国家材料服役安全科学中心
National Center for Materials Service Safety

基于Transformer的大模型（LLM）

汇报人：李庚

2023年6月7日

汇报提纲

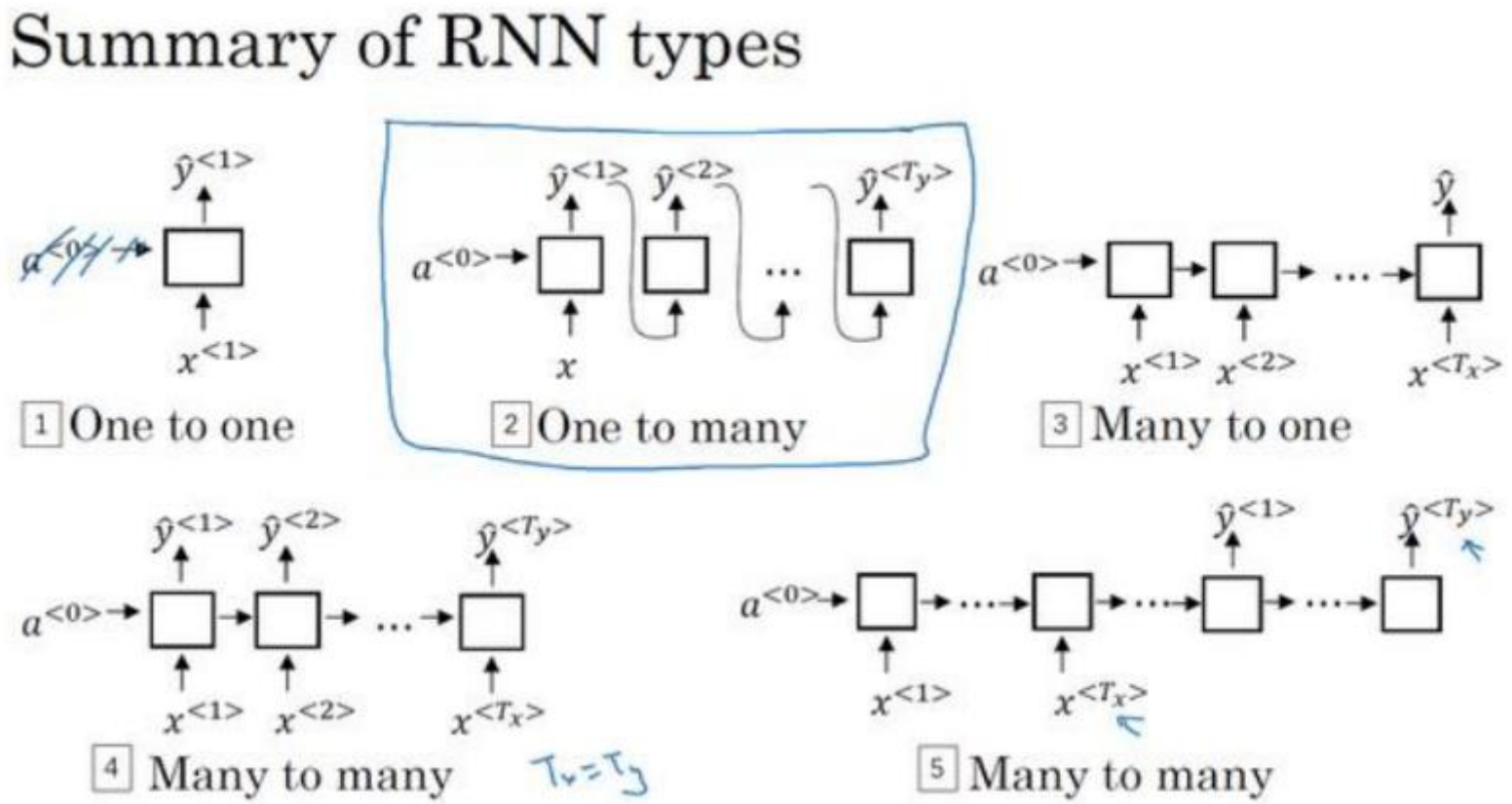
一、Transformer

二、基于Transformer的大模型(LLM)

1 Transformer

Transformer

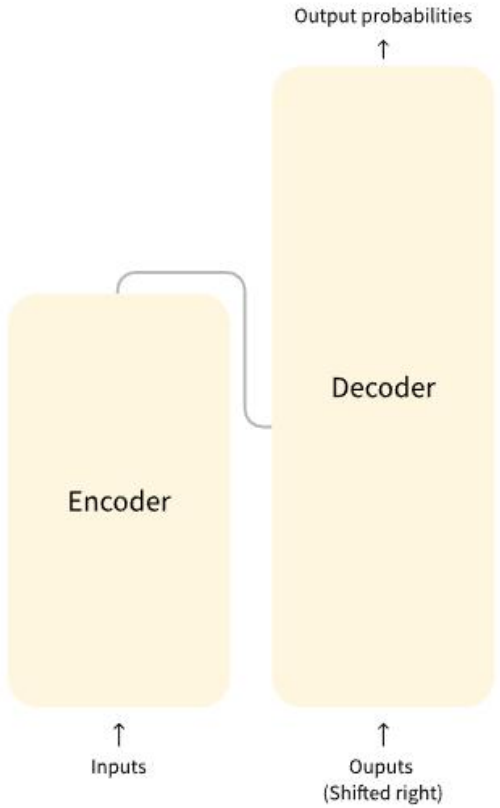
RNN主要的框架如下图所示：



Transformer

该模型主要由两个块组成：

- 编码器（左）：编码器接收输入并构建它的表示（其特征）。这意味着模型经过优化以从输入中获取理解。
- 解码器（右）：解码器使用编码器的表示（特征）和其他输入来生成目标序列。这意味着该模型针对生成输出进行了优化。



这些部分中的每一个都可以独立使用，具体取决于任务：

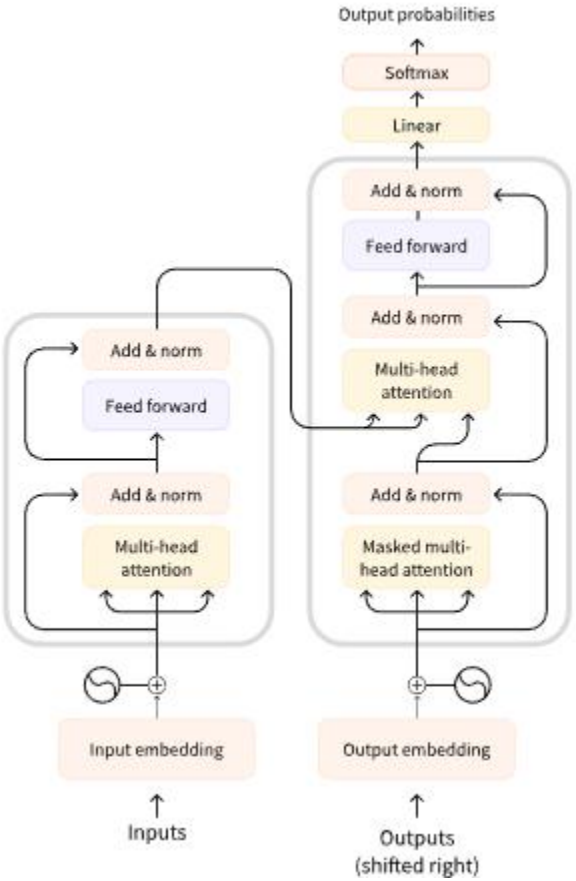
- 仅编码器模型：适用于需要理解输入的任务，例如句子分类和命名实体识别。
- 仅解码器模型：适用于文本生成等生成任务。
- 编码器-解码器模型或序列到序列模型：适用于需要输入的生成任务，例如翻译或摘要。

我们将在后面的部分中独立深入研究这些架构。

Transformer

Transformer 模型的一个关键特征是它们构建有称为注意力层的特殊层。事实上，介绍 Transformer 架构的论文的标题是“Attention Is All You Need”！我们将在PPT的后面探讨注意力层的细节；现在，只需要知道这一层会告诉模型在处理每个单词的表示时特别注意你传递给它的句子中的某些单词（并或多或少地忽略其他单词）

同样的概念适用于与自然语言相关的任何任务：一个词本身具有意义，但该意义深受上下文的影响，上下文可以是正在研究的词之前或之后的任何其他词（或词）。

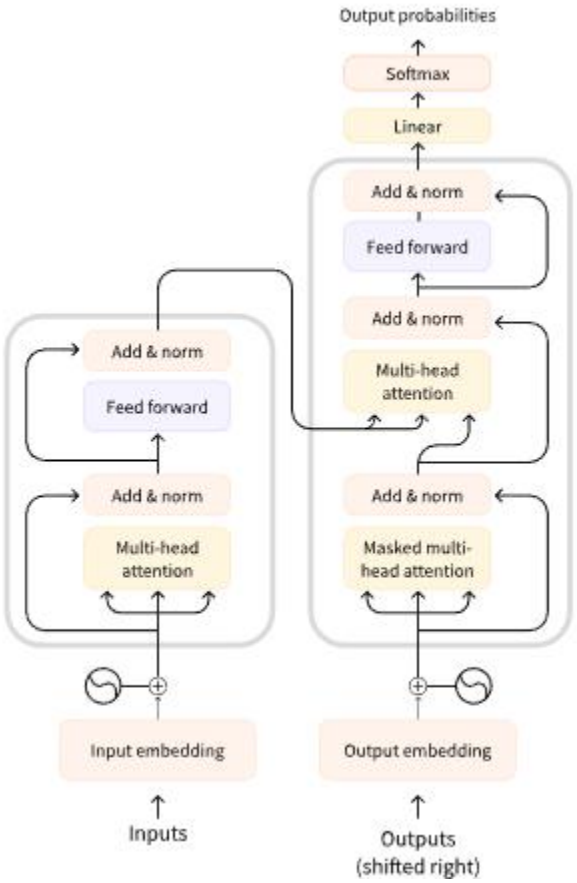


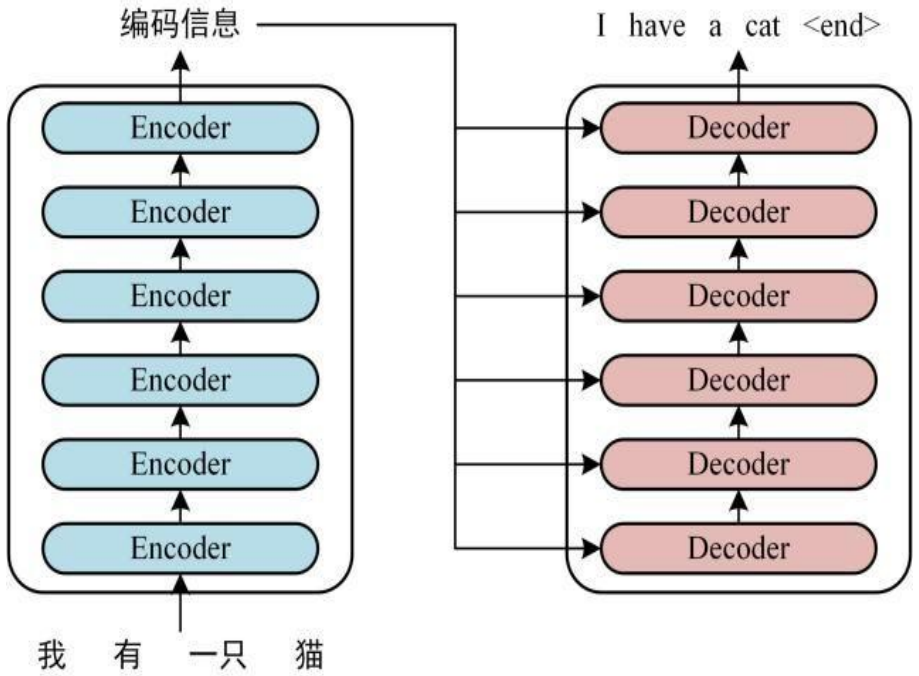
原来的 Transformer 架构是这样的，左边是编码器，右边是解码器：

请注意，解码器块中的第一个注意层关注解码器的所有（过去）输入，但第二个注意层使用编码器的输出。因此，它可以访问整个输入句子以最好地预测当前单词。这非常有用，因为不同的语言可能有语法规则，将单词按不同的顺序排列，或者在句子后面提供的某些上下文可能有助于确定给定单词的最佳翻译。

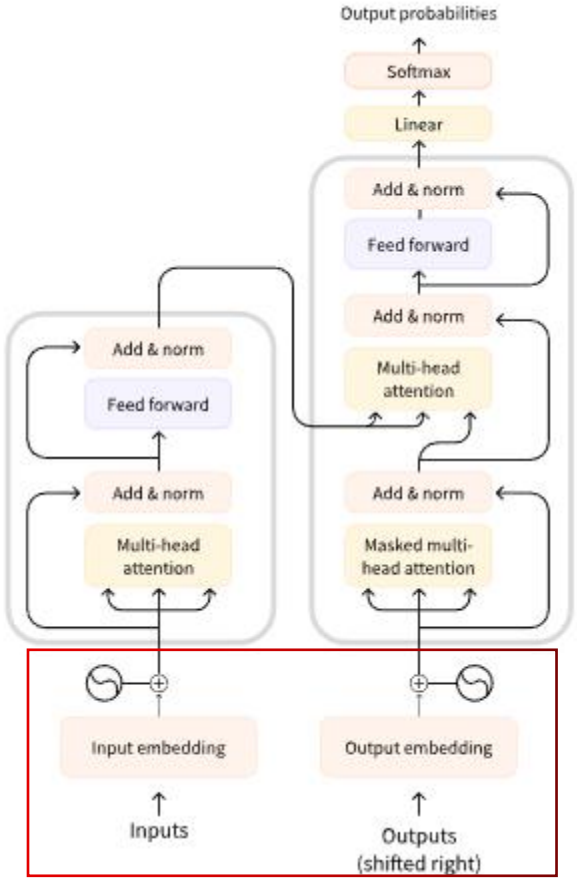
自注意力机制和多头注意力机制可参考：https://github.com/Mrgengli/deep-learning-notebook/blob/main/Attention%E6%9C%BA%E5%88%B6.md

Transformer 输入





2.Transformer 输入



Transformer 中单词的输入表示 x 由单词 Embedding 和位置 Embedding（Positional Encoding）相加得到，通常定义为 TransformerEmbedding 层，

单词 Embedding

单词的 Embedding 有很多种方式可以获取，例如可以采用 Word2Vec、Glove 等算法预训练得到，也可以在 Transformer 中训练得到。

位置 Embedding

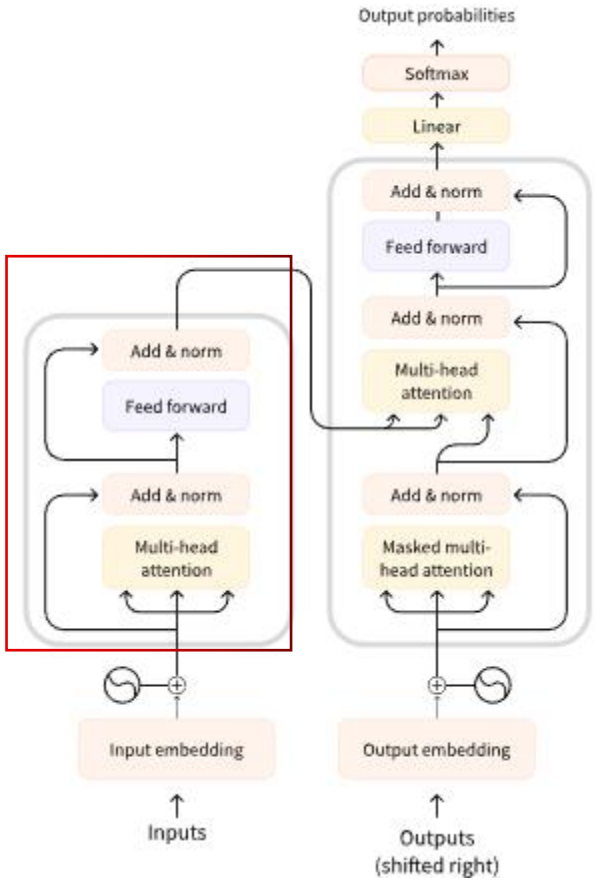
Transformer 中除了单词的 Embedding，还需要使用位置 Embedding 表示单词出现在句子中的位置。因为 Transformer 不采用 RNN 的结构，而是使用全局信息，不能利用单词的顺序信息，而这部分信息对于 NLP 来说非常重要。所以 Transformer 中使用位置 Embedding 保存单词在序列中的相对或绝对位置。

位置 Embedding 用 PE 表示，PE 的维度与单词 Embedding 是一样的。PE 可以通过训练得到，也可以使用某种公式计算得到。在 Transformer 中采用了后者，计算公式如下：

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

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

2..Transformer的Encoder



Encoder block是由6个encoder堆叠而成，Nx=6。上图2中的灰框部分就是一个encoder的内部结构，从图中我们可以看出一个encoder由Multi-Head Attention 和 全连接神经网络Feed Forward Network构成。

首先必须要了解self-attention

- 软性注意力机制：在选择信息的时候，计算N个输入的加权平均，再输入到神经网络中计算
- 硬性注意力机制：选择输入序列某一个位置上的信息，比如随机选择一个信息或者选择概率最高的信息

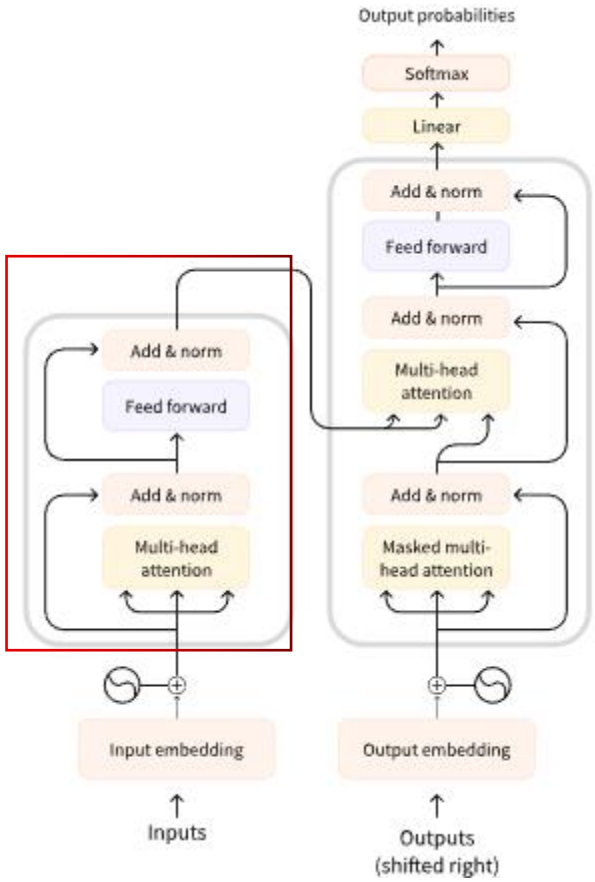
软注意力机制：

注意力值的计算可以分为两步：

（1）在所有输入信息上计算注意力分布

（2）根据注意力分布来计算输入信息的加权平均。

2..Transformer的Encoder



注意力值计算

1.计算输入信息的注意力分布

X：输入信息向量（信息存储器）

Q：查询向量，用来查询并选择X中的某些信息

Z：是在[1:N]中的值表示被选择信息的索引位置，选择了第i个输入信息

选择第i个输入信息的概率 α_i 为：

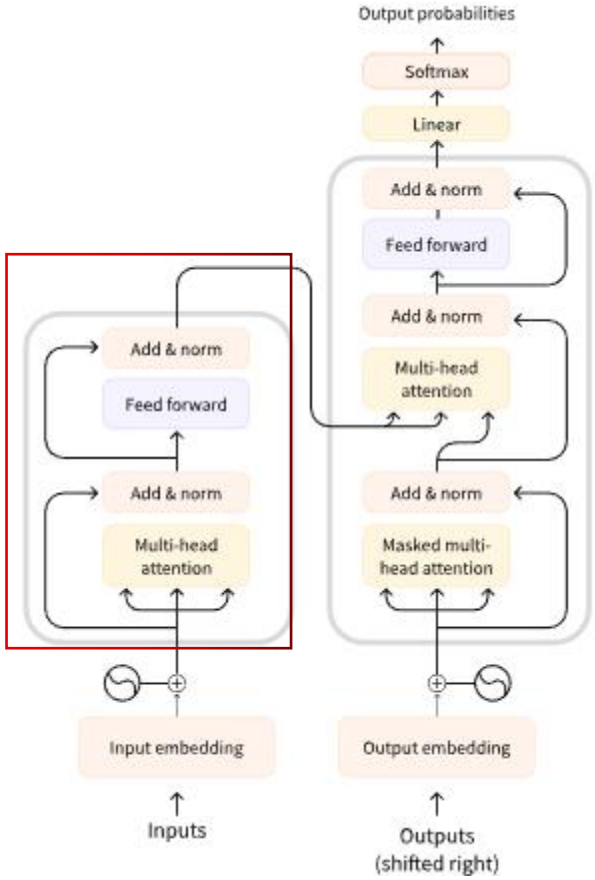
$$\begin{aligned}\alpha_i &= p(z=i|X,\mathbf{q}) \\ &= \text{softmax}\left(s(\mathbf{x}_i,\mathbf{q})\right) \\ &= \frac{\exp\left(s(\mathbf{x}_i,\mathbf{q})\right)}{\sum_{j=1}^{N}\exp\left(s(\mathbf{x}_j,\mathbf{q})\right)}\end{aligned}$$

α_i 构成的概率向量就称为注意力分布，

$s(\mathbf{x}_i,\mathbf{q})$ 是注意力打分函数（其实就是用来计算X和Q的相关性），有以下几种形式：

加性模型	$s(\mathbf{x}_i,\mathbf{q}) = \mathbf{v}^{\mathrm{T}}\tanh(W\mathbf{x}_i + U\mathbf{q})$
点积模型	$s(\mathbf{x}_i,\mathbf{q}) = \mathbf{x}_i^{\mathrm{T}}\mathbf{q},$
缩放点积模型	$s(\mathbf{x}_i,\mathbf{q}) = \frac{\mathbf{x}_i^{\mathrm{T}}\mathbf{q}}{\sqrt{d}},$
双线性模型	$s(\mathbf{x}_i,\mathbf{q}) = \mathbf{x}_i^{\mathrm{T}}W\mathbf{q},$

2..Transformer的Encoder

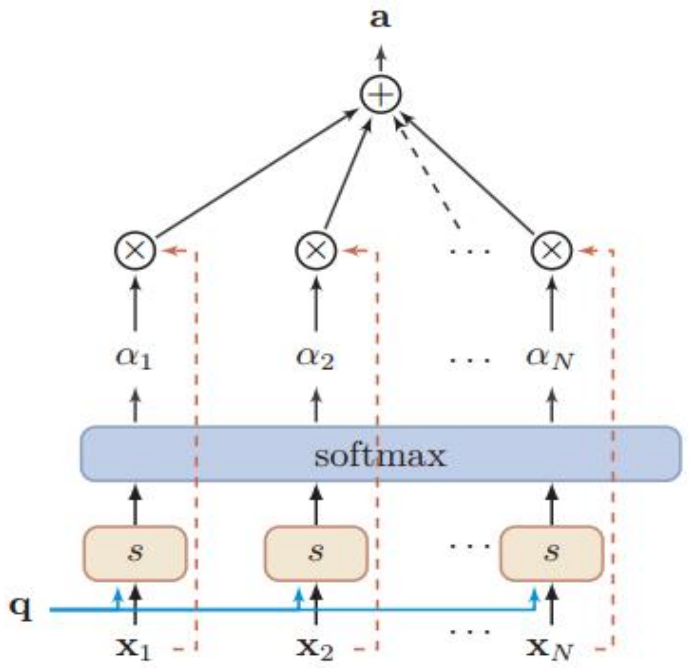


2.用注意力分布对X进行加权平均

注意力分布 α_i 表示在给定查询Q时，输入信息向量X中第i个信息与查询q的相关程度。采用“软性”信息选择机制给出查询所得的结果，就是用加权平均的方式对输入信息进行汇总，得到Attention值

$$\mathbf{att}(X,\mathbf{q}) = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i$$

下图是计算Attention值的过程图：



2..Transformer的Encoder

（二）键值对注意力模式

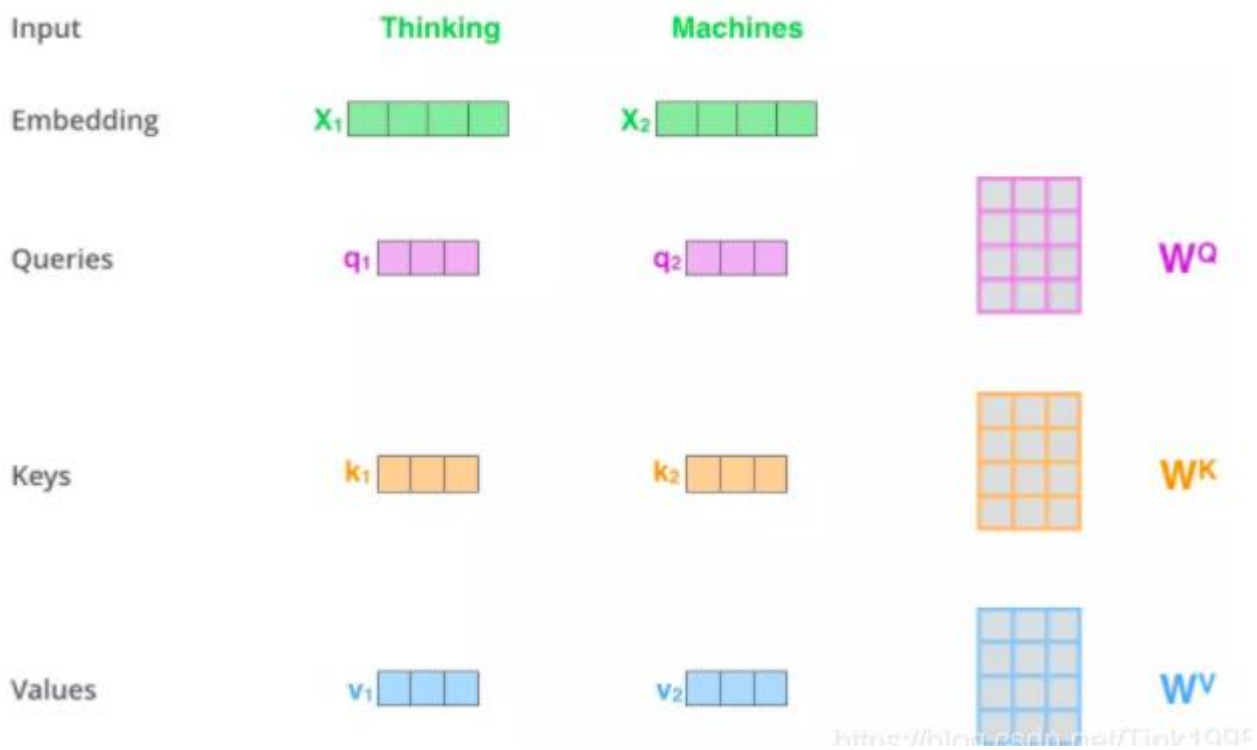
我们用键值对来表示输入信息，那么N个输入信息就表示为：$(K, V) = [(k_1, v_1), (k_2, v_2), ..., (k_N, v_N)]$

其中“键”用来计算注意分布 α_i ，“值”用来计算聚合信息。

那么就可以将注意力机制看做是一种软寻址操作：把输入信息X看做是存储器中存储的内容，元素由地址Key（键）和值Value组成，当前有个Key=Query的查询，目标是取出存储器中对应的Value值，即Attention值。而在软寻址中，并非需要硬性满足Key=Query的条件来取出存储信息，而是通过计算Query与存储器内元素的地址Key的相似度来决定，从对应的元素Value中取出多少内容。每个地址Key对应的Value值都会被抽取内容出来，然后求和，这就相当于由Query与Key的相似性来计算每个Value值的权重，然后对Value值进行加权求和。加权求和得到最终的Value值，也就是Attention值。

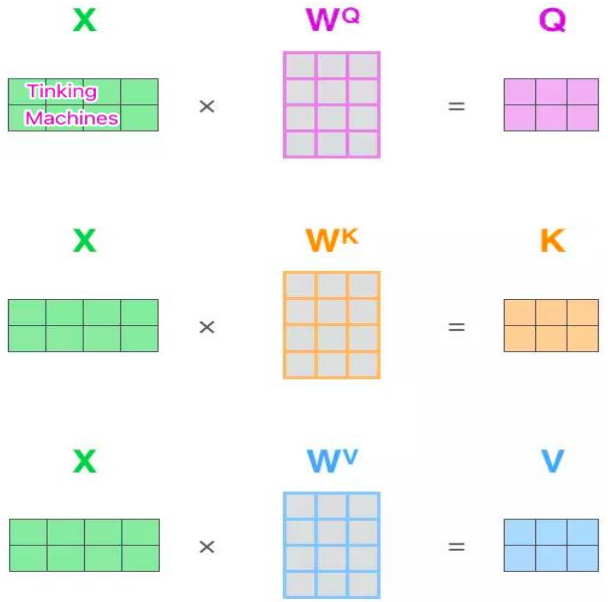
2..Transformer的Multi-Head Attention 结构

self-attention，假如输入序列是"Thinking Machines"，x1，x2就是对应地"Thinking"和"Machines"添加过位置编码之后的词向量，然后词向量通过三个权值矩阵 W^Q, W^K, W^V ，转变成为计算Attention值所需的Query，Keys，Values向量。



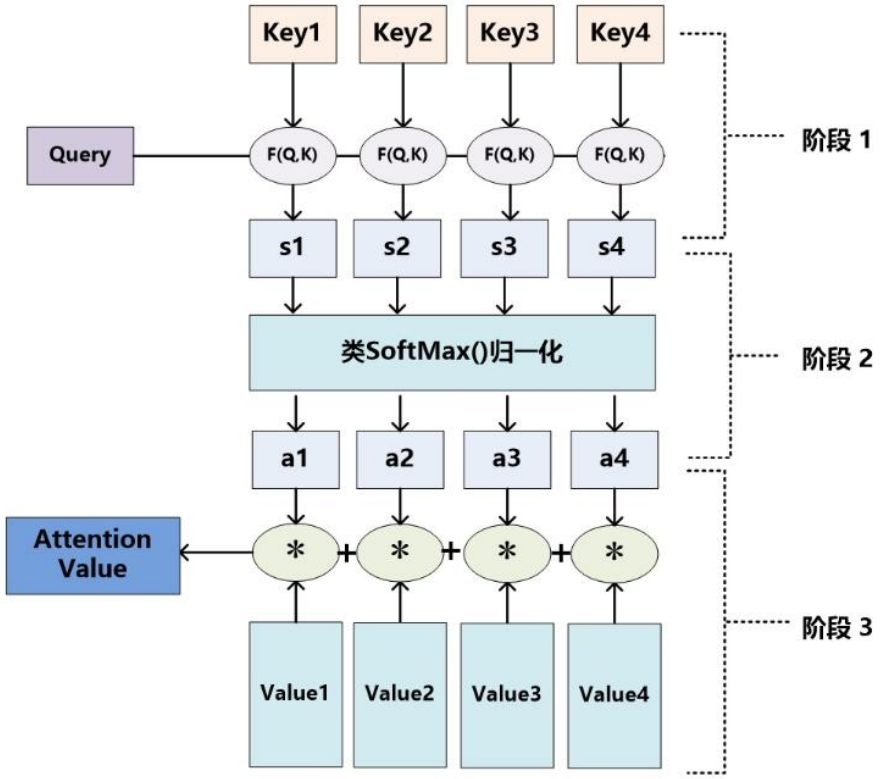
2..Transformer的Multi-Head Attention 结构

因为咱们再实际使用中，每一个样本，也就是每一条序列数据都是以矩阵的形式输入地，故可以看到上图中，X矩阵是由"Tinking"和"Machines"词向量组成的矩阵，然后跟W^Q, W^K, W^V过变换得到Q，K，V。假设词向量是512维，X矩阵的维度是(2,512)，均是(512,64)维，得到的Query，Keys，Values就都是(2,64)维。



得到Q，K，V之后，接下来就是计算Attention值了。获得的记为z。

2..Transformer的Encoder



具体的计算可以分为三个步骤：

1.根据Query和Key计算二者的相似度。可以用上面所列出的加性模型、点积模型或余弦相似度来计算，得到注意力得分si

$$s_i = F(Q, k_i)$$

2.用softmax函数对注意力得分进行数值转换。一方面可以进行归一化，得到所有权重系数之和为1的概率分布，另一方面可以用softmax函数的特性突出重要元素的权重；

$$\alpha_i = \text{softmax}(s_i) = \frac{\exp(s_i)}{\sum_{j=1}^{N} \exp(s_j)}$$

3.根据权重系数对Value进行加权求和：

$$Attention((K,V),Q) = \sum_{i=1}^{N} \alpha_i v_i$$

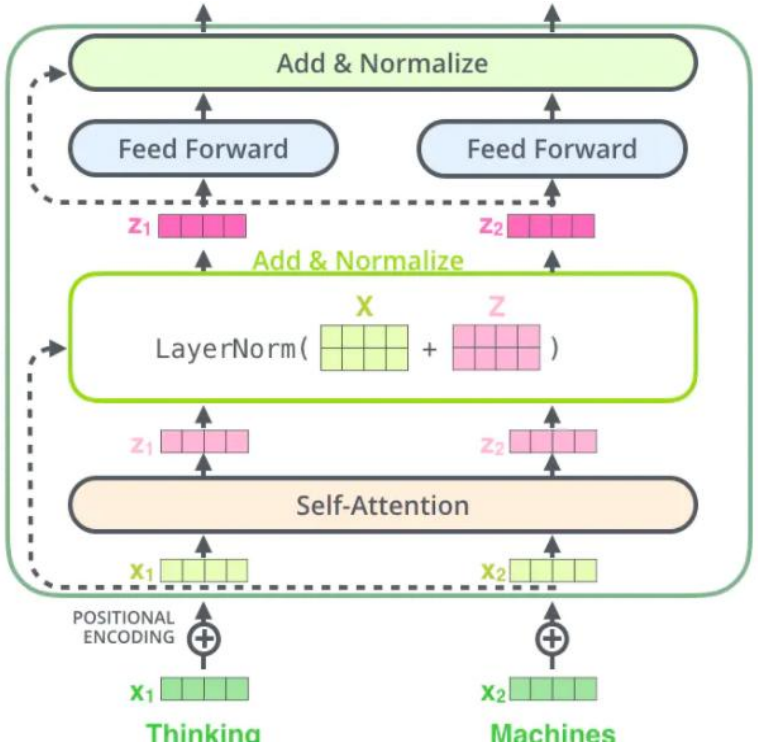
则上面的公式就可以整理为：

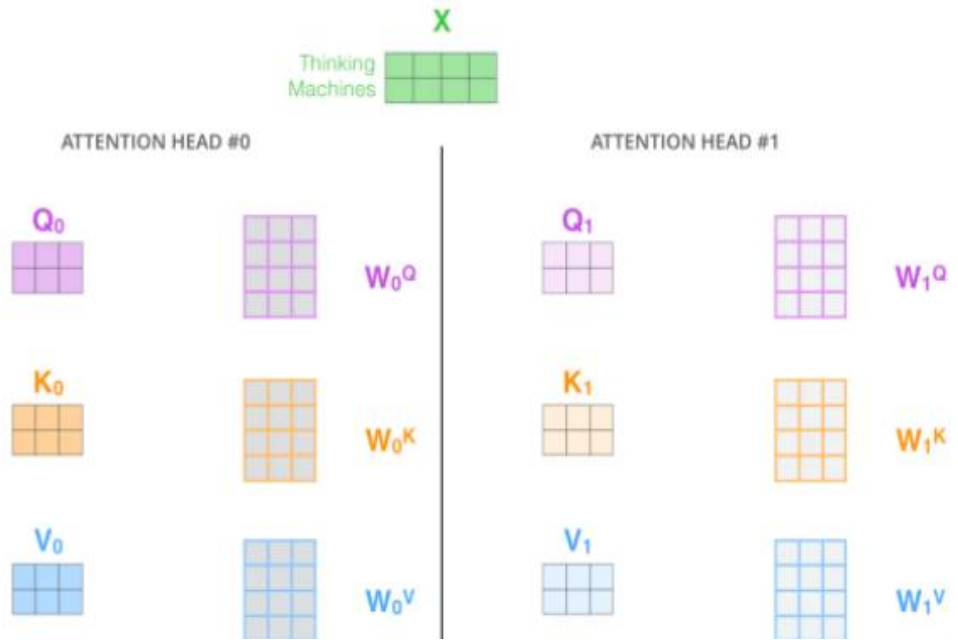
$$\begin{aligned} \mathbf{att}\Big((K,V),\mathbf{q}\Big) &= \sum_{i=1}^{N} \alpha_i \mathbf{v}_i, \\ &= \sum_{i=1}^{N} \frac{\exp\Big(s(\mathbf{k}_i,\mathbf{q})\Big)}{\sum_j \exp\Big(s(\mathbf{k}_j,\mathbf{q})\Big)} \mathbf{v}_i \end{aligned}$$

2..Transformer的Encoder

说了这么多好像都只是在说self-attention，那么Multi-Head Attention呢？

Multi-Head Attention 很简单，就是在self-attention的基础上，对于输入的embedding矩阵，self-attention只使用了一组 W^Q, W^K, W^V 来进行变换得到Query，Keys，Values。而Multi-Head Attention使用多组 W^Q, W^K, W^V 得到多组Query，Keys，Values，然后每组分别计算得到一个Z矩阵，最后将得到的多个Z矩阵进行拼接。Transformer里面是使用了8组不同的 W^Q, W^K, W^V



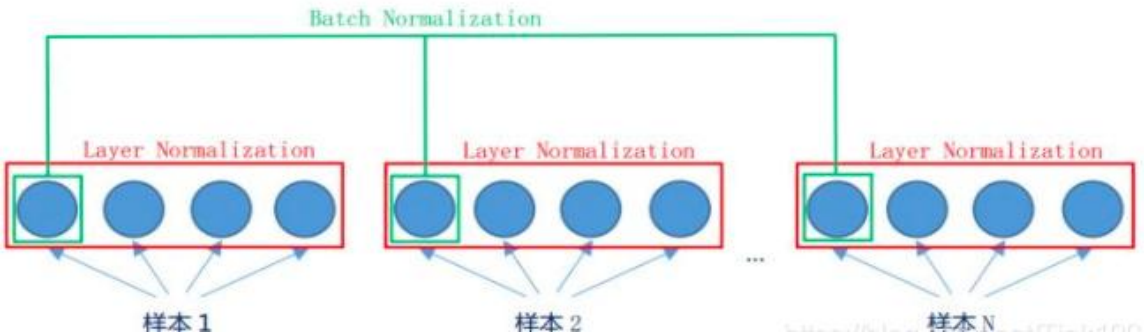


从上图中可以看到，在经过Multi-Head Attention得到矩阵Z之后，并没有直接传入全连接神经网络FNN，而是经过了一步：Add＆Normalize。

2..Transformer的Add＆Normalize

Add

Add，就是在Z的基础上加了一个残差块X，加入残差块X的目的是为了防止在深度神经网络训练中发生退化问题，退化的意思就是深度神经网络通过增加网络的层数，Loss逐渐减小，然后趋于稳定达到饱和，然后再继续增加网络层数，Loss反而增大。



Normalize

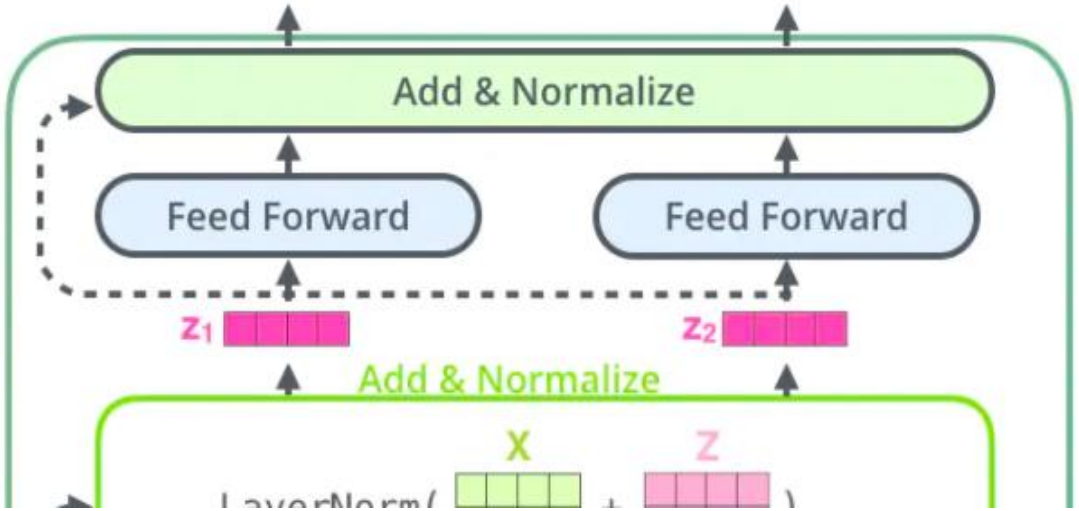
为什么要进行Normalize呢？

在神经网络进行训练之前，都需要对于输入数据进行Normalize归一化，目的有二：1，能够加快训练的速度。2.提高训练的稳定性。

为什么使用Layer Normalization（LN）而不使用Batch Normalization（BN）呢？

先看图，LN是在同一个样本中不同神经元之间进行归一化，而BN是在同一个batch中不同样本之间的同一位置的神经元之间进行归一化。
BN是对于相同的维度进行归一化，但是咱们NLP中输入的都是词向量，一个300维的词向量，单独去分析它的每一维是没有意义地，在每一维上进行归一化也是适合地，因此这里选用的是LN。

2..Transformer的Feed-Forward Networks



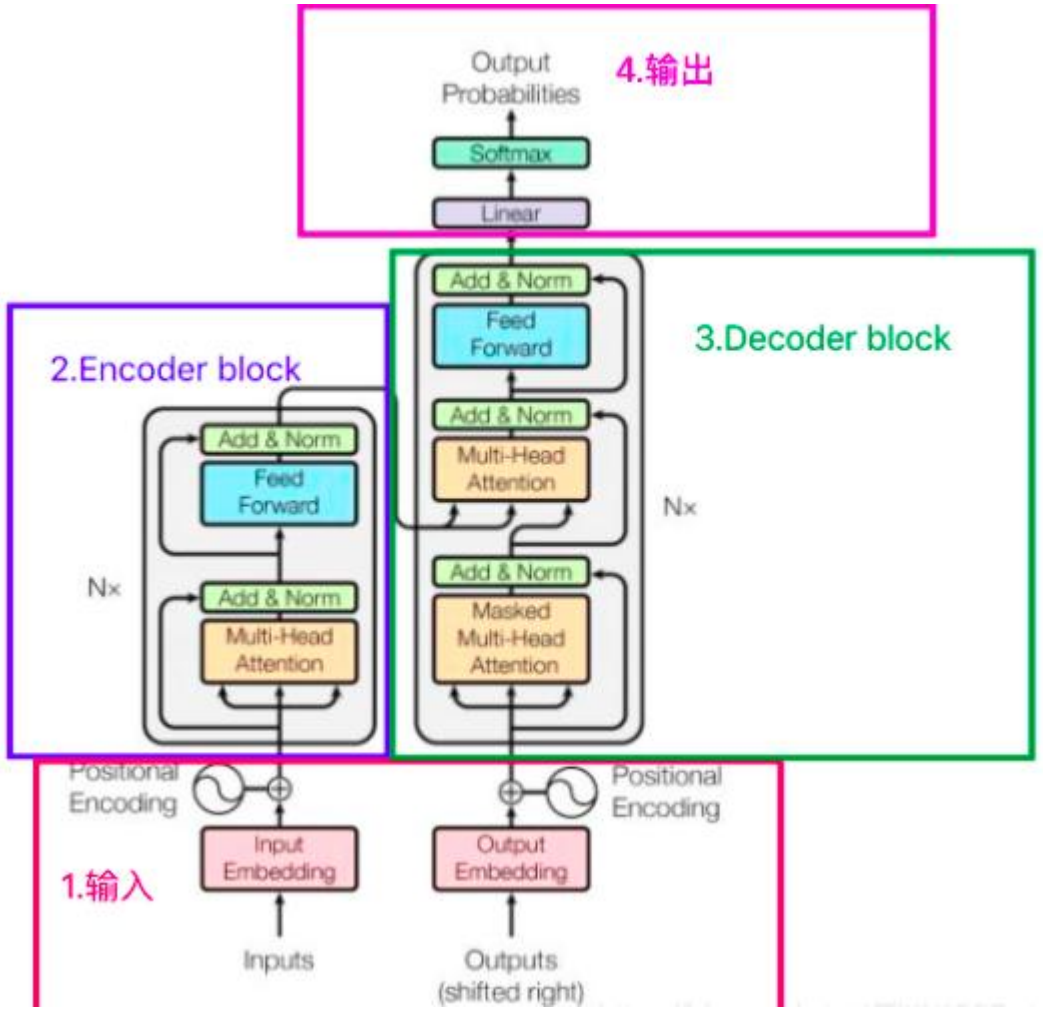
全连接层公式如下：

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

这里的全连接层是一个两层的神经网络，先线性变换，然后ReLU非线性，再线性变换。
这里的x就是我们Multi-Head Attention的输出Z，还是引用上面的例子，那么Z是(2,64)维的矩阵，假设W1是(64,1024)，其中W2与W1维度相反(1024,64)，那么按照上面的公式：FFN(Z)=(2,64)x(64,1024)x(1024,64)=(2,64)，我们发现维度没有发生变化，这两层网络就是为了将输入的Z映射到更加高维的空间中(2,64)x(64,1024)=(2,1024)，然后通过非线性函数ReLU进行筛选，筛选完后再变回原来的维度。

然后经过Add＆Normalize，输入下一个encoder中，经过6个encoder后输入到decoder中，至此Transformer的Encoder部分就全部介绍完了，

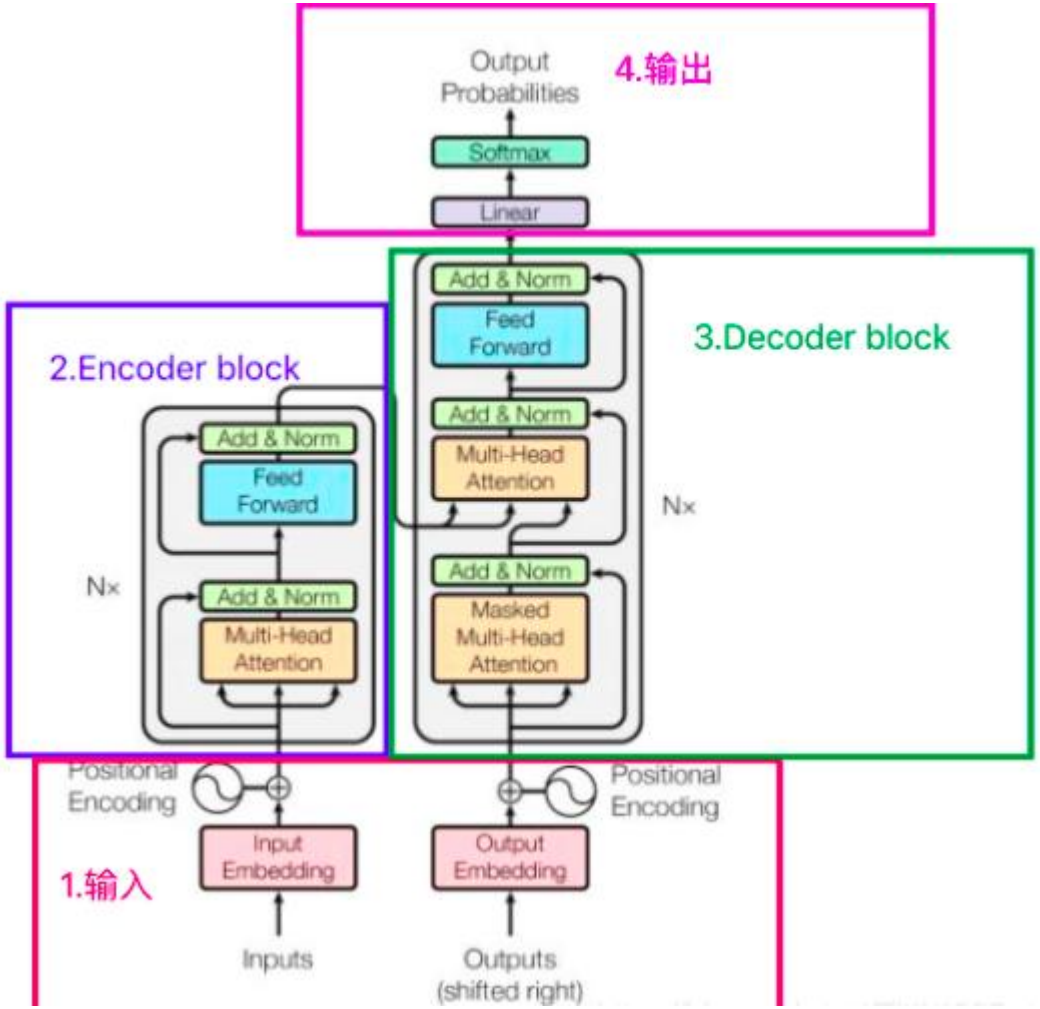
2..Transformer的Decoder



从图中我们可以看出一个decoder由Masked Multi-Head Attention，Multi-Head Attention 和 全连接神经网络FNN构成。比Encoder多了一个Masked Multi-Head Attention，其他的结构与encoder相同，那么咱们就先来看看这个**Masked Multi-Head Attention。**

与Encoder的Multi-Head Attention计算原理一样，只是多加了一个mask码。mask 表示掩码，它对某些值进行掩盖，使其在参数更新时不产生效果。Transformer 模型里面涉及两种 mask，分别是 padding mask 和 sequence mask。为什么需要添加这两种mask码呢?

2..Transformer的Decoder



在Encoder中的Multi-Head Attention也是需要进行mask地，只不过Encoder中只需要padding mask即可，而Decoder中需要padding mask和sequence mask。

1.padding mask

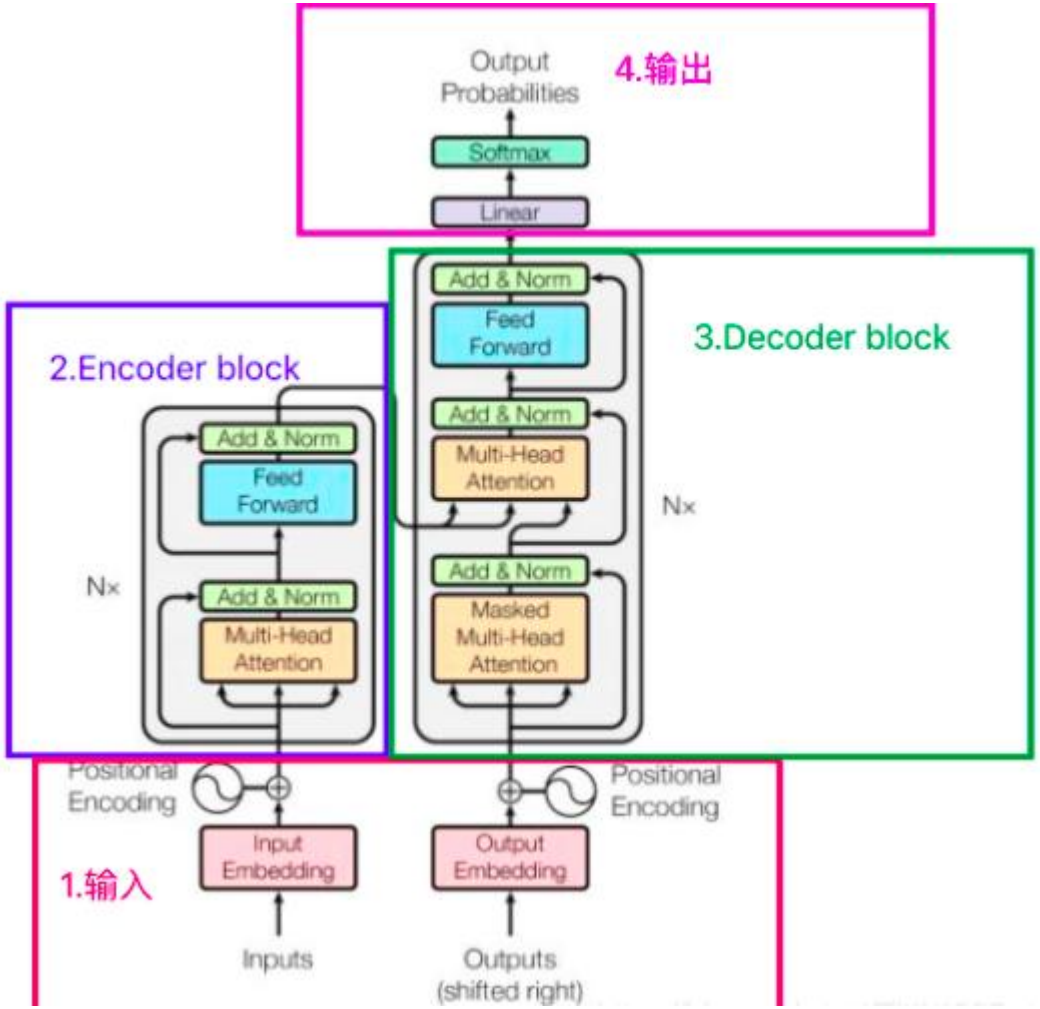
什么是 padding mask 呢？因为每个批次输入序列长度是不一样的也就是说，我们要对输入序列进行对齐。具体来说，就是给在较短的序列后面填充 0。但是如果输入的序列太长，则是截取左边的内容，把多余的直接舍弃。因为这些填充的位置，其实是没什么意义的，所以我们的attention机制不应该把注意力放在这些位置上，所以我们需要进行一些处理。具体的做法是，把这些位置的值加上一个非常大的负数(负无穷)，这样的话，经过 softmax，这些位置的概率就会接近0!

2.sequence mask

sequence mask 是为了使得 decoder 不能看见未来的信息。对于一个序列，在 time_step 为 t 的时刻，我们的解码输出应该只能依赖于 t 时刻之前的输出，而不能依赖 t 之后的输出。因此我们需要想一个办法，把 t 之后的信息给隐藏起来。这在训练的时候有效，因为训练的时候每次我们是将target数据完整输入进decoder中地，预测时不需要，预测的时候我们只能得到前一时刻预测出的输出。

那么具体怎么做呢？也很简单：产生一个上三角矩阵，上三角的值全为0。把这个矩阵作用在每一个序列上，就可以达到我们的目的。

2..Transformer的Decoder



在Encoder中的Multi-Head Attention也是需要进行mask地，只不过Encoder中只需要padding mask即可，而Decoder中需要padding mask和sequence mask。

1.padding mask

什么是 padding mask 呢？因为每个批次输入序列长度是不一样的也就是说，我们要对输入序列进行对齐。具体来说，就是给在较短的序列后面填充 0。但是如果输入的序列太长，则是截取左边的内容，把多余的直接舍弃。因为这些填充的位置，其实是没什么意义的，所以我们的attention机制不应该把注意力放在这些位置上，所以我们需要进行一些处理。具体的做法是，把这些位置的值加上一个非常大的负数(负无穷)，这样的话，经过 softmax，这些位置的概率就会接近0!

2.sequence mask

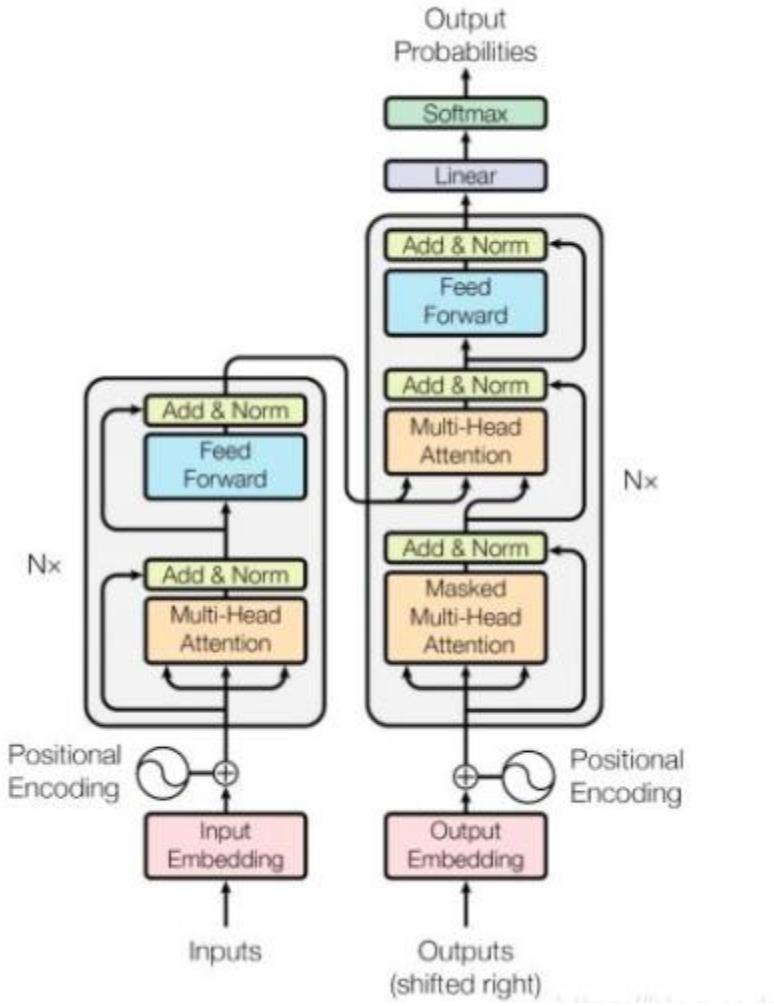
sequence mask 是为了使得 decoder 不能看见未来的信息。对于一个序列，在 time_step 为 t 的时刻，我们的解码输出应该只能依赖于 t 时刻之前的输出，而不能依赖 t 之后的输出。因此我们需要想一个办法，把 t 之后的信息给隐藏起来。这在训练的时候有效，因为训练的时候每次我们是将target数据完整输入进decoder中地，预测时不需要，预测的时候我们只能得到前一时刻预测出的输出。

那么具体怎么做呢？也很简单：产生一个上三角矩阵，上三角的值全为0。把这个矩阵作用在每一个序列上，就可以达到我们的目的。

2..Transformer的输出

Output如图中所示，首先经过一次线性变换，然后Softmax得到输出的概率分布，然后通过词典，输出概率最大的对应的单词作为我们的预测输出。

2..Transformer的优缺点



Transformer虽然好，但它也不是万能地，还是存在着一些不足之处，接下来就来介绍一下它的优缺点：

优点：

1.效果好

2.可以并行训练，速度快

3.很好地解决了长距离依赖的问题

缺点：

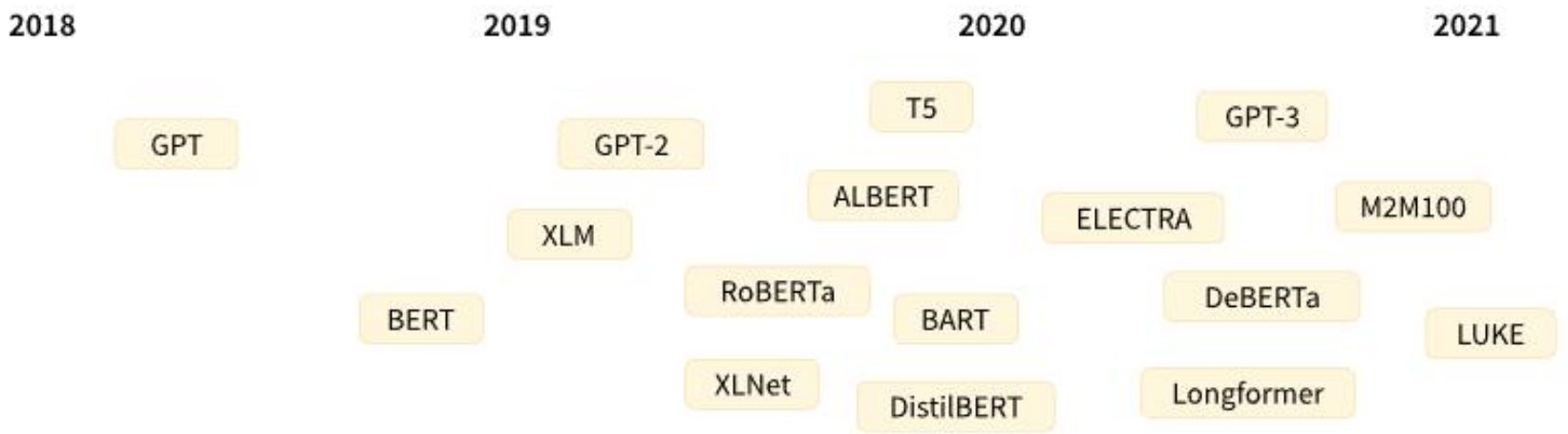
1.完全基于self-attention，对于词语位置之间的信息有一定的丢失，虽然加入了positional encoding来解决这个问题，但也还存在着可以优化的地方。

国家材料服役安全科学中心
National Center for Materials Service Safety

1 基于Transformer的大模型

Transformer的历史

以下是 Transformer 模型（简短）历史中的一些参考点：



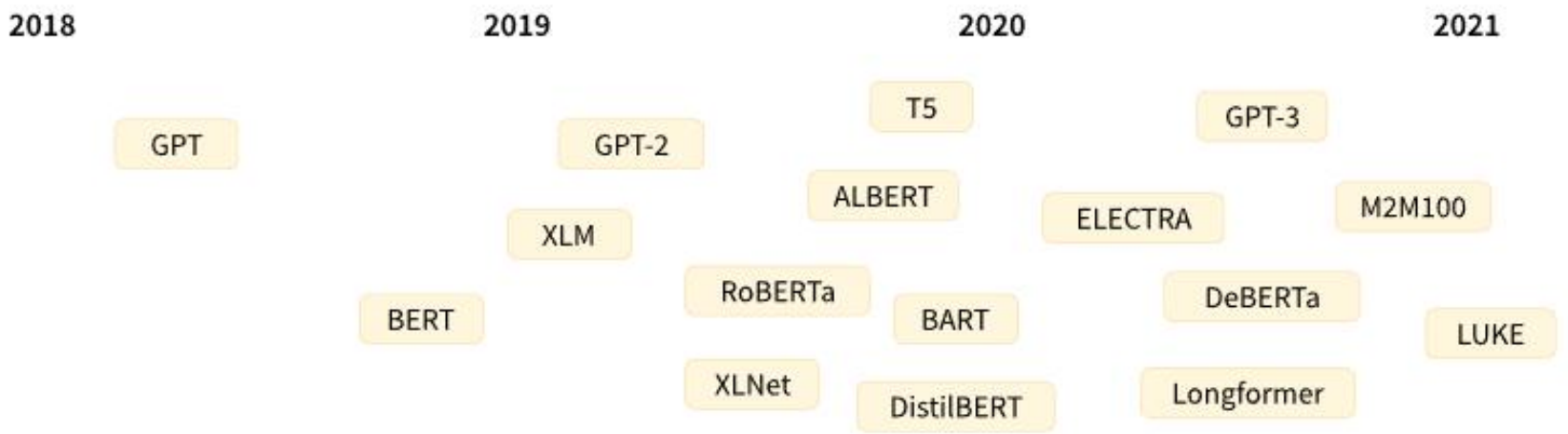
Transformer 架构于2017 年 6 月推出。最初的研究重点是翻译任务。随后引入了几个有影响力的模型，包括：

2018年6月：GPT，第一个预训练的Transformer模型，用于各种NLP任务的fine-tuning，获得state-of-the-art结果

2018 年 10 月：BERT，另一个大型预训练模型，这个模型旨在生成更好的句子摘要（后面将详细介绍！）

Transformer的历史

以下是 Transformer 模型（简短）历史中的一些参考点：



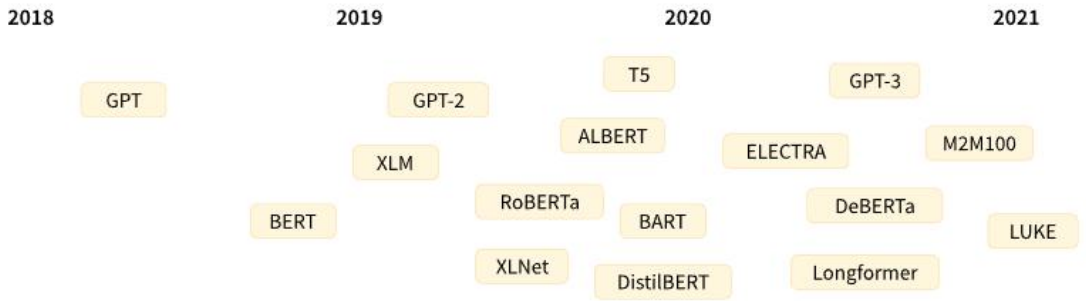
2019 年 2 月：GPT-2，GPT 的改进（和更大）版本，由于道德问题没有立即公开发布

2019 年 10 月：DistilBERT，BERT 的提炼版本，速度提高 60%，内存减少 40%，并且仍保留 BERT 97% 的性能

2019 年 10 月：BART和T5，两个使用与原始 Transformer 模型相同架构的大型预训练模型（第一个）

2020 年 5 月，GPT-3，GPT-2 的更大版本，能够在各种任务上表现良好而无需微调（称为零样本学习）

Transformer的历史



左边的列表远非全面，只是为了突出一些不同类型的Transformer 模型。大体上，它们可以分为三类：

GPT-like（也称为自回归Transformer 模型）

BERT-like（也称为自动编码Transformer 模型）

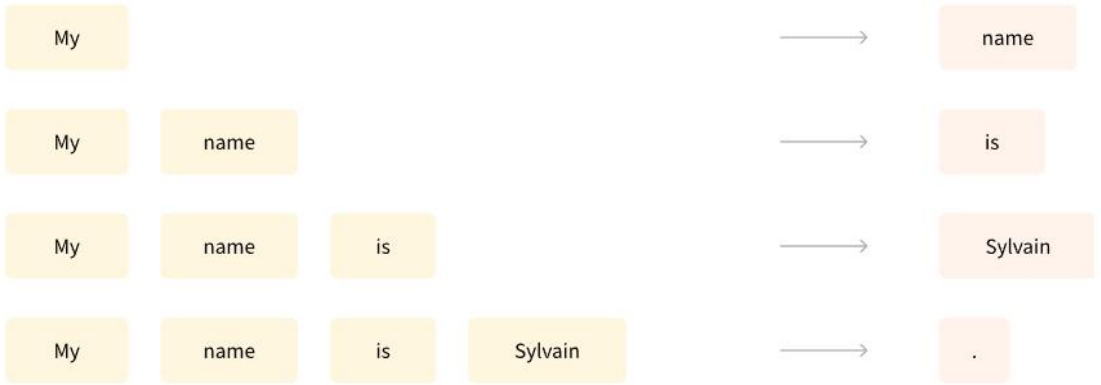
BART/T5-like（也称为序列到序列Transformer 模型）

Transformer 是大模型，除了一些特例（如 DistilBERT）外，实现更好性能的一般策略是增加模型的大小以及预训练的数据量。其中，GPT-2 是使用「transformer 解码器模块」构建的，而 BERT 则是通过「transformer 编码器」模块构建的。这是两种不同的思路，OpenAI选用的是前者GPT解码模型，而谷歌则压在了后者BERT编码模型上面，从现在的ChatGPT大红大紫上面可以看出，谷歌虽然是大语言模型的提出者，但是却因为当初选择错了道路，所以之前的优势复制东流，

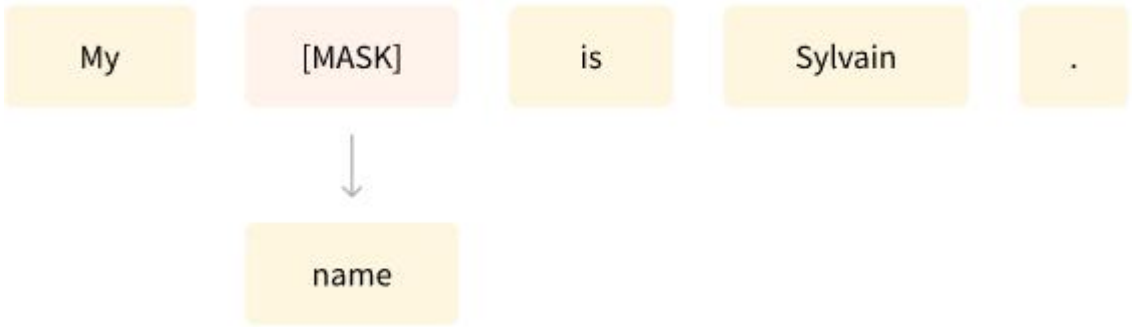
Transformer是语言模型

上面提到的所有Transformer模型（GPT、BERT、BART、T5等）都被训练成了语言模型。这意味着他们已经以自我监督的方式接受了大量原始文本的训练。自监督学习是一种训练，其中目标是根据模型的输入自动计算的。这意味着不需要人类来标记数据!

这种类型的模型对其所训练的语言形成了统计理解，但对于特定的实际任务并不是很有用。正因为如此，一般的预训练模型会经历一个迁移学习的过程。在此过程中，模型在给定任务上以监督方式进行微调——即使用人工标注的标签。



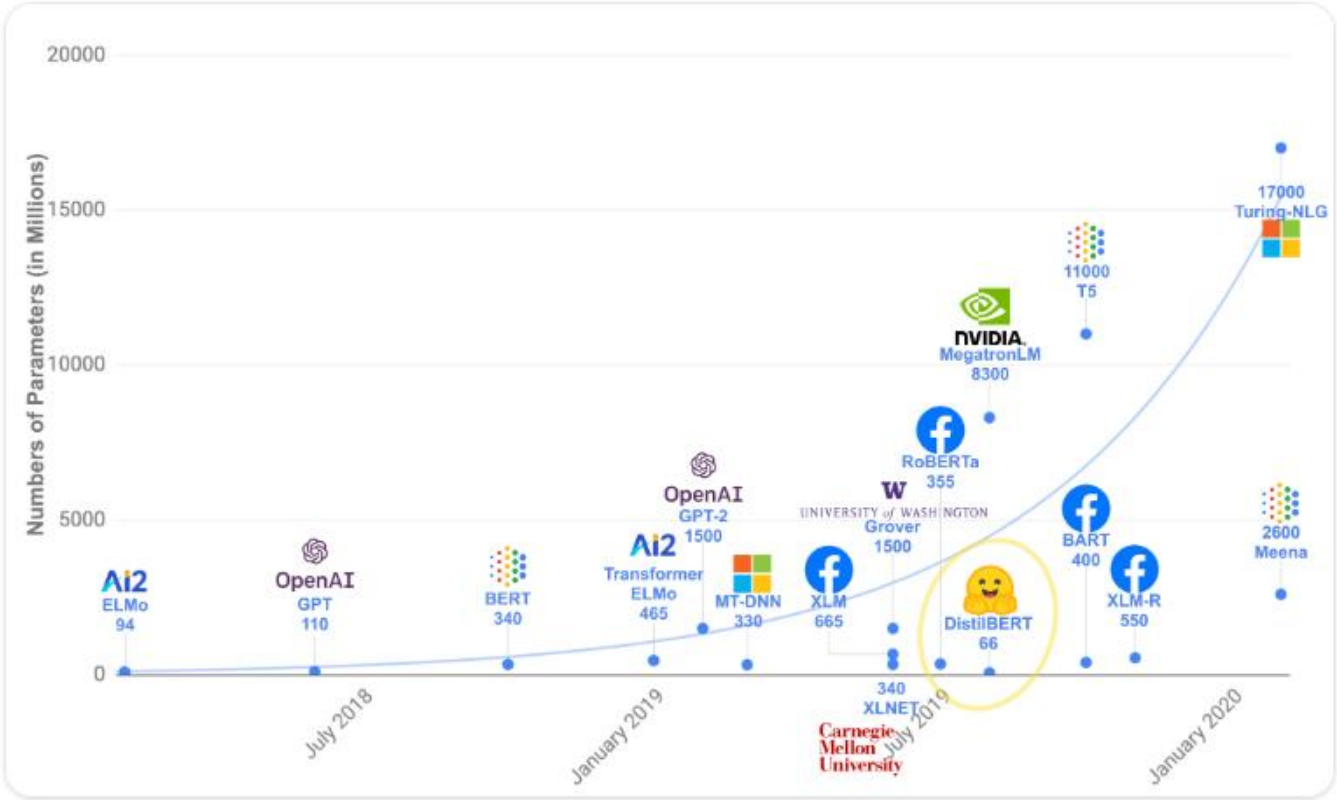
一个任务的例子是预测一个句子中的下一个单词已经阅读了前面的n 个单词。这称为因果语言建模，因为输出取决于过去和现在的输入，而不是未来的输入。



另一个例子是掩码语言建模，其中模型预测句子中的掩码词。

Transformer是大模型

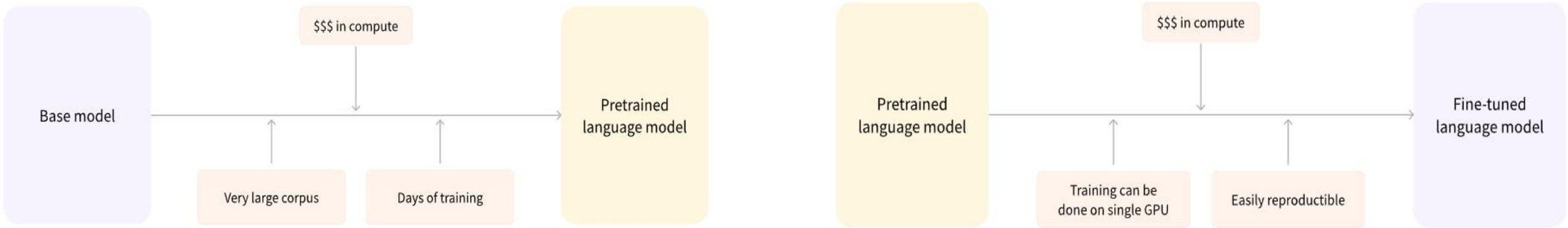
除了一些异常值（如 DistilBERT），实现更好性能的一般策略是增加模型的大小以及它们预训练的数据量。



不幸的是，训练模型，尤其是大型模型，需要大量数据。这在时间和计算资源方面变得非常昂贵。

迁移学习

预训练是从头开始训练模型的行为：随机初始化权重，并且在没有任何先验知识的情况下开始训练。



这种预训练通常是在非常大量的数据上进行的。因此，它需要非常大的语料库，训练可能需要长达数周的时间。另一方面，微调是在对模型进行预训练之后进行的训练。要执行微调，您首先需要一个预训练的语言模型，然后使用特定于您的任务的数据集执行额外的训练。那为什么不直接为最终任务进行训练呢？有几个原因：

- 预训练模型已经在与微调数据集有一些相似之处的数据集上进行了训练。因此，微调过程能够利用初始模型在预训练期间获得的知识（例如，对于 NLP 问题，预训练模型将对您用于任务的语言有某种统计理解）。
- 由于预训练模型已经在大量数据上进行了训练，因此微调需要更少的数据才能获得不错的结果。
- 出于同样的原因，获得好结果所需的时间和资源要少得多。

例如，可以利用在英语语言上训练的预训练模型，然后在 arXiv 语料库上对其进行微调，从而产生一个基于科学/研究的模型。微调只需要有限数量的数据：预训练模型获得的知识是“迁移”的，因此称为迁移学习。

Encoder models

编码器模型仅使用 Transformer 模型的编码器。在每个阶段，注意力层都可以访问初始句子中的所有单词。这些模型通常具有“双向”注意力的特征，通常称为自动编码模型。

这些模型的预训练通常围绕着以某种方式破坏给定的句子（例如，通过掩盖其中的随机词）并让模型寻找或重建初始句子。

编码器模型最适合需要理解完整句子的任务，例如**句子分类、命名实体识别（以及更普遍的单词分类）和提取式问答。**

该系列模型的代表包括：

- ALBERT
- BERT
- DistilBERT
- ELECTRA
- RoBERTa

Decoder models

解码器模型仅使用 Transformer 模型的解码器。在每个阶段，对于给定的单词，注意力层只能访问句子中位于它之前的单词。这些模型通常称为自回归模型。

解码器模型的预训练通常围绕预测句子中的下一个单词展开。

这些模型最适合涉及**文本生成**的任务。

该系列模型的代表包括:

- CTRL
- GPT
- GPT-2
- Transformer XL

Encoder-decoder models

编码器-解码器模型（也称为序列到序列模型）使用 Transformer 架构的两个部分。在每个阶段，编码器的注意力层可以访问初始句子中的所有单词，而解码器的注意力层只能访问输入中给定单词之前的单词。

这些模型的预训练可以使用编码器或解码器模型的目标来完成，但通常涉及一些更复杂的东西。例如，T5通过用单个掩码特殊词替换随机文本范围（可以包含多个单词）进行预训练，然后目标是预测该掩码词替换的文本。

序列到序列模型最适合围绕根据给定输入生成新句子的任务，例如**摘要、翻译或生成式问答**。

该系列模型的代表包括：

- BART
- mBART
- Marian
- T5

国家材料服役安全科学中心
National Center for Materials Service Safety

敬请批评指正！

2023年6月7日