了Multi-Head Attention的原理以及来源，MHA可以说是Transformer中最重要的组件，在本节中，我们将详细介绍Transformer以MHA为核心建立的整体架构以及其他重要的组件。

标准 Transformer 有一个Encoder和一个Decoder，每个Encoder和Decoder由数层transformer layer构成。每个 Transformer 层都有两个子层，一个多头自注意力（MHA）和一个全连接前馈网络（MLP）。一般来说，我们会在两个子层之间采用残差连接，然后进行Layer Norm。每一个子层的输出都是LayerNorm(x + Sublayer(x))。



### 3.5.1 Layer Norm

Batch Normalization可以显著减少前馈神经网络的训练时间。然而，Batch Normalization的效果取决于小批量的大小，在循环神经网络中的应用并不广泛。

Layer Normalization 在RNN中更加广泛，其通过学习计算同一层中所有隐藏单元的统计数据，满足

\[

y = \frac{x-\mathbb{E}[x]}{\sqrt{Var[x] + \epsilon}} \cdot \gamma + \beta

\]

descript

其中$\gamma$和$\beta$是可学习的参数。

### 3.5.2 Feed-Forward Networks

全连接的前馈神经网络对每一个layer是独立的，也就是每一个layer都有一个MLP的实例，其中包括了两个线性变换（Linear层）+ReLU激活函数，满足

\[

FFN(x) = max(0, xW\_1+b\_1)W\_2 + b\_2

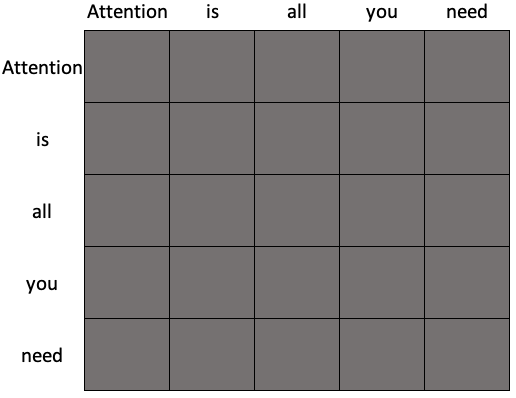
\]

descript

其中$W\_1 \in \mathbb{R}^{d\_{model} \times d\_{ff}}$, $W\_2 \in \mathbb{R}^{ d\_{ff} \times d\_{model}}$, $b\_1 \in \mathbb{R}^{d\_{ff}}$, $b\_2 \in \mathbb{R}^{d\_{model}}$。

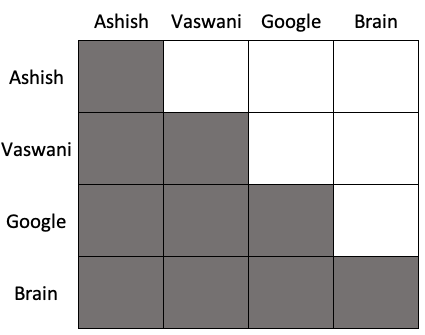
### 3.5.3 Encoder

在Encoder中，所有k、q和v一般来说都来自于上一层的输出，在Encoder中一般采用双向的attention mask，也就是每一个位置的token都可以注意到所有的位置。



### 3.5.4 Decoder

在Decoder，所有k、q和v一般来说都来自于上一层的输出，在Decoder中一般采用因果Mask下的Attention score（一个下三角矩阵），其中每一个位置只能注意到该位置之前的位置，attention score矩阵如下图

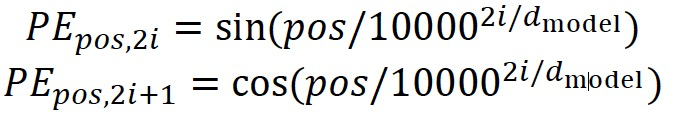


### 3.5.5 Encoder-Decoder

在Encoder-Decoder中，q来自于过去的decoder层，k和v一般来说都来自于上一层的encoder层的输出，这里也采用双向的mask。

### 3.5.6 Positional Embedding

注意力机制与位置无关（因果掩码除外）。为了利用序列的顺序，我们通过在Encoder和Decoder底部的输入embedding中添加“位置编码（positional embedding）”来注入标记的绝对位置，经典的绝对位置编码方法是正余弦编码方法，符合如下公式



对于任何固定偏移量 k，𝑃𝐸\_{𝑝𝑜𝑠+𝑘} 可以表示为 𝑃𝐸\_{𝑝𝑜𝑠} 的线性函数。

此外，位置嵌入也可以通过可学习参数（nn.Parameter）定义，并在训练中学习positional embedding。

## **3.6 常见语言模型结构：自回归模型、仅编码模型与编码-解码模型**

#### **3.6.1 自回归模型（Auto-regressive Models）**

自回归模型，通常称为解码器-仅模型（Decoder-only Models），依赖于前文信息逐步生成下一个词。该模型的核心思想是通过前面已经生成的词（例如 x1:i）来预测下一个词 xi+1，并且这种预测是逐步进行的，直到整个文本生成完成。

**示例：GPT（Generative Pretrained Transformer）**

以GPT为例，GPT是一种自回归模型。其训练流程包括两个阶段：首先是对大规模未标记文本进行生成式预训练，接着进行任务特定的微调。GPT的核心结构是多层Transformer解码器。通过最大化在给定前文的条件下预测下一个词的概率来训练模型。GPT生成的文本具有高度的连贯性，可以用于诸如文本补全等任务。

例如，给定部分文本：[[CLS], the, movie, was]

GPT模型会预测出下一个词并继续生成文本：great。

**训练目标：**

自回归模型的训练目标是最大化给定前文的情况下，预测下一个词的条件概率，即： max θ∑tlog⁡P(xt∣xt−k,…,xt−1)\text{max} \, \theta \sum\_{t} \log P(x\_t | x\_{t-k}, \dots, x\_{t-1})maxθ∑t​logP(xt​∣xt−k​,…,xt−1​)

#### **3.6.2 仅编码模型（Auto-encoding Models）**

仅编码模型，通常是编码器-仅模型（Encoder-only Models），其输入序列经过编码器处理后得到一个上下文嵌入表示。与自回归模型不同，这种模型并不生成文本，而是专注于理解输入序列的表示，广泛应用于分类任务等。

**示例：BERT（Bidirectional Encoder Representations from Transformers）**

BERT是一个典型的仅编码模型，采用了双向Transformer架构。在预训练阶段，BERT使用“掩码语言模型”（Masked Language Model，MLM）任务，即随机掩盖输入中的一些词，并要求模型预测这些被掩盖的词。通过这种方式，BERT能够捕捉到双向上下文的依赖关系。BERT在许多任务中表现出色，特别是在问答、文本分类等任务上。

例如，输入文本： [[CLS], the, movie, was, great] 经过BERT编码后，模型预测文本情感： positive。

**训练目标：**

BERT的训练目标主要包括两个任务：

1.**掩码语言模型（Masked Language Modeling，MLM）：** 随机掩盖15%的词，要求模型根据上下文预测被掩盖的词。

2.**下一个句子预测（Next Sentence Prediction，NSP）：** 预测两个句子是否相邻。

#### **3.6.3 编码-解码模型（Encoder-Decoder Models）**

编码-解码模型将输入文本通过编码器处理后，生成一个上下文嵌入表示，接着通过解码器生成目标文本。它不仅可以理解输入信息，还能生成对应的输出，广泛应用于序列到序列（Seq2Seq）任务中，如翻译、摘要生成等。

**示例：T5（Text-to-Text Transfer Transformer）**

T5模型采用了一个统一的编码器-解码器架构，预训练时使用了跨度掩码任务（span-corruption），即将文本中的多个片段替换为[MASK]，然后要求模型根据上下文恢复这些片段。T5将所有任务转换为文本到文本的格式，使得它在多个NLP任务中都有良好的表现。

例如，对于表格到文本的生成任务，输入为： [name: clowns | eatType: coffeee shop] T5会生成对应的描述文本： clowns is a coffee shop.。

**训练目标：**

编码-解码模型的训练目标是使用输入的文本生成目标文本，并通过最大化生成文本的条件概率进行训练。

#### **3.6.4 总结**

**自回归模型（仅解码器）**

**优点：** 能够自然地生成文本，适用于生成任务（如文本补全）。

**缺点：** 只能单向依赖前文信息，不能利用上下文双向信息。

**示例：** GPT，适用于生成任务。

**仅编码模型（仅编码器）**

**优点：** 能够利用双向上下文信息，适用于理解任务（如情感分析、问答）。

**缺点：** 不适合生成任务。

**示例：** BERT，适用于理解任务。

**编码-解码模型**

**优点：** 既能理解输入，也能生成输出，适用于序列到序列任务（如翻译、摘要生成）。

**缺点：** 训练目标较复杂，需要特殊的任务设计。

**示例：** T5，适用于文本生成任务。