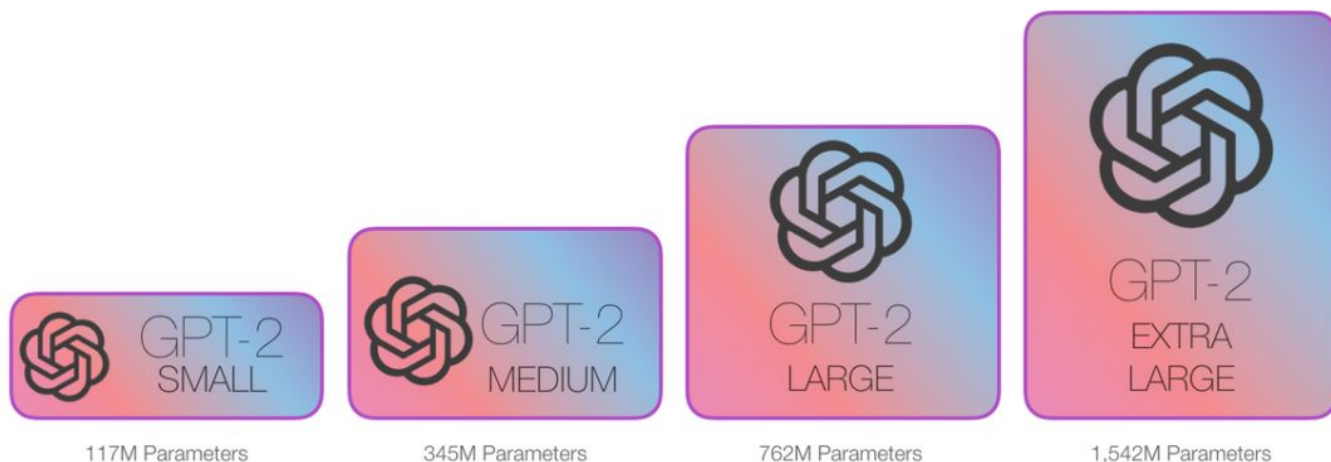


# 完全图解GPT-2：看完这篇就够了（一）

Jay Alammar 机器学习算法与Python学习 1月4日

点击 [机器学习算法与Python学习](#)，选择加星标

精彩内容不迷路



选自github.io，作者：Jay Alammar

机器之心编译

今年涌现出了许多机器学习的精彩应用，令人目不暇接，OpenAI 的 GPT-2 就是其中之一。它在文本生成上有着惊艳的表现，其生成的文本在上下文连贯性和情感表达上都超过了人们对目前阶段语言模型的预期。仅从模型架构而言，GPT-2 并没有特别新颖的架构，它和只带有解码器的 transformer 模型很像。

然而，GPT-2 有着超大的规模，它是一个在海量数据集上训练的基于 transformer 的巨大模型。GPT-2 成功的背后究竟隐藏着什么秘密？本文将带你一起探索取得优异性能的 GPT-2 模型架构，重点阐释其中关键的自注意力（self-attention）层，并且看一看 GPT-2 采用的只有解码器的 transformer 架构在语言建模之外的应用。

作者之前写过一篇相关的介绍性文章「The Illustrated Transformer」，本文将在其基础上加入更多关于 transformer 模型内部工作原理的可视化解释，以及这段时间以来关于 transformer 模型的新进展。基于 transformer 的模型在持续演进，我们希望本文使用的这一套可视化表达方法可以使此类模型更容易解释。

## Contents

- **Part 1: GPT2 And Language Modeling**
  - What is a Language Model
  - Transformers for Language Modeling
  - One Difference From BERT
  - The Evolution of The Transformer Block
  - Crash Course in Brain Surgery: Looking Inside GPT-2
  - A Deeper Look Inside
  - End of part #1: The GPT-2, Ladies and Gentlemen
- **Part 2: The Illustrated Self-Attention**
  - Self-Attention (without masking)
  - 1- Create Query, Key, and Value Vectors
  - 2- Score
  - 3- Sum
  - The Illustrated Masked Self-Attention
  - GPT-2 Masked Self-Attention
  - Beyond Language modeling
  - You've Made it!
- **Part 3: Beyond Language Modeling**
  - Machine Translation
  - Summarization
  - Transfer Learning
  - Music Generation

### 第一部分：GPT-2 和语言建模

首先，究竟什么是语言模型 (language model) ？

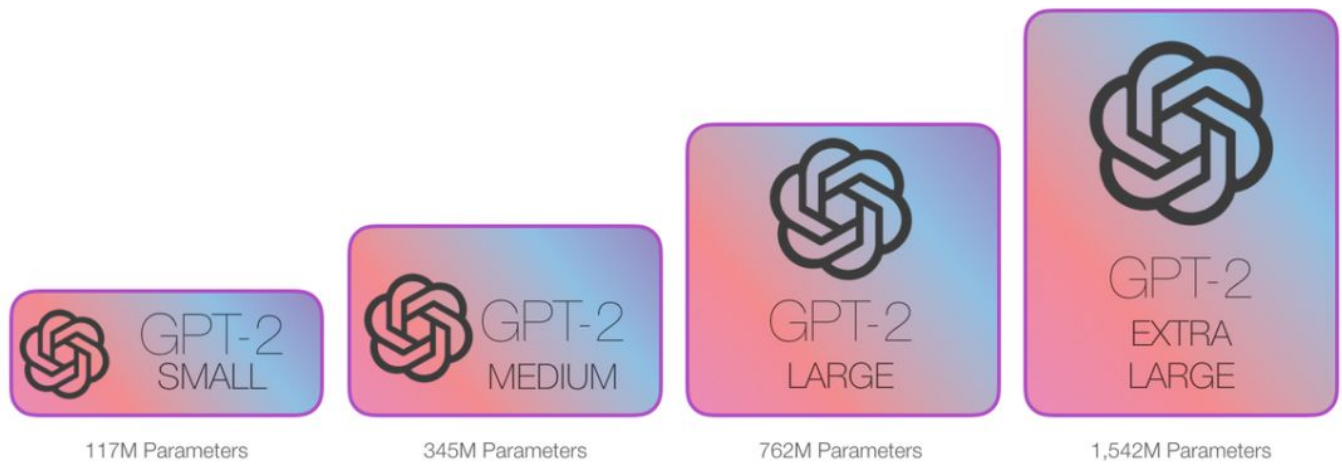
#### 何为语言模型

简单说来，语言模型的作用就是根据已有句子的一部分，来预测下一个单词会是什么。最著名的语言模型你一定见过，就是我们手机上的输入法，它可以根据当前输入的内容智能推荐下一个词。



从这个意义上说，我们可以说 GPT-2 基本上相当于输入法的单词联想功能，但它比你手机上安装的此类应用大得多，也更加复杂。OpenAI 的研究人员使用了一个从网络上爬取的 40GB 超大数据集「WebText」训练 GPT-2，该数据集也是他们的工作成果的一部分。

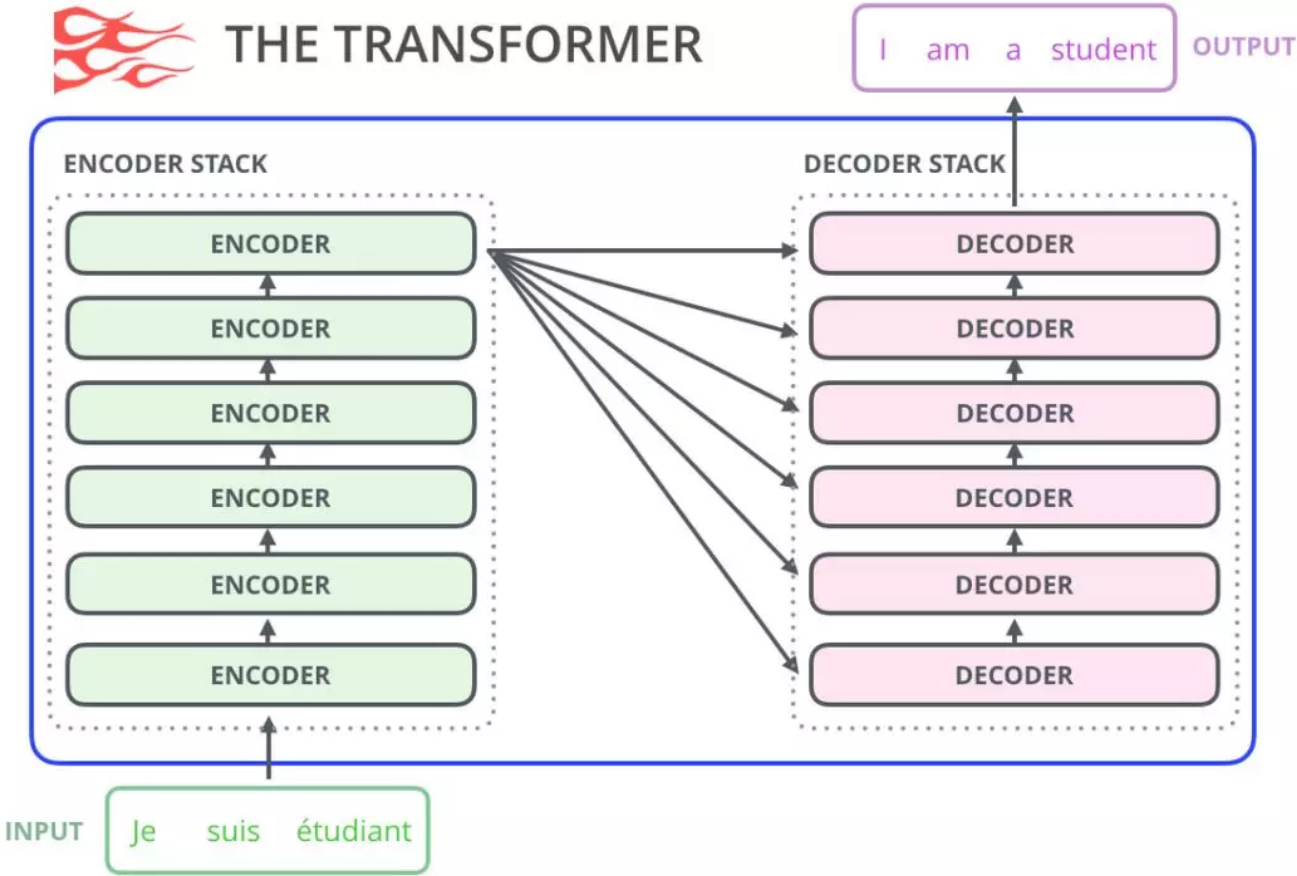
如果从占用存储大小的角度进行比较，我现在用的手机输入法「SwiftKey」也就占用了 50MB 的空间，而 GPT-2 的最小版本也需要至少 500MB 的空间来存储它的全部参数，最大版本的 GPT-2 甚至需要超过 6.5GB 的存储空间。



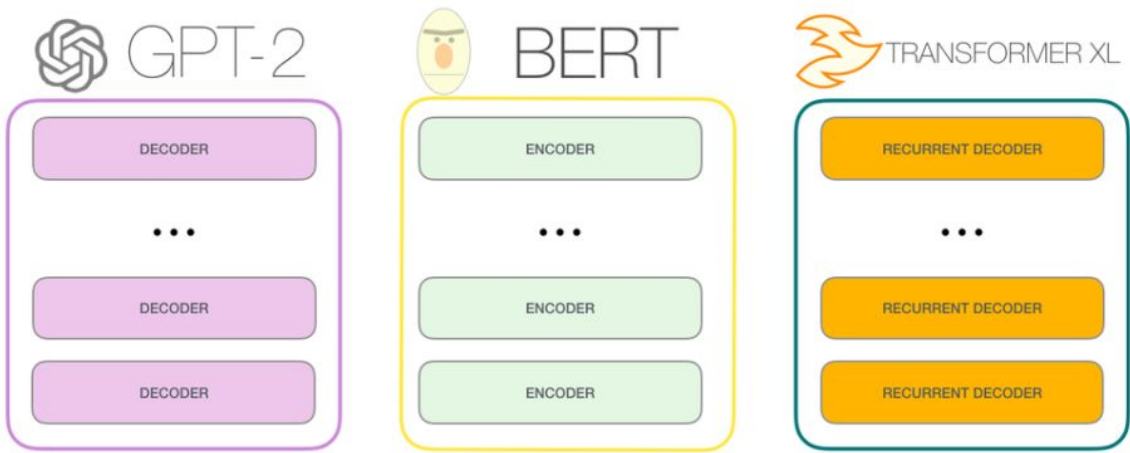
读者可以用「AllenAI GPT-2 Explorer」 (<https://gpt2.apps.allenai.org/?text=Joel%20is>) 来体验 GPT-2 模型。它可以给出可能性排名前十的下一个单词及其对应概率，你可以选择其中一个单词，然后看到下一个可能单词的列表，如此往复，最终完成一篇文章。

### 使用 Transformers 进行语言建模

正如本文作者在「The Illustrated Transformer」这篇文章中所述，原始的 transformer 模型由编码器（encoder）和解码器（decoder）组成，二者都是由被我们称为「transformer 模块」的部分堆叠而成。这种架构在机器翻译任务中取得的成功证实了它的有效性，值得一提的是，这个任务之前效果最好的方法也是基于编码器-解码器架构的。

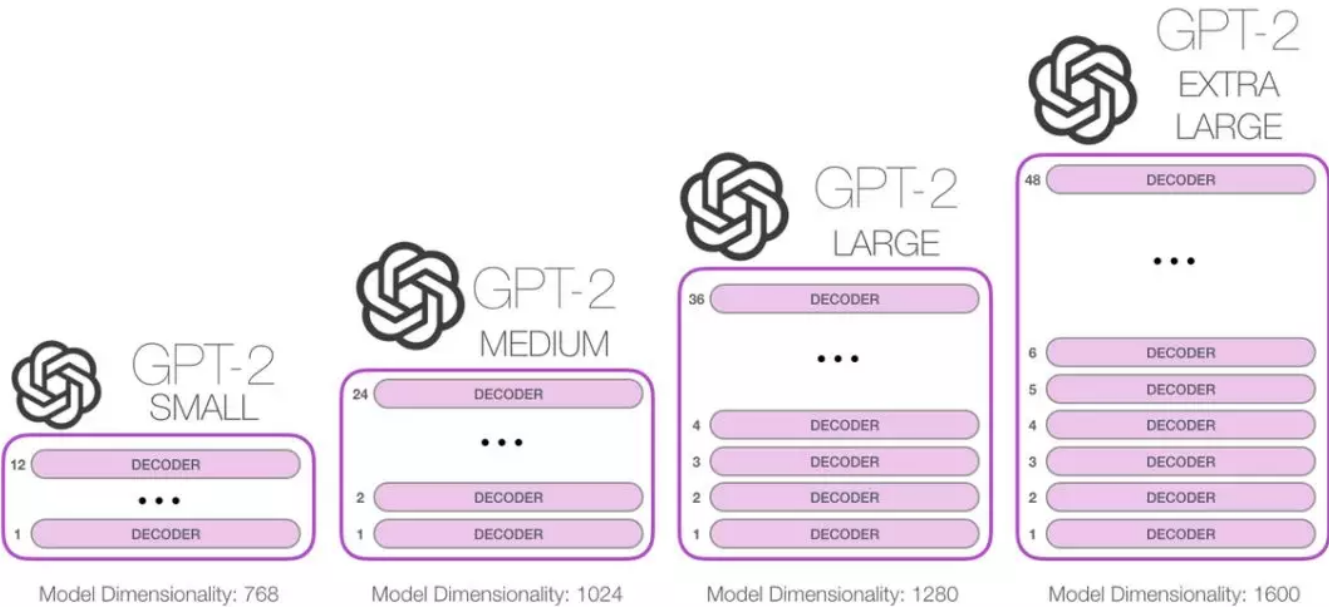


Transformer 的许多后续工作尝试去掉编码器或解码器，也就是只使用一套堆叠得尽可能多的 transformer 模块，然后使用海量文本、耗费大量的算力进行训练（研究者往往要投入数百甚至数千美元来训练这些语言模型，而在 AlphaStar 项目中则可能要花费数百万美元）。



那么我们究竟能将这些模块堆叠到多深呢？事实上，这个问题的答案也就是区别不同 GPT-2 模型的主要因素之一，如下图所示。「小号」的 GPT-2 模型堆叠了 12 层，「中号」24 层，「大号」36 层，还有一个「特大号」堆叠了整整 48 层。





与 BERT 的区别

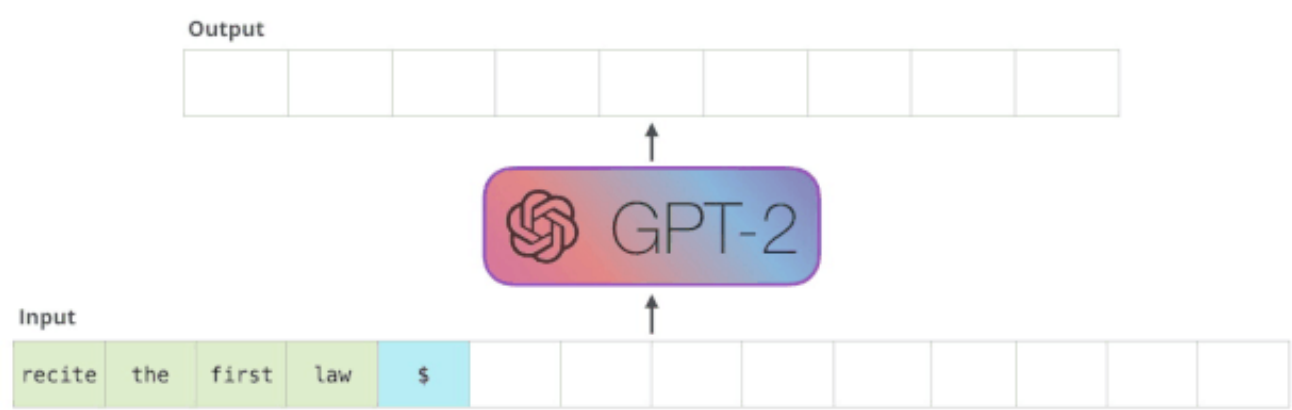
机器人第一法则

机器人不得伤害人类，或者目睹人类将遭受危险而袖手旁观。

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。我们将在下一节中详述二者的区别，但这里需要指出的是，二者一个很关键的不同之处在于：GPT-2 就像传统的语言模型一样，一次只输出一个单词（token）。下面是引导训练好的模型「背诵」机器人第一法则的例子：



这种模型之所以效果好是因为在每个新单词产生后，该单词就被添加在之前生成的单词序列后面，这个序列会成为模型下一步的新输入。这种机制叫做自回归（auto-regression），同时也是令 RNN 模型效果拔群的重要思想。



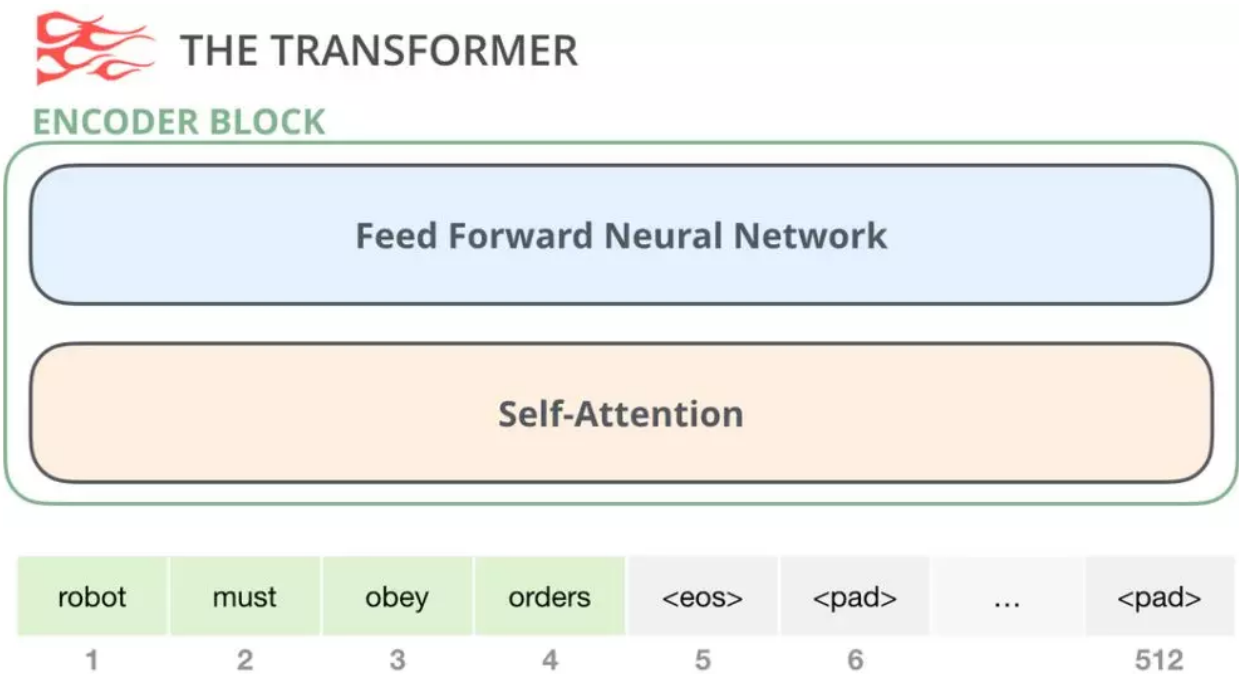
GPT-2，以及一些诸如 TransformerXL 和 XLNet 等后续出现的模型，本质上都是自回归模型，而 BERT 则不然。这就是一个权衡的问题了。虽然没有使用自回归机制，但 BERT 获得了结合单词前后的上下文信息的能力，从而取得了更好的效果。XLNet 使用了自回归，并且引入了一种能够同时兼顾前后的上下文信息的方法。

Transformer 模块的演进

原始的 transformer 论文引入了两种类型的 transformer 模块，分别是：编码器模块和解码器模块。

1. 编码器模块

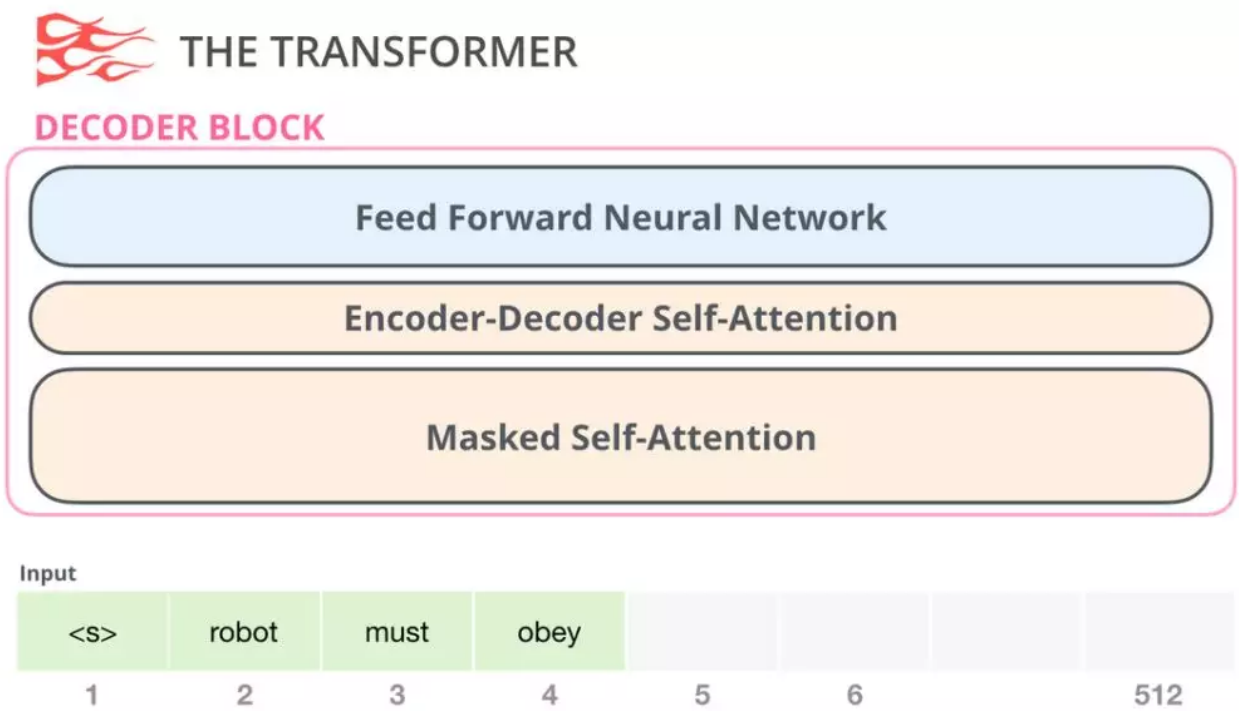
首先是编码器（encoder）模块：



原始 transformer 论文中的编码器模块可以接受长度不超过最大序列长度（如 512 个单词）的输入。如果序列长度小于该限制，我们就在其后填入预先定义的空白单词（如上图中的<pad>）。

2. 解码器模块

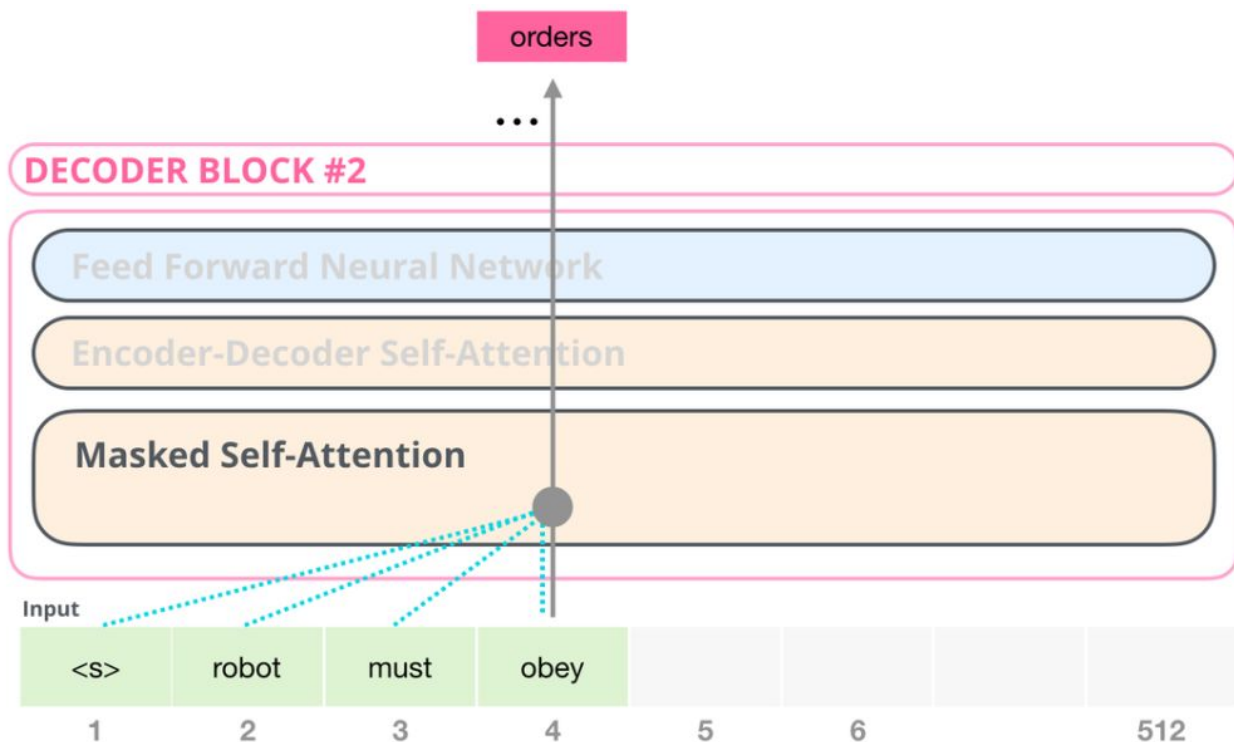
其次是解码器模块，它与编码器模块在架构上有一点小差异——加入了一层使得它可以重点关注编码器输出的某一片段，也就是下图中的编码器-解码器自注意力（encoder-decoder self-attention）层。



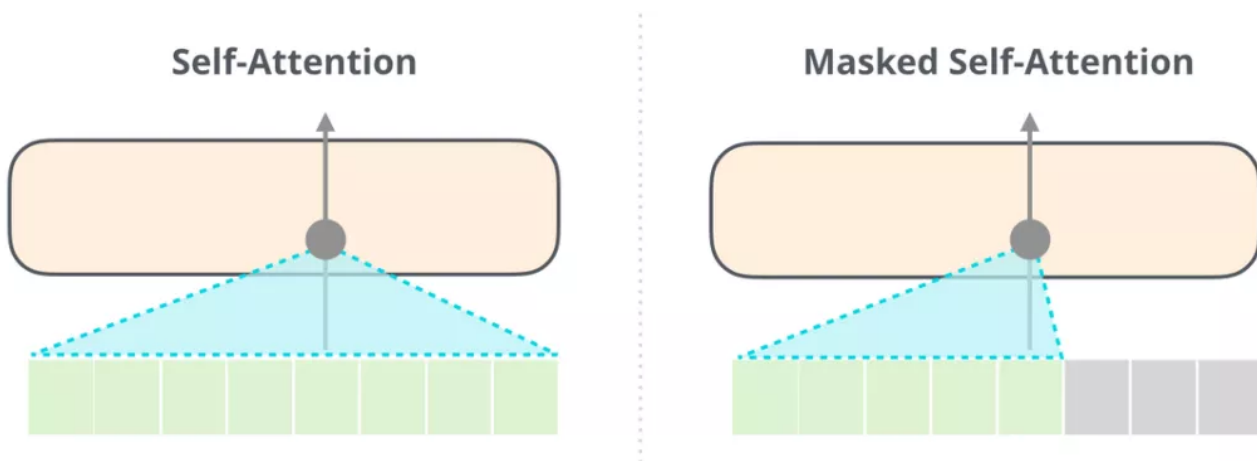


解码器在自注意力 (self-attention) 层上还有一个关键的差异: 它将后面的单词掩盖掉了。但并不像 BERT 一样将它们替换成特殊定义的单词 <mask>, 而是在自注意力计算的时候屏蔽了来自当前计算位置右边所有单词的信息。

举个例子, 如果我们重点关注 4 号位置单词及其前续路径, 我们可以模型只允许注意当前计算的单词以及之前的单词:

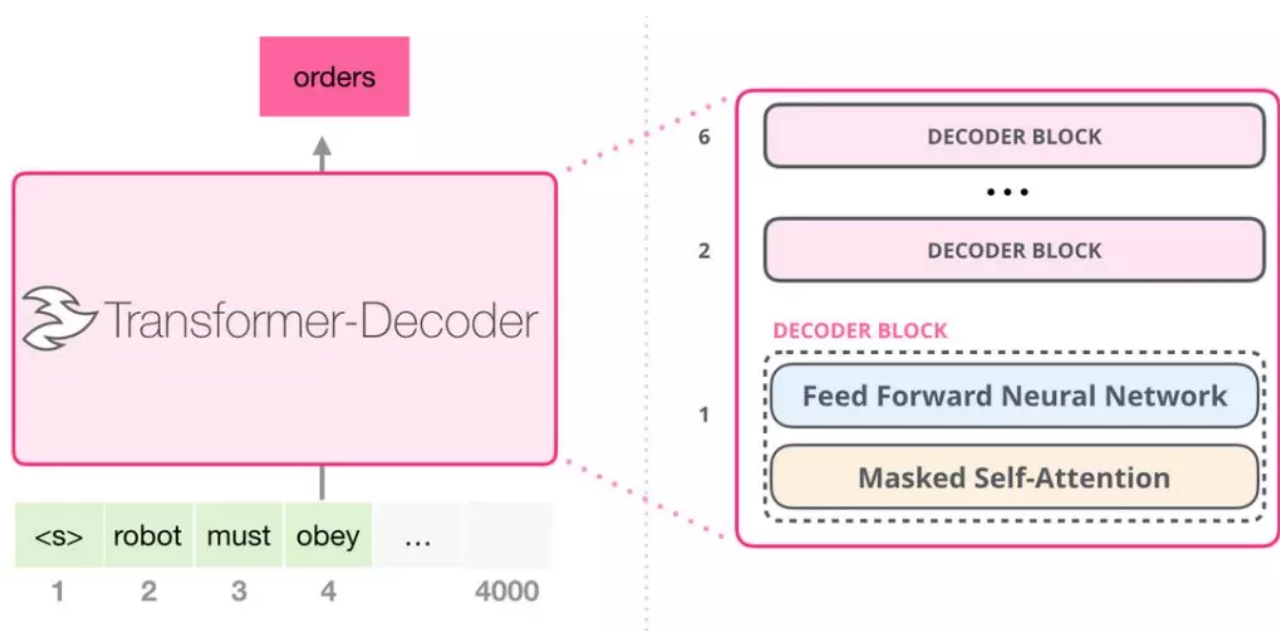


能够清楚地区分 BERT 使用的自注意力 (self-attention) 模块和 GPT-2 使用的带掩模的自注意力 (masked self-attention) 模块很重要。普通的自注意力模块允许一个位置看到它右侧单词的信息 (如下左图), 而带掩模的自注意力模块则不允许这么做 (如下右图)。



### 3. 只包含解码器的模块

在 transformer 原始论文发表之后，一篇名为「Generating Wikipedia by Summarizing Long Sequences」的论文提出用另一种 transformer 模块的排列方式来进行语言建模——它直接扔掉了所有的 transformer 编码器模块.....我们姑且就管它叫做「Transformer-Decoder」模型吧。这个早期的基于 transformer 的模型由 6 个 transformer 解码器模块堆叠而成：



图中所有的解码器模块都是一样的，因此本文只展开了第一个解码器的内部结构。可以看见，它使用了带掩模的自注意力层。请注意，该模型在某个片段中可以支持最长 4000 个单词的序列，相较于 transformer 原始论文中最长 512 单词的限制有了很大的提升。

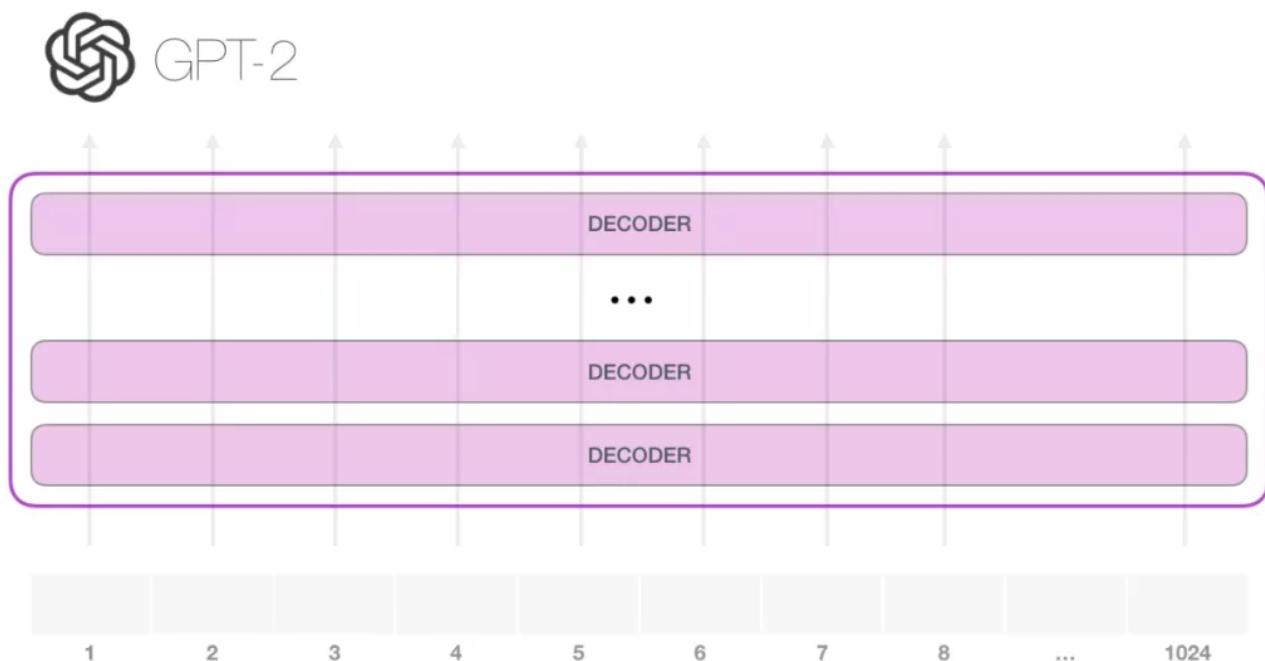
这些解码器模块和 transformer 原始论文中的解码器模块相比，除了去除了第二个自注意力层之外，并无很大不同。一个相似的架构在字符级别的语言建模中也被验证有效，它使用更深的自注意力层构建语言模型，一次预测一个字母/字符。

OpenAI 的 GPT-2 模型就用了这种只包含编码器（decoder-only）的模块。

GPT-2 内部机制速成

在我内心，字字如刀；电闪雷鸣，使我疯癫。  
——Budgie

接下来，我们将深入剖析 GPT-2 的内部结构，看看它是如何工作的。

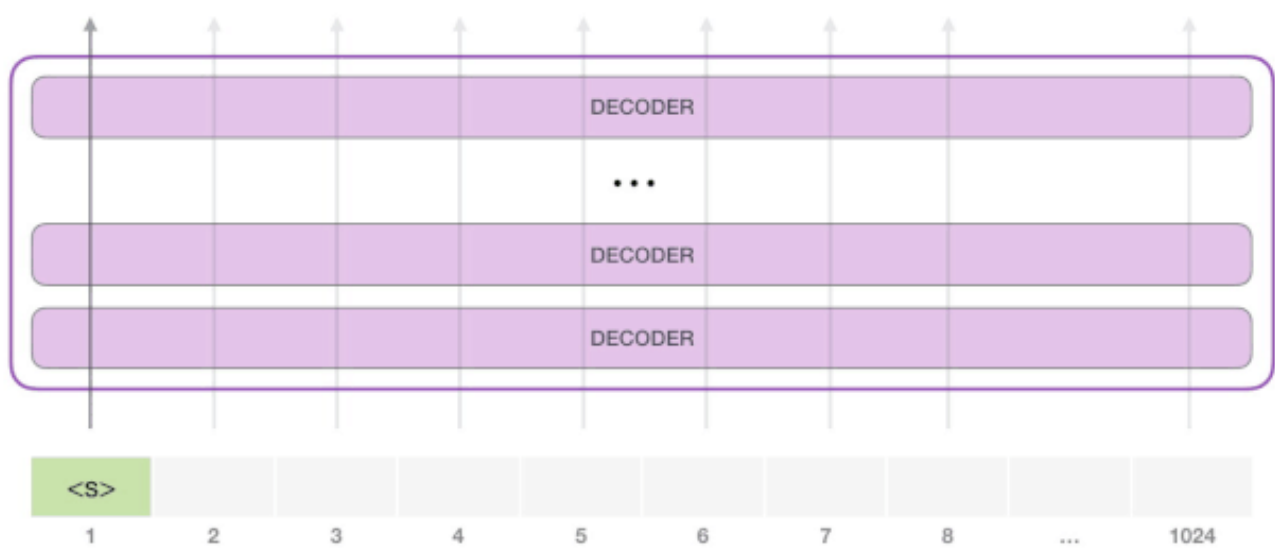


GPT-2 可以处理最长 1024 个单词的序列。每个单词都会和它的前续路径一起「流过」所有的解码器模块。

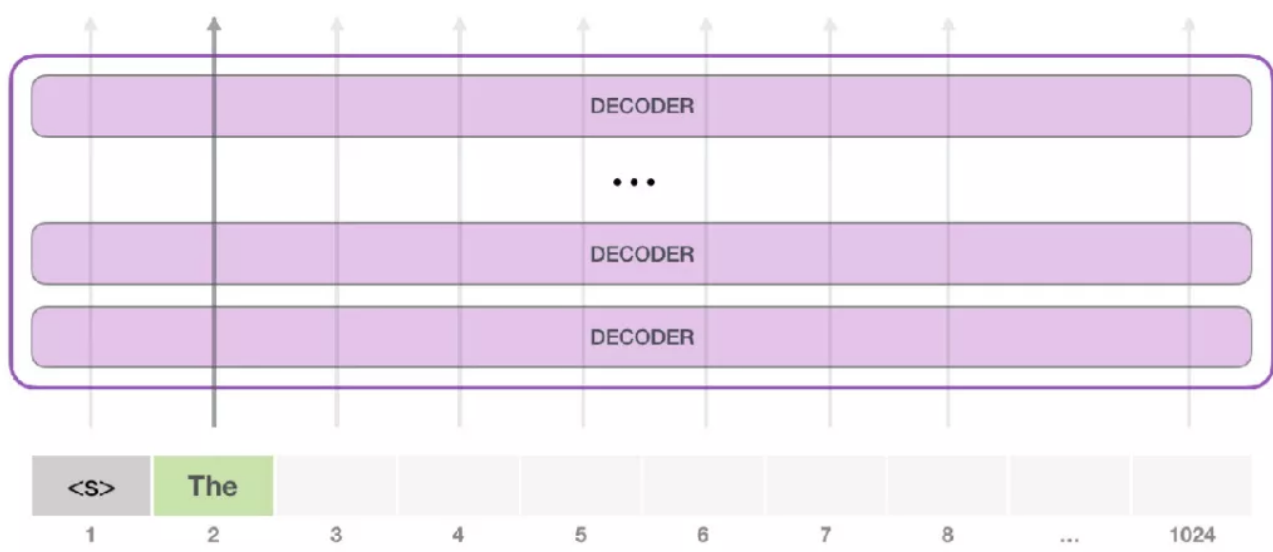
想要运行一个训练好的 GPT-2 模型，最简单的方法就是让它自己随机工作（从技术上说，叫做生成无条件样本）。换句话说，我们也可以给它一点提示，让它说一些关于特定主题的话（即生成交互式条件样本）。在随机情况下，我们只简单地提供一个预先定义好的起始单词（训练好的模型使用「`<|endoftext|>`」作为它的起始单词，不妨将其称为 `<s>`），然后让它自己生成文字。

此时，模型的输入只有一个单词，所以只有这个单词的路径是活跃的。单词经过层层处理，最终得到一个向量。向量可以对于词汇表的每个单词计算一个概率（词汇表是模型能「说出」的所有单词，GPT-2 的词汇表中有 50000 个单词）。在本例中，我们选择概率最高的单词「The」作为下一个单词。

但有时这样会出问题——就像如果我们持续点击输入法推荐单词的第一个，它可能会陷入推荐同一个词的循环中，只有你点击第二或第三个推荐词，才能跳出这种循环。同样的，GPT-2 也有一个叫做「top-k」的参数，模型会从概率前 k 大的单词中抽样选取下一个单词。显然，在之前的情况下， $\text{top-k} = 1$ 。



接下来，我们将输出的单词添加在输入序列的尾部构建新的输入序列，让模型进行下一步的预测：

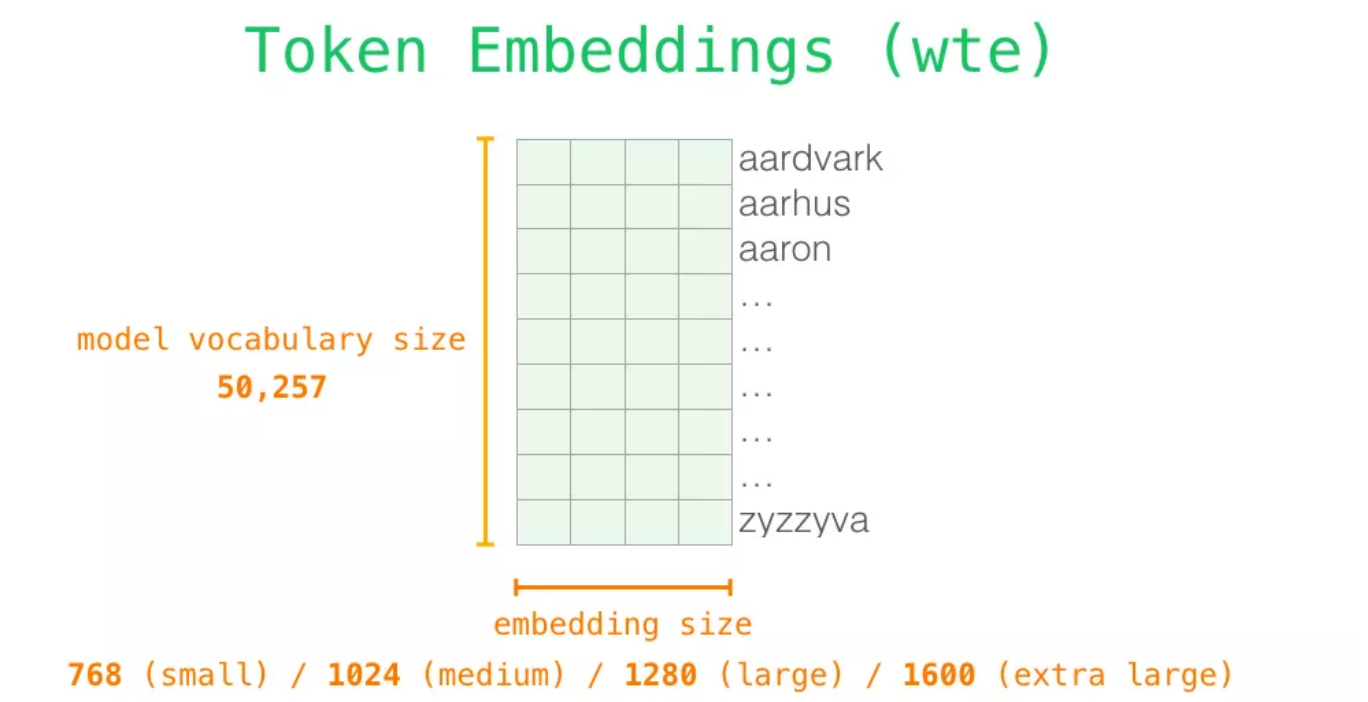


请注意，第二个单词的路径是当前唯一活跃的路径了。GPT-2 的每一层都保留了它们对第一个单词的解释，并且将运用这些信息处理第二个单词（具体将在下面一节对自注意力机制的讲解中详述），GPT-2 不会根据第二个单词重新解释第一个单词。

更加深入了解内部原理

1. 输入编码

让我们更加深入地了解一下模型的内部细节。首先，让我们从模型的输入开始。正如我们之前讨论过的其它自然语言处理模型一样，GPT-2 同样从嵌入矩阵中查找单词对应的嵌入向量，该矩阵也是模型训练结果的一部分。

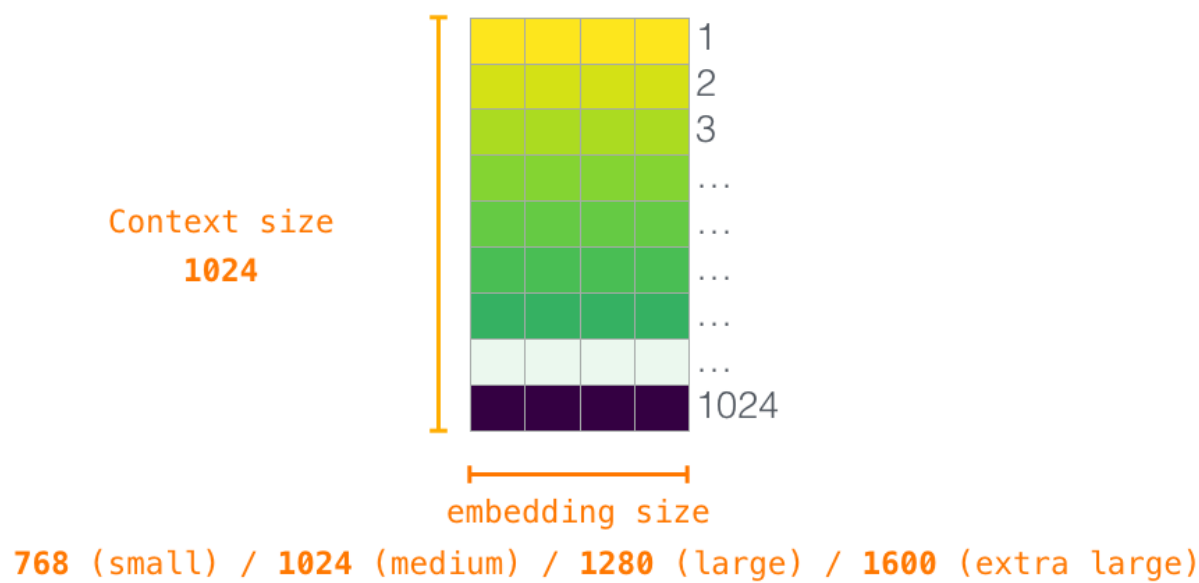


每一行都是一个词嵌入向量：一个能够表征某个单词，并捕获其意义的数字列表。嵌入向量的长度和GPT-2 模型的大小有关，最小的模型使用了长为 768 的嵌入向量来表征一个单词。

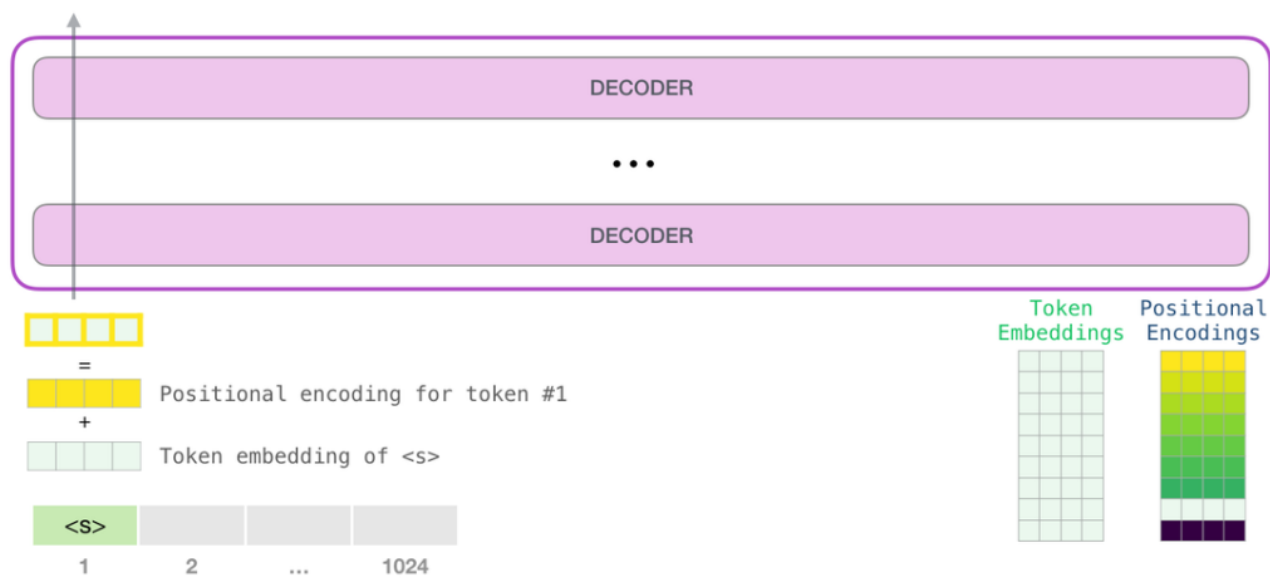
所以在一开始，我们需要在嵌入矩阵中查找起始单词 <s> 对应的嵌入向量。但在将其输入给模型之前，我们还需要引入位置编码——一些向 transformer 模块指出序列中的单词顺序的信号。1024 个输入序列位置中的每一个都对应一个位置编码，这些编码组成的矩阵也是训练模型的一部分。



# Positional Encodings (wpe)



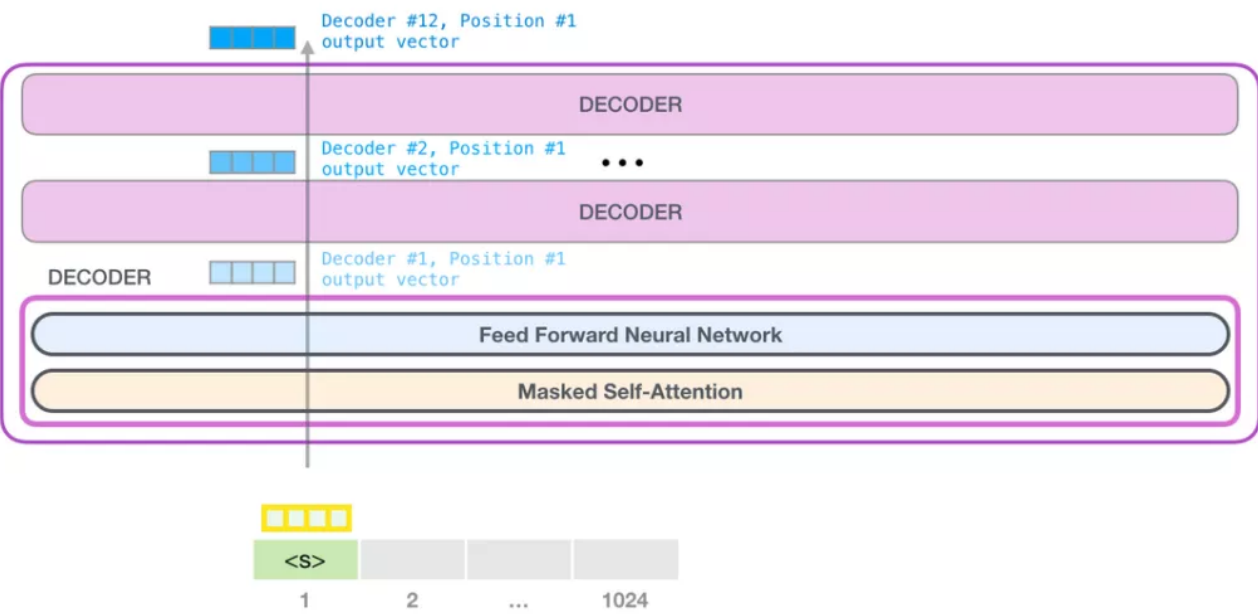
至此，输入单词在进入模型第一个 transformer 模块之前所有的处理步骤就结束了。如上文所述，训练后的 GPT-2 模型包含两个权值矩阵：嵌入矩阵和位置编码矩阵。



将单词输入第一个 transformer 模块之前需要查到它对应的嵌入向量，再加上 1 号位置位置对应的位置向量。

### 3. 堆栈之旅

第一个 transformer 模块处理单词的步骤如下：首先通过自注意力层处理，接着将其传递给神经网络层。第一个 transformer 模块处理完但此后，会将结果向量被传入堆栈中的下一个 transformer 模块，继续进行计算。每一个 transformer 模块的处理方式都是一样的，但每个模块都会维护自己的自注意力层和神经网络层中的权重。



4. 回顾自注意力机制

语言的含义是极度依赖上下文的，比如下面这个机器人第二法则：

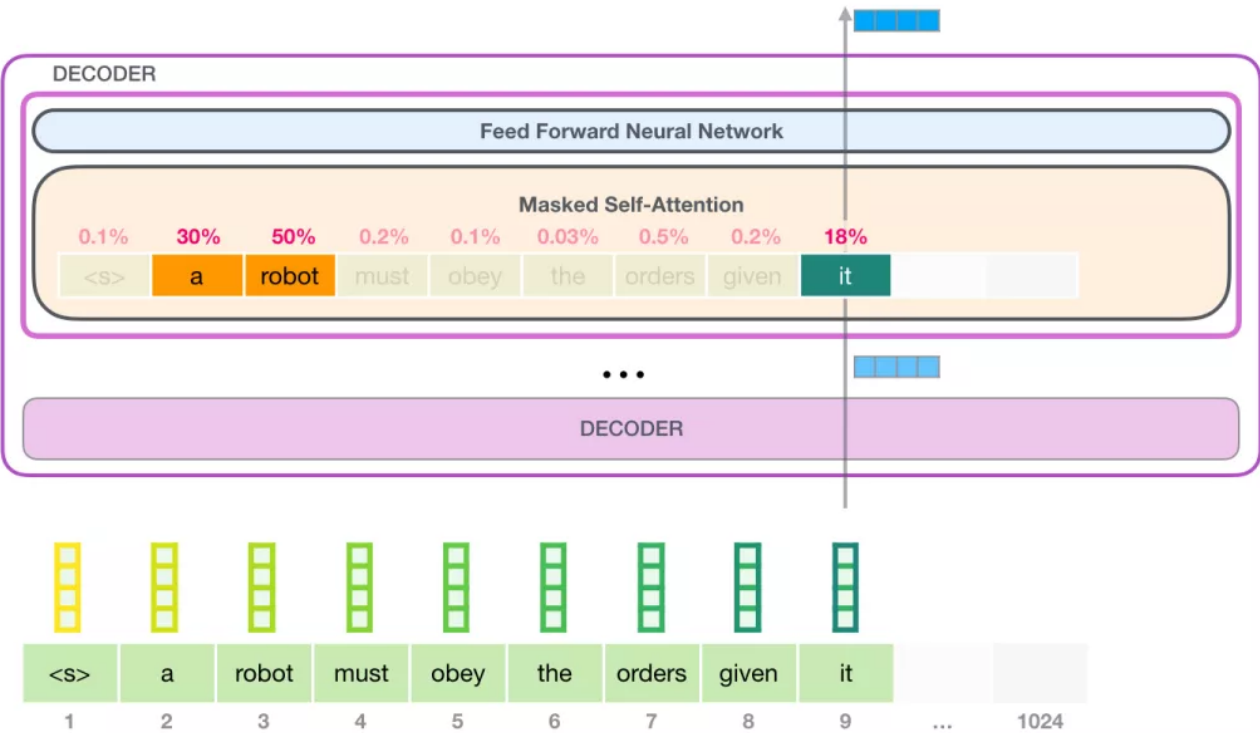
*机器人第二法则*  
机器人必须遵守人类给它的命令，除非该命令违背了第一法则。

我在这句话中高亮表示了三个地方，这三处单词指代的是其它单词。除非我们知道这些词指代的上下文联系起来，否则根本不可能理解或处理这些词语的意思。当模型处理这句话的时候，它必须知道：

- 「它」指代机器人
- 「命令」指代前半句话中人类给机器人下的命令，即「人类给它的命令」
- 「第一法则」指机器人第一法则的完整内容

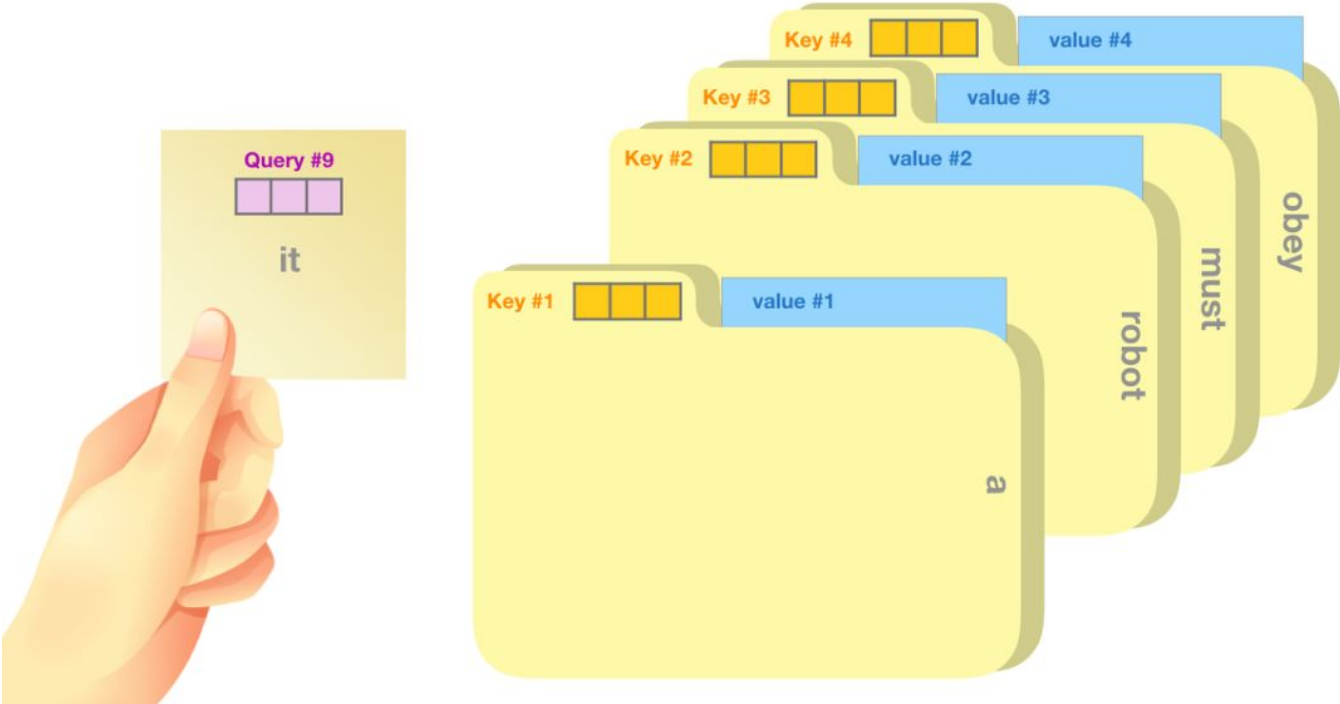
这就是自注意力机制所做的工作，它在处理每个单词（将其传入神经网络）之前，融入了模型对于用来解释某个单词的上下文的相关单词的理解。具体做法是，给序列中每一个单词都赋予一个相关度得分，之后对他们的向量表征求和。

举个例子，最上层的 transformer 模块在处理单词「it」的时候会关注「a robot」，所以「a」、「robot」、「it」这三个单词与其得分相乘加权求和后的特征向量会被送入之后的神经网络层。



自注意力机制沿着序列中每一个单词的路径进行处理，主要由 3 个向量组成：

1. 查询向量（Query 向量）：当前单词的查询向量被用来和其它单词的键向量相乘，从而得到其它词相对于当前词的注意力得分。我们只关心目前正在处理的单词的查询向量。
2. 键向量（Key 向量）：键向量就像是序列中每个单词的标签，它使我们搜索相关单词时用来匹配的对象。
3. 值向量（Value 向量）：值向量是单词真正的表征，当我们算出注意力得分后，使用值向量进行加权求和得到能代表当前位置上下文的向量。










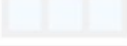

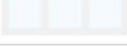









一个简单粗暴的比喻是在档案柜中找文件。查询向量就像一张便利贴，上面写着你正在研究的课题。键向量像是档案柜中文件夹上贴的标签。当你找到和便利贴上所写相匹配的文件夹时，拿出它，文件夹里的东西便是值向量。只不过我们最后找的并不是单一的值向量，而是很多文件夹值向量的混合。

将单词的查询向量分别乘以每个文件夹的键向量，得到各个文件夹对应的注意力得分（这里的乘指的是向量点乘，乘积会通过 softmax 函数处理）。



我们将每个文件夹的值向量乘以其对应的注意力得分，然后求和，得到最终自注意力层的输出。

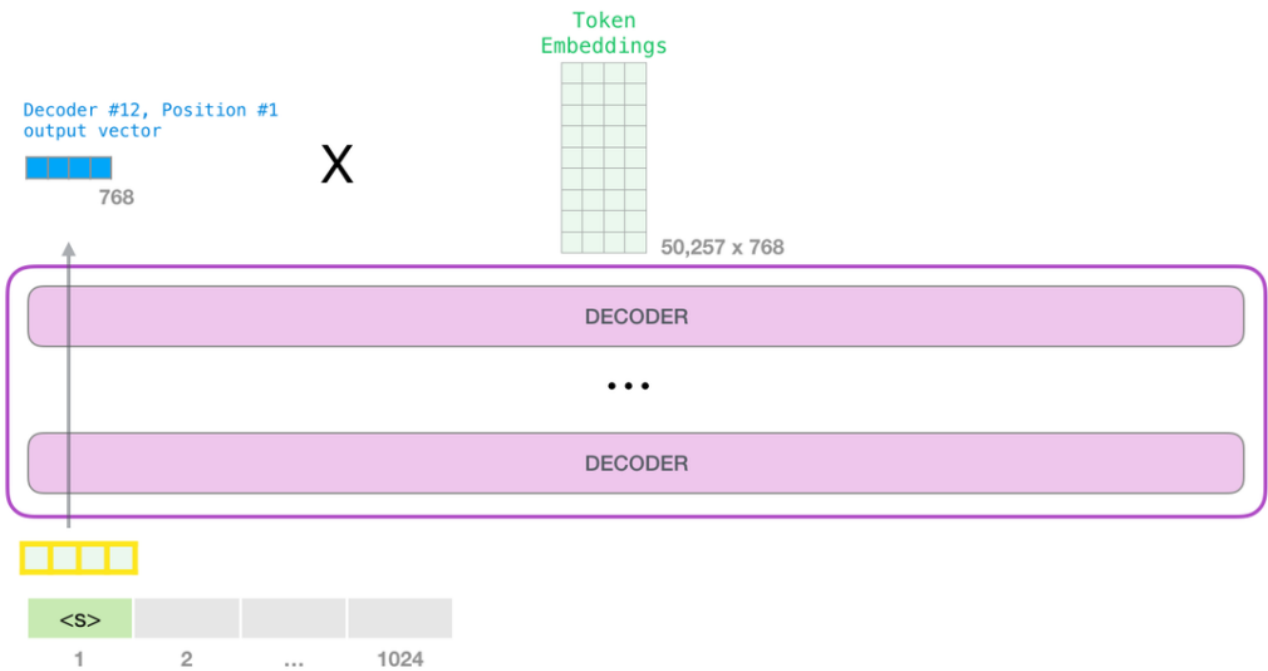
Word	Value vector	Score	Value X Score
<s>		0.001	
a		0.3	
robot		0.5	
must		0.002	
obey		0.001	
the		0.0003	
orders		0.005	
given		0.002	
it		0.19	
		Sum:	

这样将值向量加权混合得到的结果是一个向量，它将其 50% 的「注意力」放在了单词「robot」上，30% 的注意力放在了「a」上，还有 19% 的注意力放在「it」上。我们之后还会更详细地讲解自注意力机制，让我们先继续向前探索 transformer 堆栈，看看模型的输出。

5. 模型输出

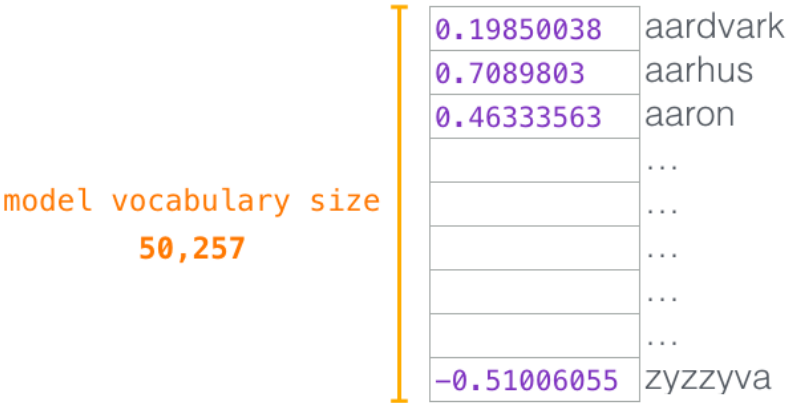
当最后一个 transformer 模块产生输出之后（即经过了它自注意力层和神经网络层的处理），模型会将输出的向量乘上嵌入矩阵。



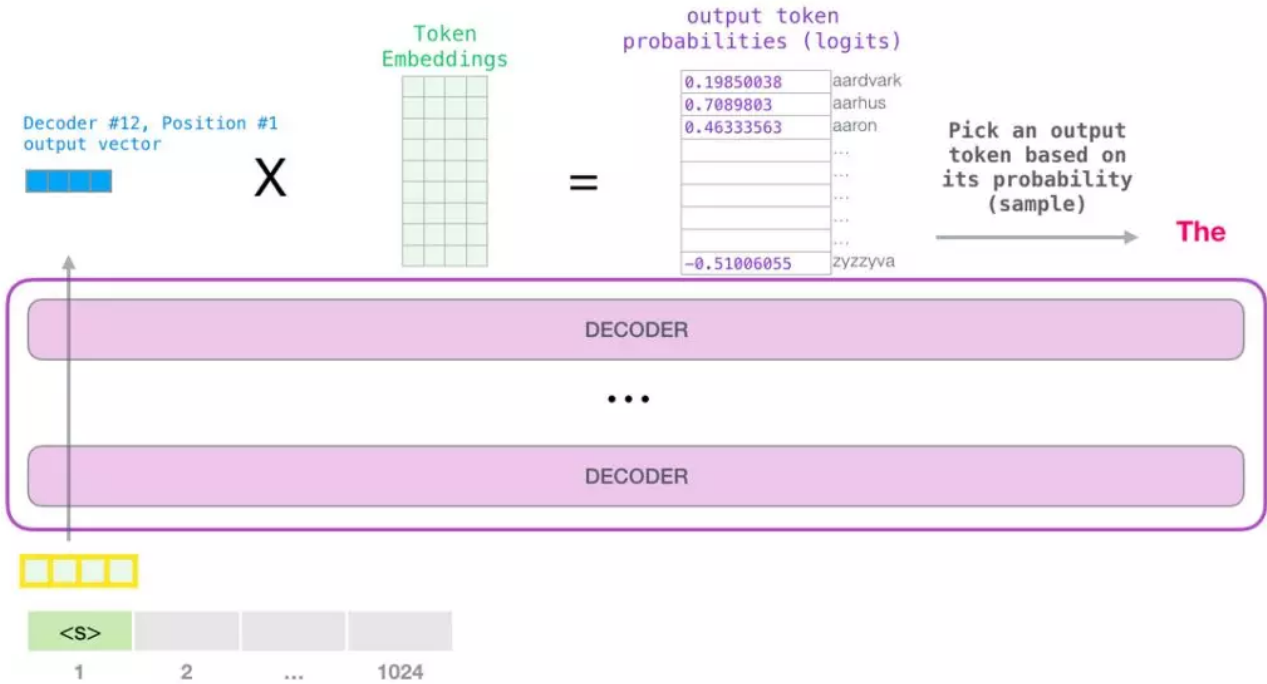


我们知道，嵌入矩阵的每一行都对应模型的词汇表中一个单词的嵌入向量。所以这个乘法操作得到的结果就是词汇表中每个单词对应的注意力得分。

## output token probabilities (logits)



我们简单地选取得分最高的单词作为输出结果（即 top-k = 1）。但其实如果模型考虑其他候选单词的话，效果通常会更好。所以，一个更好的策略是对于词汇表中得分较高的一部分单词，将它们的得分作为概率从整个单词列表中进行抽样（得分越高的单词越容易被选中）。通常一个折中的方法是，将 top-k 设为 40，这样模型会考虑注意力得分排名前 40 位的单词。



这样，模型就完成了一轮迭代，输出了一个单词。模型会接着不断迭代，直到生成一个完整的序列——序列达到 1024 的长度上限或序列中产生了一个终止符。

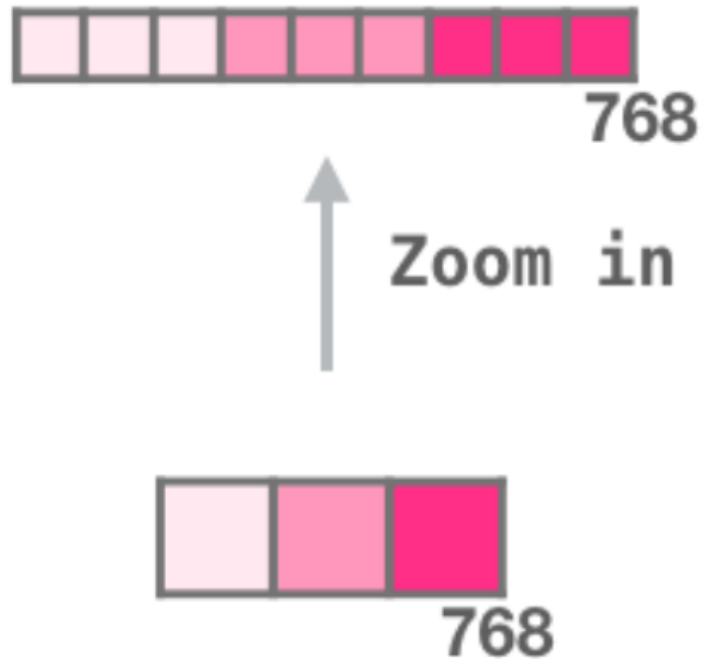
第一部分结语：大家好，这就是 GPT-2

本文是 GPT-2 模型工作原理的一个概览。如果你还是对自注意力层内部深层的细节很好奇，请继续关注机器之心的系列文章。我们将引入更多可视化语言来试着解释自注意力层的工作原理，同时也是为了能够更好地描述之后基于 transformer 的模型（说的就是你们，TransformerXL 还有 XLNet）。

这篇文章中有一些过分简化的地方：

- 1. 混用了「单词」（word）和「词」（token）这两个概念。但事实上，GPT-2 使用字节对编码（Byte Pair Encoding）方式来创建词汇表中的词（token），也就是说词（token）其实通常只是单词的一部分。
- 2. 举的例子其实是 GPT-2 在「推断/评价」（inference / evaluation）模式下运行的流程，所以一次只处理一个单词。在训练过程中，模型会在更长的文本序列上进行训练，并且一次处理多个词（token）。训练过程的批处理大小（batch size）也更大（512），而评价时的批处理大小只有 1。
- 3. 为了更好地组织空间中的图像，作者画图时随意转置了向量，但在实现时需要更精确。
- 4. Transformer 模块使用了很多归一化（normalization）层，这在训练中是很关键的。我们在「The Illustrated Transformer」（<https://jalamar.github.io/illustrated-transformer/>）译文中提到了其中一些，但本文更关注自注意力层。

5. 有时文章需要用更多的小方块来代表一个向量，我把这些情况叫做「放大」，如下图所示。



原文地址: <https://jalammar.github.io/illustrated-gpt2/>

