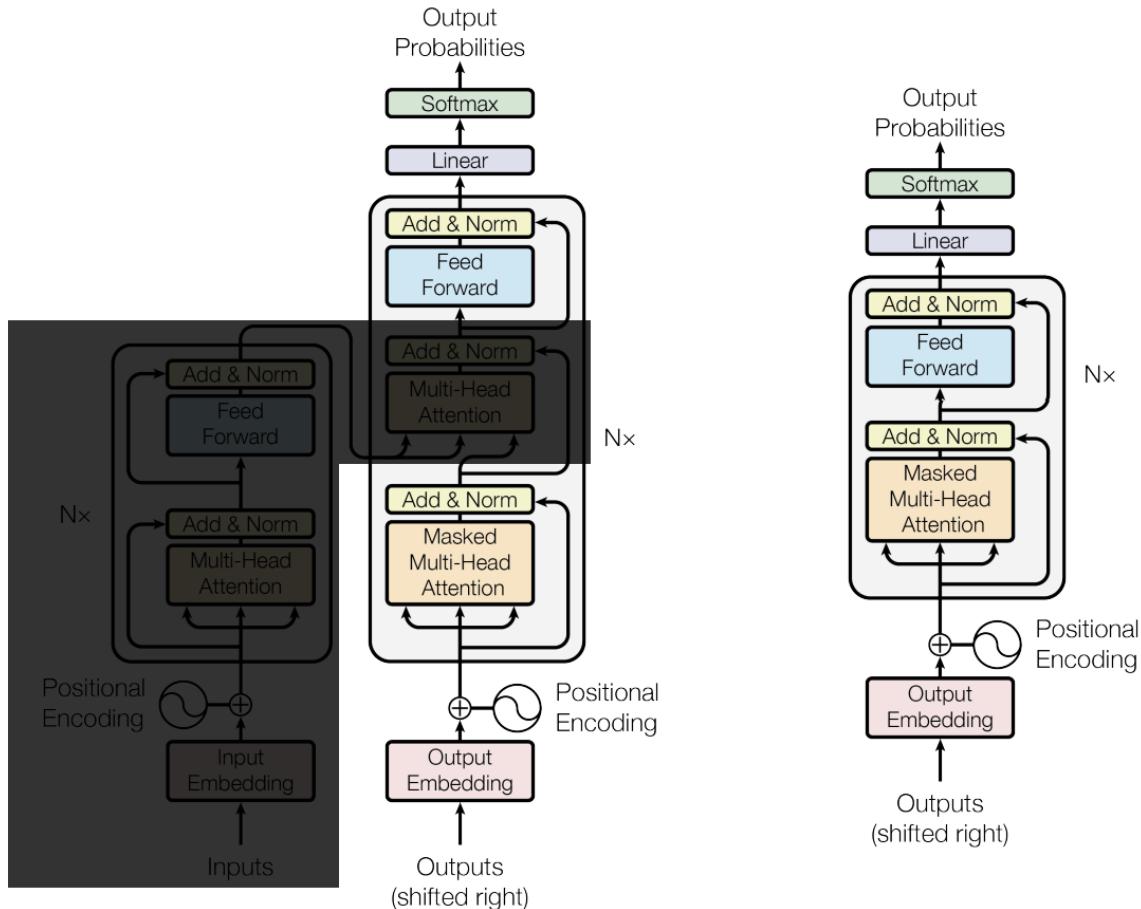


GPT

GPT结构



📌 GPT (Generative Pretrained Transformer) 采用的是**Decoder-Only**架构。简单来说，它基于Transformer模型，但仅使用Transformer的**解码器部分**，而不包含编码器。下面是这一结构的主要特点和工作原理：

1. Transformer架构概述

Transformer模型由两部分组成：编码器（Encoder）和解码器（Decoder）。编码器主要用于处理输入序列，将其转换为上下文表示，而解码器则生成输出序列。GPT只使用了Transformer的解码器部分。

2. 解码器（Decoder）结构

GPT的解码器结构由多个相同的层（Layer）堆叠而成，每一层都包含以下几个关键组件：

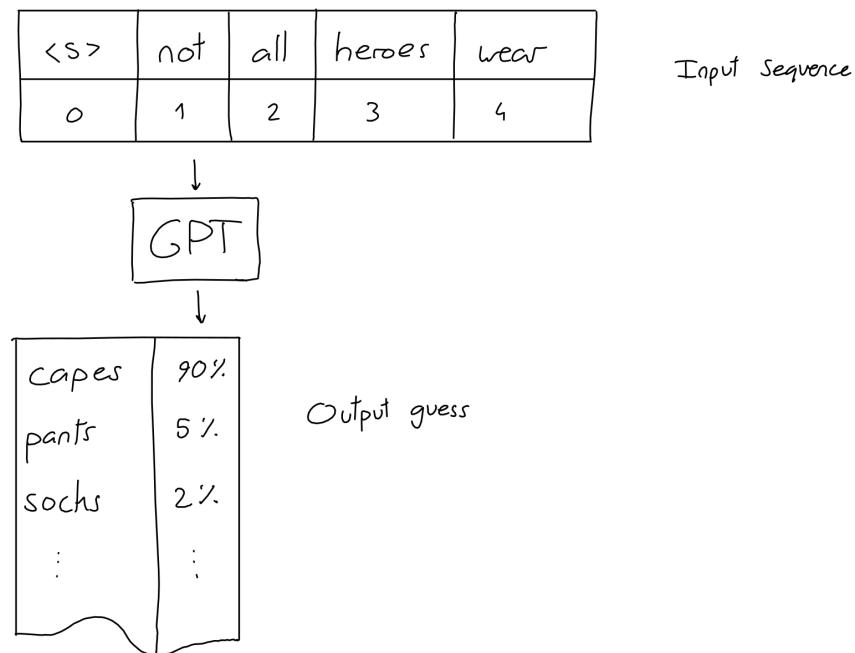
- **自注意力机制（Self-Attention）**：通过自注意力机制，模型能够根据输入序列中的各个位置的上下文信息来调整对不同位置的关注权重。

- **前馈神经网络 (Feedforward Network)**：通常由两个全连接层组成，用于进一步处理自注意力层的输出。
- **层归一化 (Layer Normalization)**：用于对每一层的输出进行标准化，改善训练的稳定性。

GPT微观流程

输入和输出

 在我们了解其他内容之前，我们需要知道：GPT 的输入和输出是什么？



 输入是 N 个单词（又称 token）的序列。输出是对最有可能放在输入序列末尾的单词的猜测。

就是这样！你看到的所有令人印象深刻的 GPT 对话、故事和示例都是用这个简单的输入输出方案制作的：给它一个输入序列 - 获取下一个单词。

Not all heroes wear -> capes

当然，我们经常想得到多个单词，但这不是问题：得到下一个单词后，我们将其添加到序列中，并得到下一个单词。

Not all heroes wear capes -> but

Not all heroes wear capes but -> all

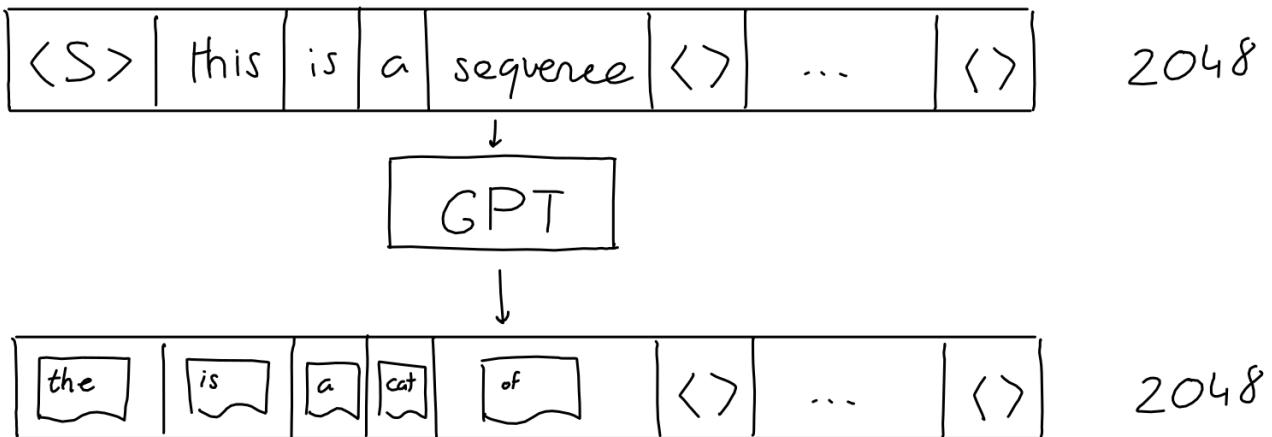
Not all heroes wear capes but all -> villans

Not all heroes wear capes but all villans -> do

根据需要重复，最终你会得到生成的长文本。

其实准确地说，我们需要从两个方面去纠正上述内容。

- 输入序列实际上固定为 2048 个字（对于 GPT-3）。我们仍然可以传递短序列作为输入：我们只需用“空”值填充所有额外的位置。
- GPT 输出不只是一个猜测，而是一个猜测序列（长度为 2048）（每个可能单词的概率）。序列中的每个“下一个”位置都有一个猜测。但在生成文本时，我们通常只查看序列中最后一个单词的猜测。



编码

但 GPT 实际上无法理解单词。作为一种深度学习算法，它对数字向量进行操作。那么我们如何将单词转换成向量呢？

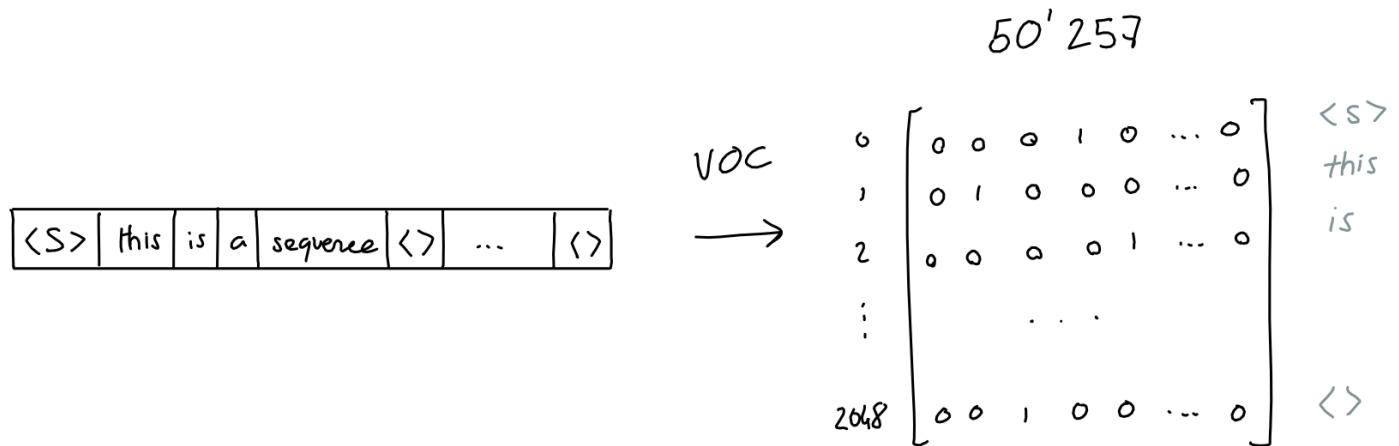
第一步是保留所有单词的词汇表，这样我们就可以为每个单词赋予一个值。Aardvark 为 0，aaron 为 1，依此类推。（GPT 的词汇表有 50257 个单词）。

注意：为了提高效率，GPT-3 实际上使用了字节级字节对编码 (BPE) 标记化。这意味着词汇表中的“单词”不是完整的单词，而是文本中经常出现的字符组（对于字节级 BPE，为字节）。使用 GPT-3 字节级 BPE 标记器，“**Not all heroes wear capes**”被拆分为标记“Not” “all” “heroes” “wear” “cap” “es”，词汇表中的 ID 为 3673、477、10281、5806、1451、274。

因此，我们可以将每个单词变成一个大小为 50257 的独热编码向量，其中只有索引 i（单词的值）处的维度为 1，其他所有维度均为 0。



钉子图标 当然，我们对序列中的每个单词都这样做，



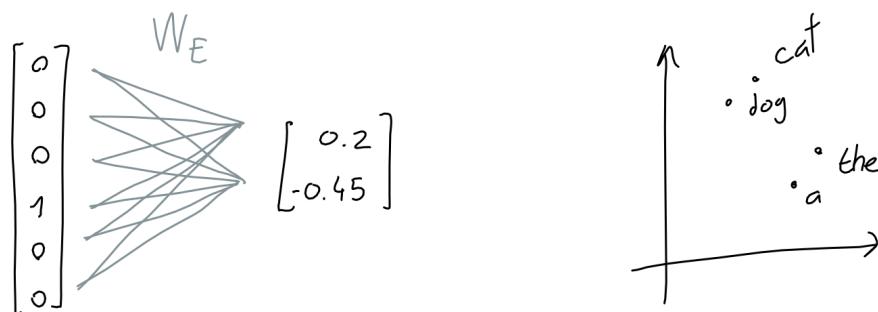
其结果是一个由 1 和 0 组成的 2048×50257 矩阵。

嵌入

钉子图标 50257 作为 one-hot 向量来说相当大，而且其中大部分都是零，且 token 与 token 之间彼此正交。这浪费了很多空间。

为了解决这个问题，我们学习了一个嵌入函数：一个神经网络，它接受一个长度为 50257 的 1 和 0 向量，并输出一个长度为 n 的数字向量。在这里，我们试图将单词含义的信息存储（或投影）到更小的维度空间。

例如，如果嵌入维度为 2，则就像将每个单词存储在 2D 空间中的特定坐标处。



钉子图标 当然，嵌入维度通常大于 2：GPT 使用 12288 维度。

实际上，每个单词的独热向量都会与学习到的嵌入权重相乘，最终得到一个 12288 维的嵌入向量。

从算术角度来说，我们将 2048×50257 序列编码矩阵与 50257×12288 嵌入权重矩阵（学习到）相乘，最终得到 2048×12288 序列嵌入矩阵。

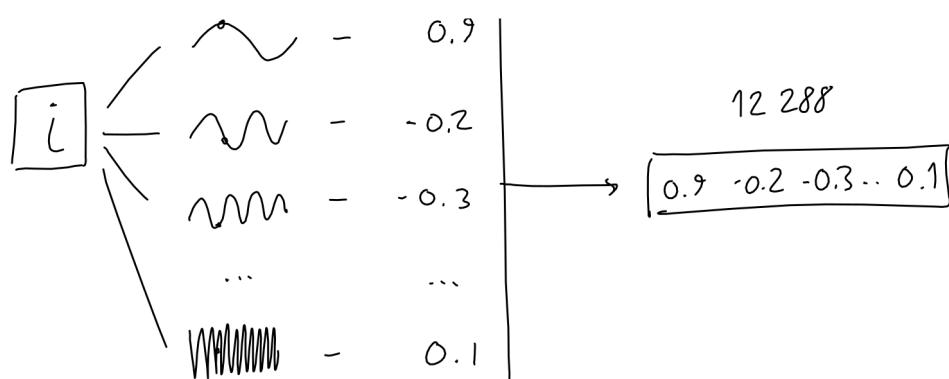
$$\begin{matrix} & 50257 \\ \begin{matrix} 0 \\ 1 \\ 2 \\ \vdots \\ 2048 \end{matrix} & \times \begin{matrix} 50257 \\ W_E \end{matrix} = \begin{matrix} 12288 \\ 2048 \end{matrix} \begin{bmatrix} 0.1 & \dots & 0.2 \\ \vdots & \ddots & \vdots \\ 0.3 & \dots & 2.5 \end{bmatrix} \end{matrix}$$
$$\begin{matrix} 2048 & \xrightarrow{\quad\quad\quad} & 12288 \\ \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \end{matrix} & \xrightarrow{W_E} & \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \end{matrix} \end{matrix}$$

其结果等价于直接将 $\text{token_id}[x]$ 告诉 **嵌入权重矩阵**，然后从权重矩阵的第 $\{x\}$ 行取出向量，并将 2048 行向量直接相拼接。

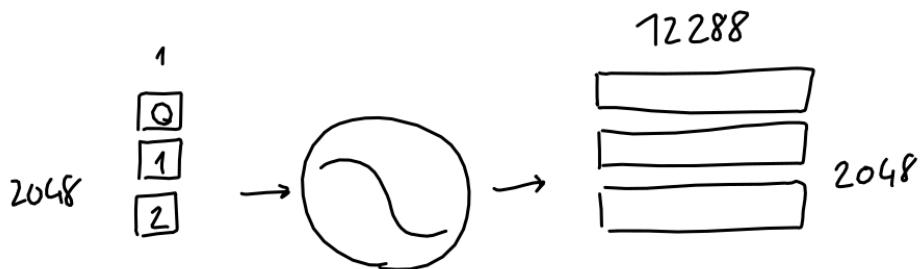
这意味着：到目前为止，序列中的 token 没有信息交互。

位置编码

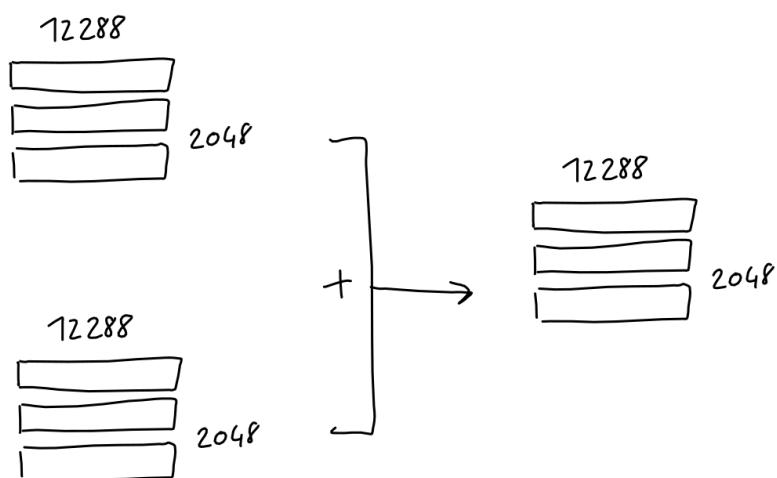
为了对序列中当前标记的位置进行编码，作者获取标记的位置（标量 i ，在 $[0-2047]$ 中）并将其实例化为 12288 个正弦函数，每个函数都有不同的频率。



对于每个标记，结果是 12288 个数字向量。与嵌入一样，我们将这些向量组合成一个具有 2048 行的矩阵，其中每行是序列中标记的 12288 列位置编码。



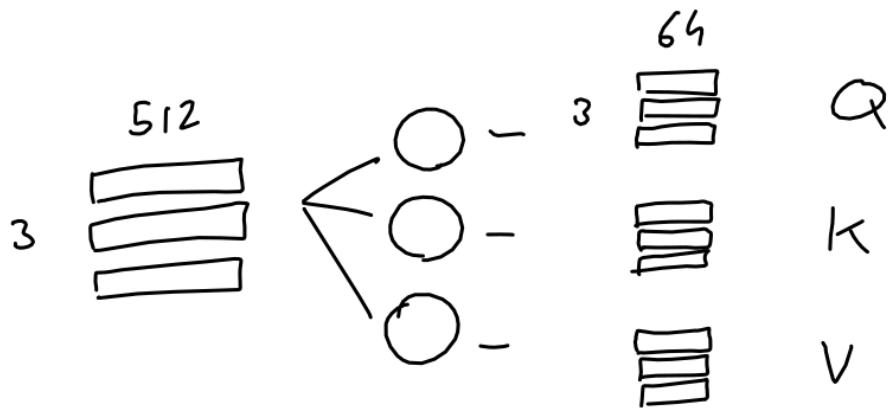
最后，这个序列位置编码矩阵具有与序列嵌入矩阵相同的形状，可以简单地添加到其中。



Attention (简化版)

简单来说，注意力机制的目的在于：对于序列中的每个输出，预测需要关注哪些输入token以及关注程度。这里，想象一个由 3 个 token 组成的序列，每个 token 都用 512 个值的嵌入表示。

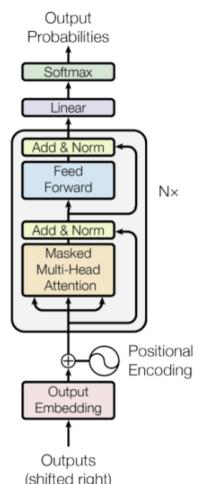
该模型学习了 3 个线性投影，它们都应用于序列嵌入。换句话说，我们学习了 3 个权重矩阵，它们将我们的序列嵌入转换为三个独立的 3×64 矩阵，每个矩阵用于不同的任务。



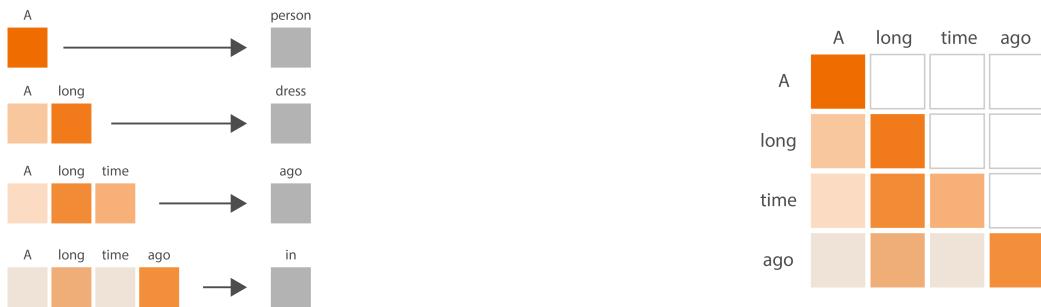
钉子图标 前两个矩阵（“查询”和“键”）相乘 (QK^T)，得到一个 3×3 矩阵。该矩阵（通过 softmax 归一化）表示每个 token 对其他 token 的重要性。

注意：此 (QK^T) 是 GPT 中唯一一个跨序列单词进行操作的运算。它是唯一一个矩阵行交互的运算。

$$Q \times K^T = \begin{bmatrix} 3 \\ 512 \end{bmatrix} \times \begin{bmatrix} 3 \\ 512 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 4.1 & -1 & 0.1 \\ 1 & . & 2.2 \\ . & 1.2 & -4. \end{bmatrix}$$

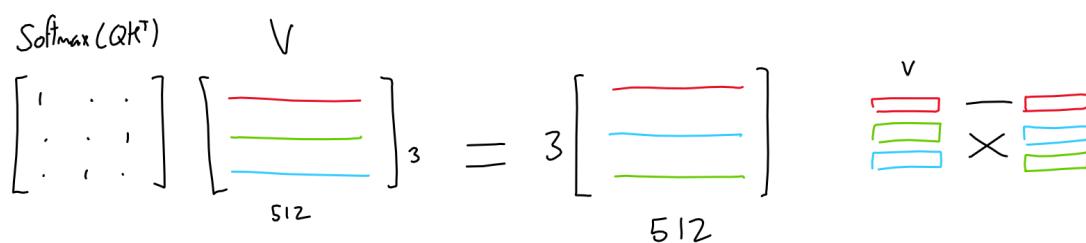


$$\text{Softmax}(QK^T) = \begin{bmatrix} 0 & 1 & 2 \\ 0.9 & 0.1 & 0.2 \\ 1 & . & 0.8 \\ . & 0.9 & 0.3 \end{bmatrix} \quad \begin{array}{ccc} 0 & 1 & 2 \\ \diagdown & \diagup & \diagdown \\ 1 & & 2 \end{array}$$



第三个矩阵（“值”）与这个重要性矩阵相乘，结果是，对于每个token，都混合了所有其他token的值，并根据其各自token的重要性进行加权。

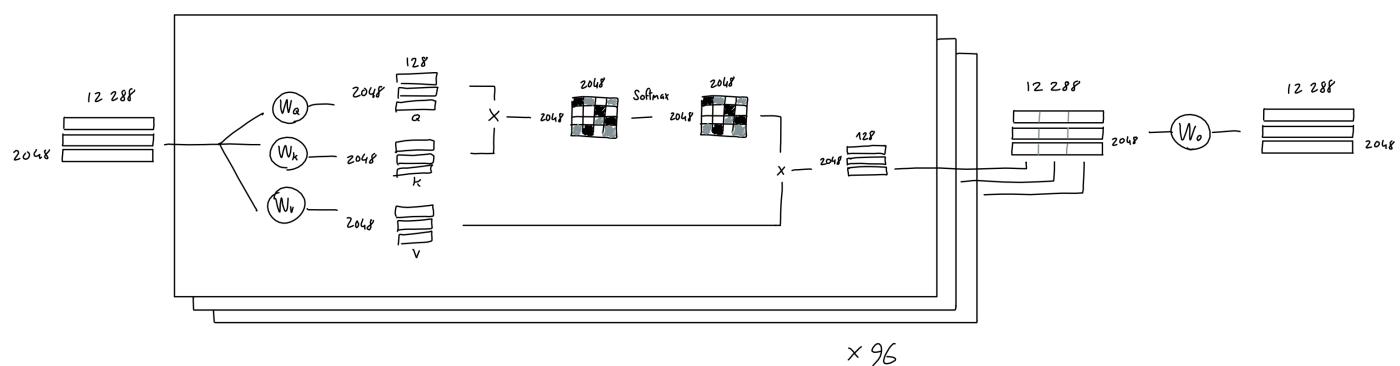
掩码注意力会形成一个上三角矩阵，保证根据当前序列预测下一个token时，不会与后续token产生信息交互



多头注意力机制

现在，在作者提出的 GPT 模型中，他们使用了多头注意力机制。这意味着上述过程会重复多次（在 GPT-3 中为 96 次），每次都使用不同的学习查询、键、值投影权重。

每个注意力头（单个 2048×128 矩阵）的结果被连接在一起，得到一个 2048×12288 矩阵，然后将其与线性投影（不会改变矩阵形状）相乘，以获得良好的测量效果。

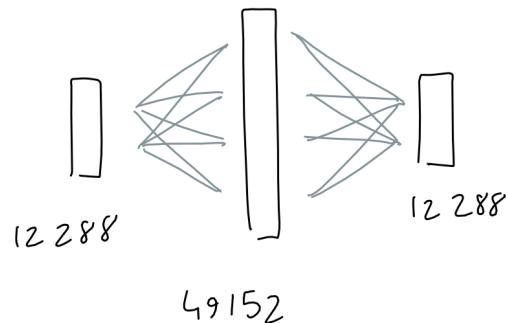
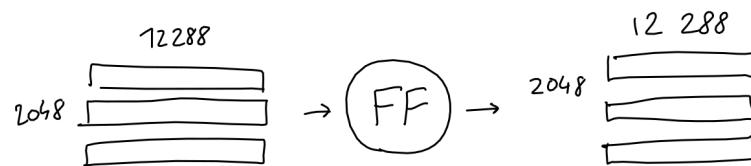


“吹毛求疵”：上图表示每个头部都有单独的权重矩阵。然而，在实践中，注意力模型实现可能会对所有头部使用一个大的组合权重张量，进行一次矩阵乘法，然后将其拆分成每个头部的 a 、 k 、 v 矩阵。不用担心：理论上，它也不会影响模型输出，因为代数运算是相同的。

前馈

钉子图标 前馈模块是一个老式的多层感知器，具有 1 个隐藏层。获取输入，乘以学习到的权重，添加学习到的偏差，重复此操作，即可获得结果。

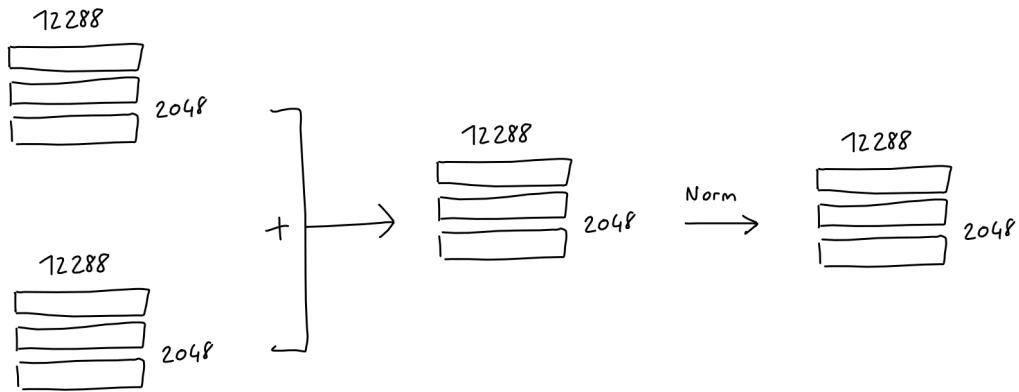
这里，输入和输出形状相同 (2048×12288)，但隐藏层的大小为 4×12288 。



需要明确的是：我将此操作画成一个圆圈，但与架构中的其他学习投影（嵌入、查询/键/值投影）不同，这个“圆圈”实际上由两个连续的投影（学习权重矩阵与输入相乘）组成，每个投影之后都添加了学习偏差，最后添加一个 $ReLU$ 。

Add&Norm

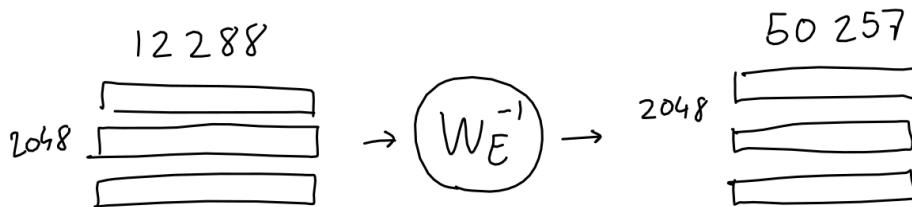
钉子图标 在多头注意力和前馈块之后，块的输入被添加到其输出，结果被归一化。这在深度学习模型中很常见（自 ResNet 以来）。



解码

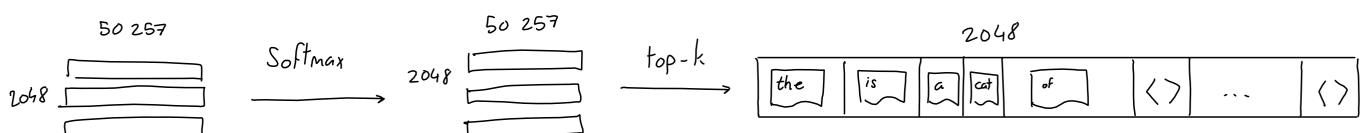
📌 经过 GPT-3 注意力/神经网络机制的所有 96 层后，输入被处理成一个 2048×12288 的矩阵。对于序列中 2048 个输出位置中的每一个，这个矩阵应该包含一个 12288 向量，其中包含有关应该出现哪个单词的信息。但我们如何提取这些信息呢？

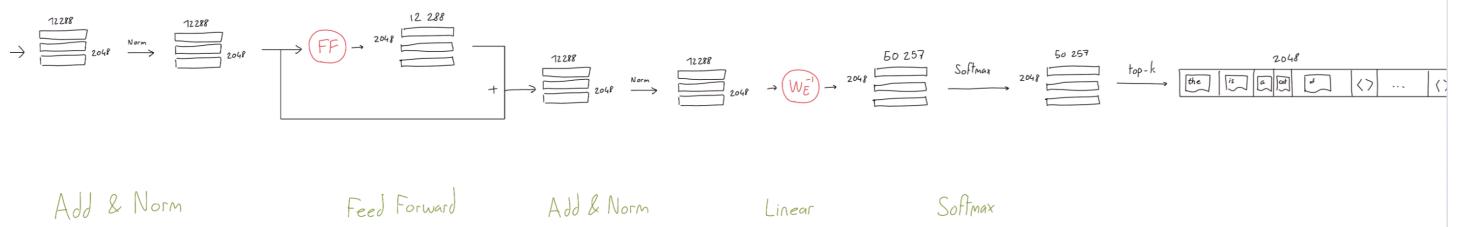
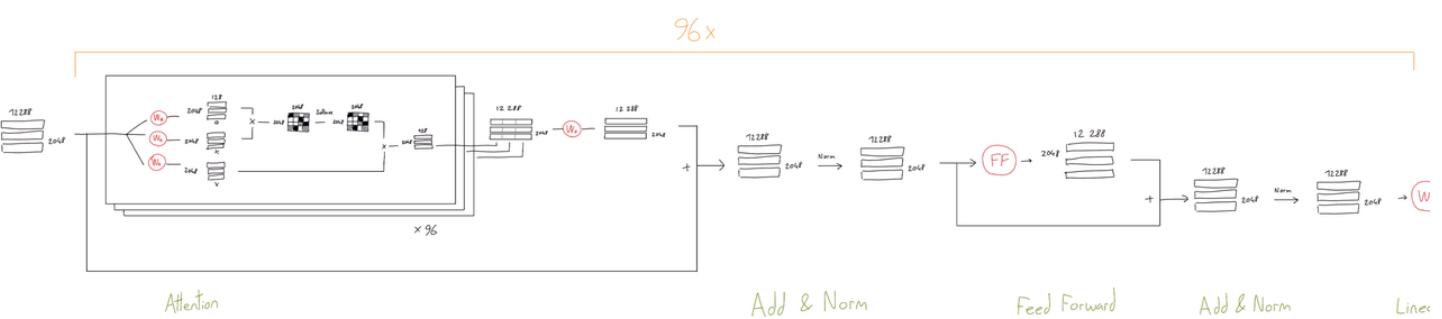
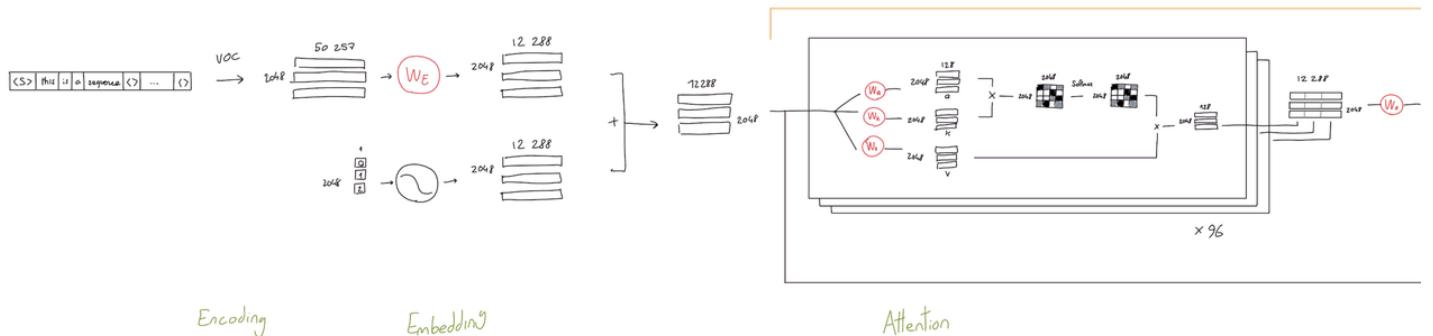
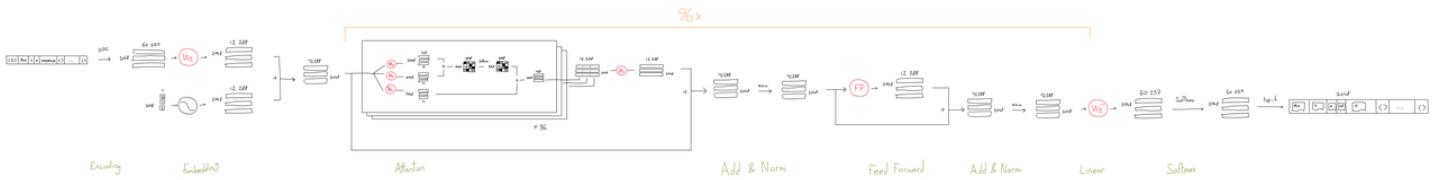
如果您还记得在“嵌入”部分中，我们学习了一种映射，它将给定的单词（一个单词的独热编码）转换为 12288 个向量嵌入。事实证明，我们可以反转此映射，将输出的 12288 个向量嵌入转换回 50257 个单词编码。这个想法是，如果我们花费所有这些精力来学习从单词到数字的良好映射，我们不妨重复使用它！



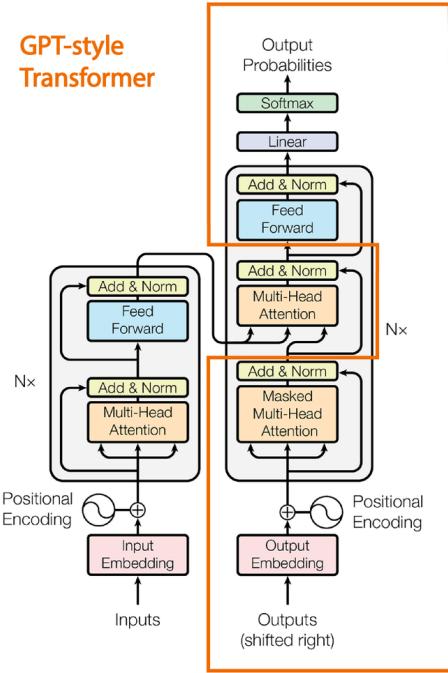
📌 当然，这样做不会像开始时那样给我们 1 和 0，但这是一件好事：经过快速的 softmax，我们可以将结果值视为每个单词的概率。

此外，GPT 论文中提到了参数 top-k，它将输出中可能采样的单词数量限制为 k 个最有可能的预测单词。例如，如果 top-k 参数为 1，我们总是选择最有可能的单词。





Add & Norm Feed Forward Add & Norm Linear Softmax

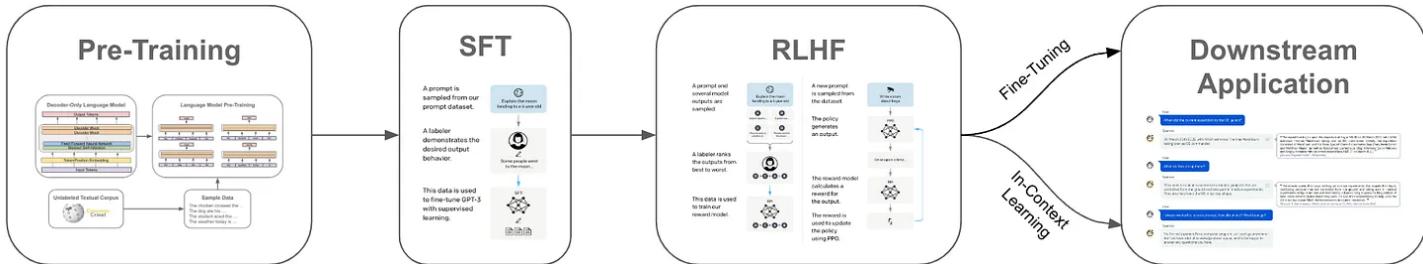


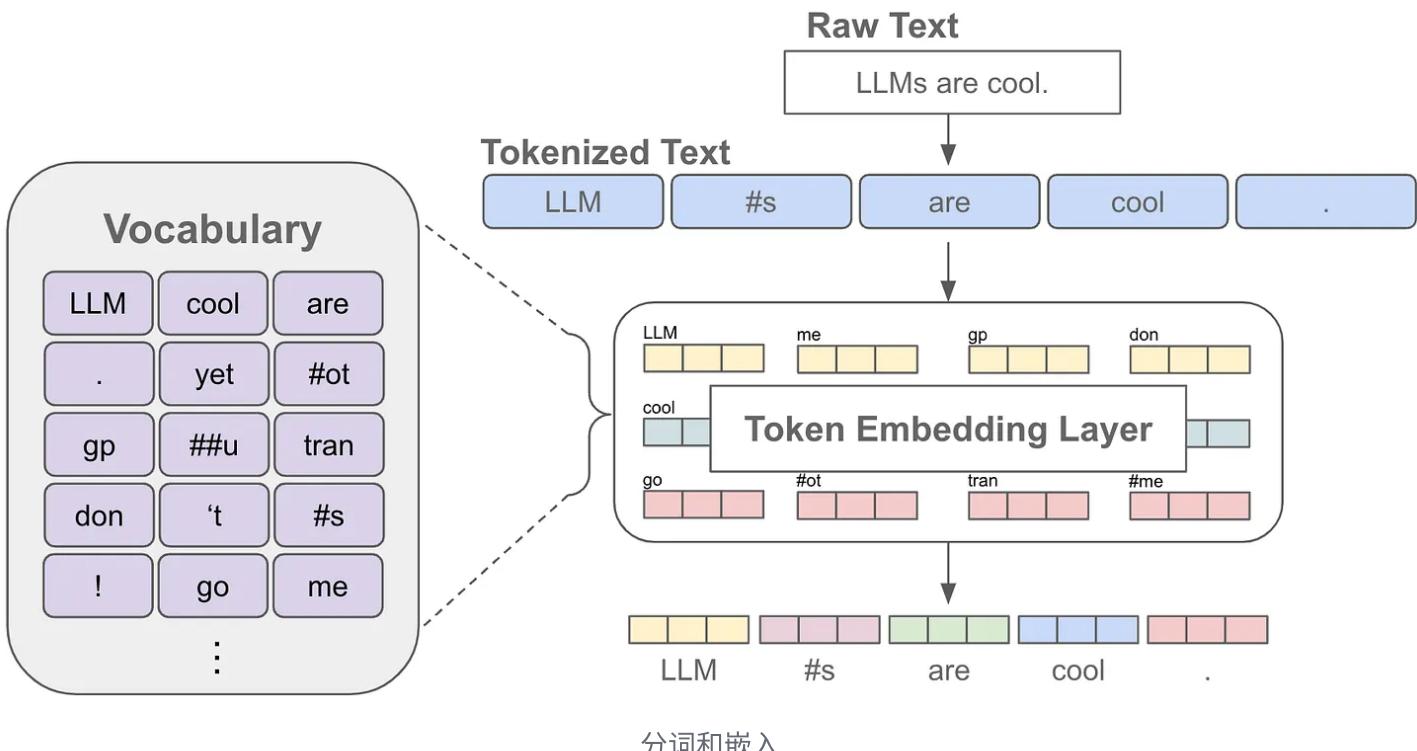
大语言模型预训练

大语言模型的训练分为几个步骤：

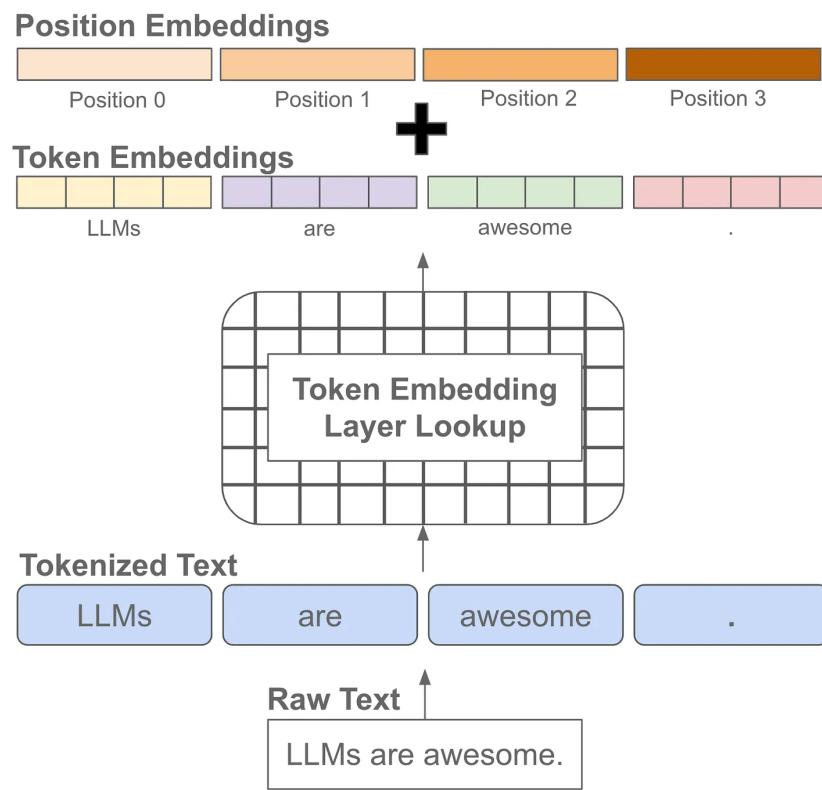
预训练（Pre-train）、有监督对齐（SFT）、强化学习对齐（RLHF）、下游微调（Fine-tune）

第一步（也是计算成本最高的一步）是**预训练**，我们将在本概述中重点介绍这一步骤。在预训练期间，我们会获得大量未标记的文本语料库，并通过以下方式训练模型：*i*) 从数据集中抽取一些文本；*ii*) 训练模型预测下一个单词。这是一个**自监督**目标，因为不需要任何标签。相反，下一个标记的基本事实已经存在于语料库本身中——**监督来源是隐式的**。这样的训练目标称为**下一个标记预测**，或**标准语言建模目标**。



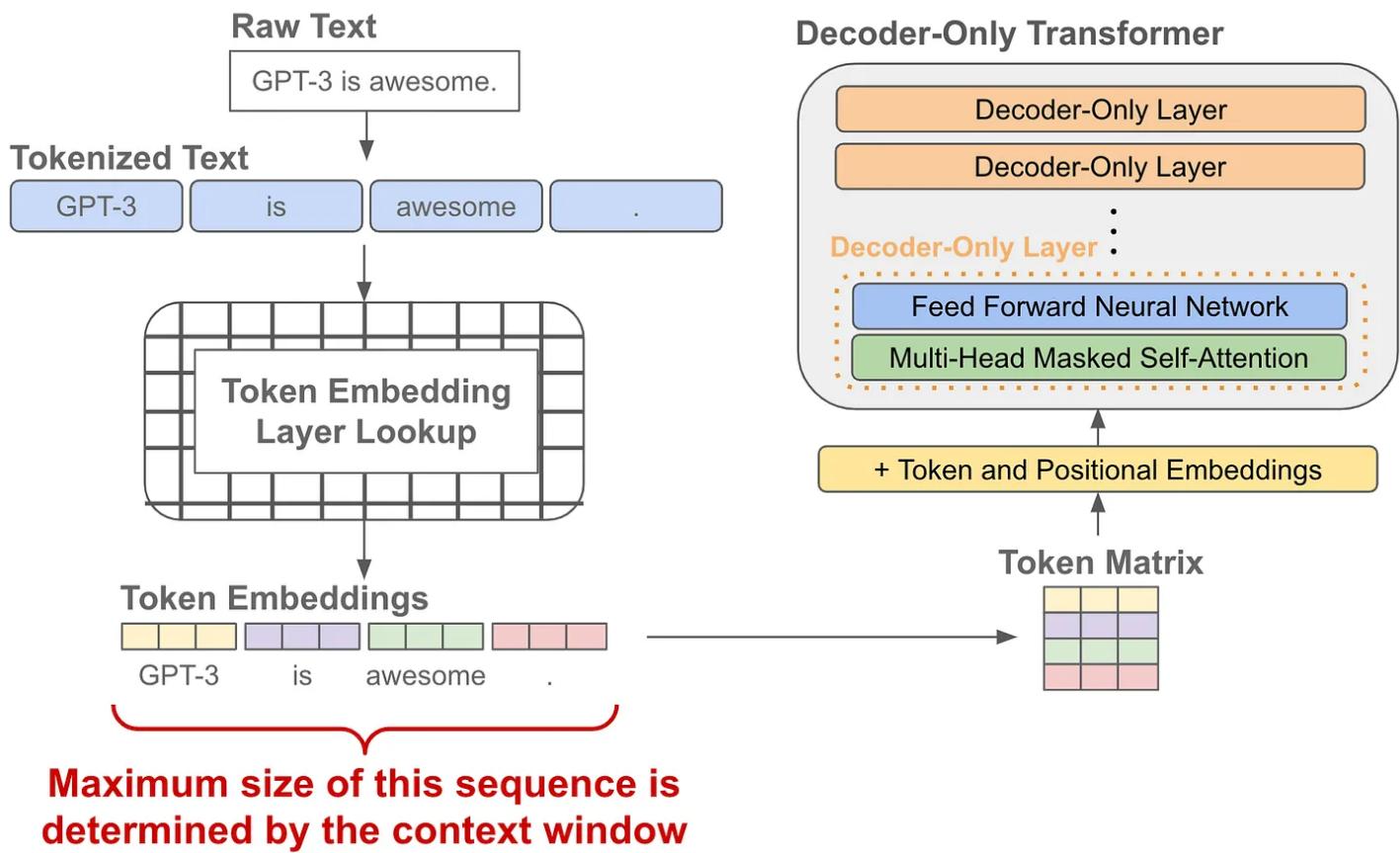


分词和嵌入



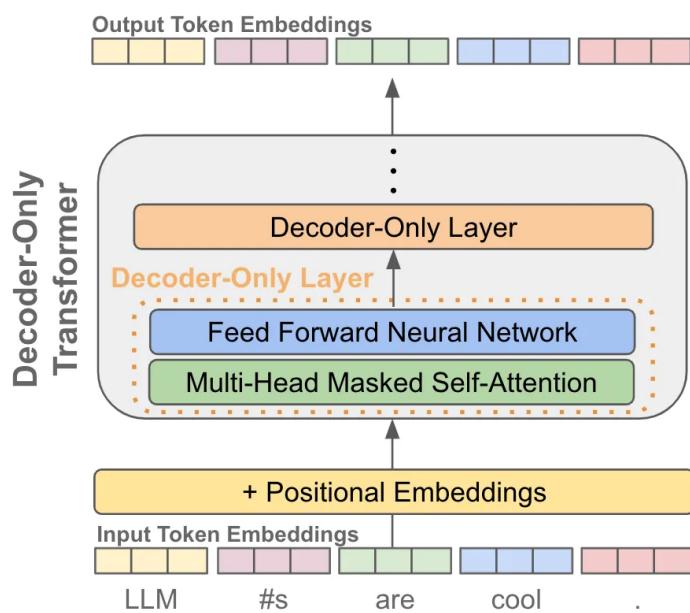
加入位置信息，目前广泛采用的是RoPE位置编码，因为其有良好的外推性

📌 **上下文窗口。** 语言模型使用特定大小的标记序列进行预训练，该大小称为上下文窗口的大小或上下文长度。此大小（通常在 1K 到 8K 个标记范围内（尽管某些模型要大得多！））通常根据硬件和内存限制进行选择(3)。鉴于我们只学习此长度输入的位置嵌入，上下文窗口限制了 LLM 可以处理的输入数据量。然而，后续训练的长文本外推技术（如PI插值、NTK-Aware）可以推断出比训练期间看到的输入更长的输入。

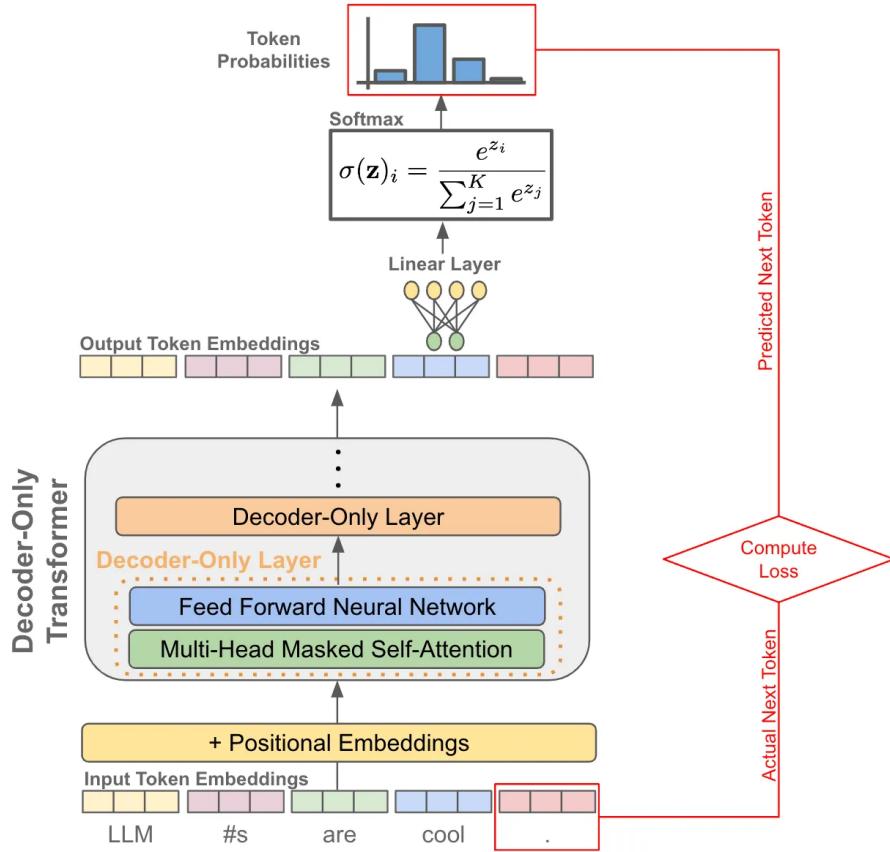


预训练过程

📌 预测下一个 token。有了 token 嵌入（带有位置嵌入）之后，我们将这些向量传递到仅解码器转换器中，该转换器为每个 token 嵌入生成相应的输出向量



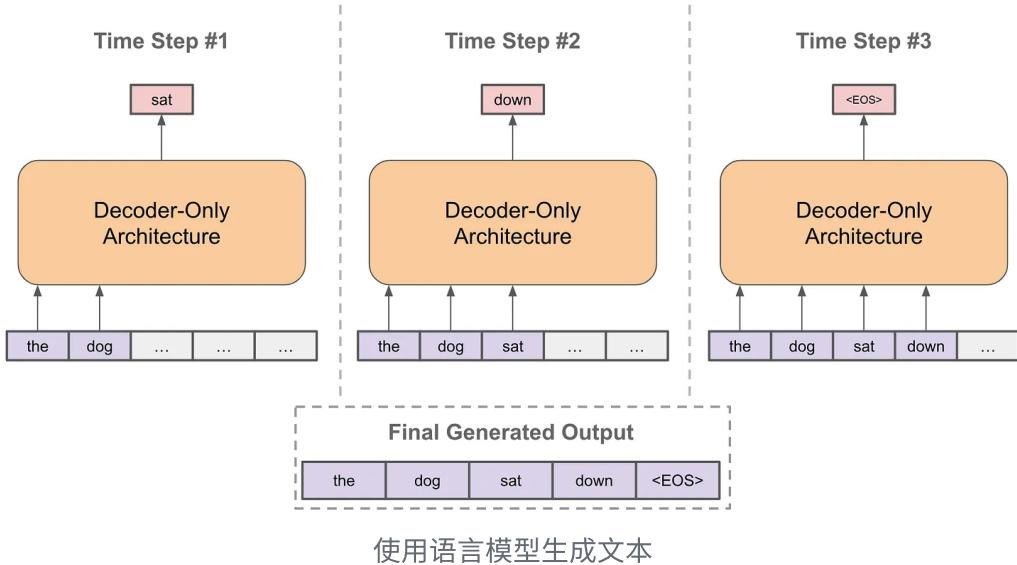
📌 给定每个token的输出向量，我们可以通过以下方式执行下一个token预测：*i*)获取token的输出向量；*ii*)使用此向量预测序列中的下一个token。



如上所示，通过将token的输出向量作为输入传递到线性层来预测下一个token，该线性层输出与词汇表大小相同的向量。应用 softmax 变换后，会形成token词汇表的概率分布，我们可以*i*)在推理期间从该分布中抽取下一个token，或者*ii*)在预训练期间训练模型以最大化下一个token正确的概率。

预测整个序列中的token。在预训练期间，我们不会只预测单个下一个标记。相反，我们会对序列中的每个token执行下一个标记预测，并汇总所有token的损失。由于使用了因果自注意力，每个输出标记向量仅考虑当前token和序列中位于其之前的token。因此，可以使用仅解码器转换器的单次前向传递对整个序列执行下一个token预测，因为每个标记都不了解位于其之后的token。

大语言模型推理



使用语言模型生成文本

现在，我们了解了如何预训练语言模型，但在进行推理时也会使用下一个标记预测！下一个标记预测是训练和使用 *LLM* 的所有方面的基础。从初始（可能为空）输入序列或前缀开始，语言模型通过遵循自回归下一个标记预测过程（参见上文）生成文本，步骤如下：

1. 预测下一个 token
2. 将预测的 token 添加到当前输入序列中
3. 重复

采样策略

对于垃圾邮件分类任务，输出概率最高的值是可以的。如果电子邮件有 90% 的概率是垃圾邮件，则将其归类为垃圾邮件。但是，对于语言模型，总是选择最有可能的token（贪婪采样）会产生无聊的输出。想象一下，无论你问什么问题，模型总是用最常见的单词来回答。

选择下一个标记。在上一节中，我们已经了解了如何创建标记的概率分布。但是，我们实际上如何从该分布中选择下一个token？通常，我们只是从该分布中抽样下一个token。但是，存在许多采样策略，这些策略通过修改标记的概率分布来对这种方法进行细微的改变。常见的采样策略如下：

- 贪婪搜索（每次只采样概率最高的token）
- 随机搜索
 - 温度
 - 核采样
 - Top-K 采样

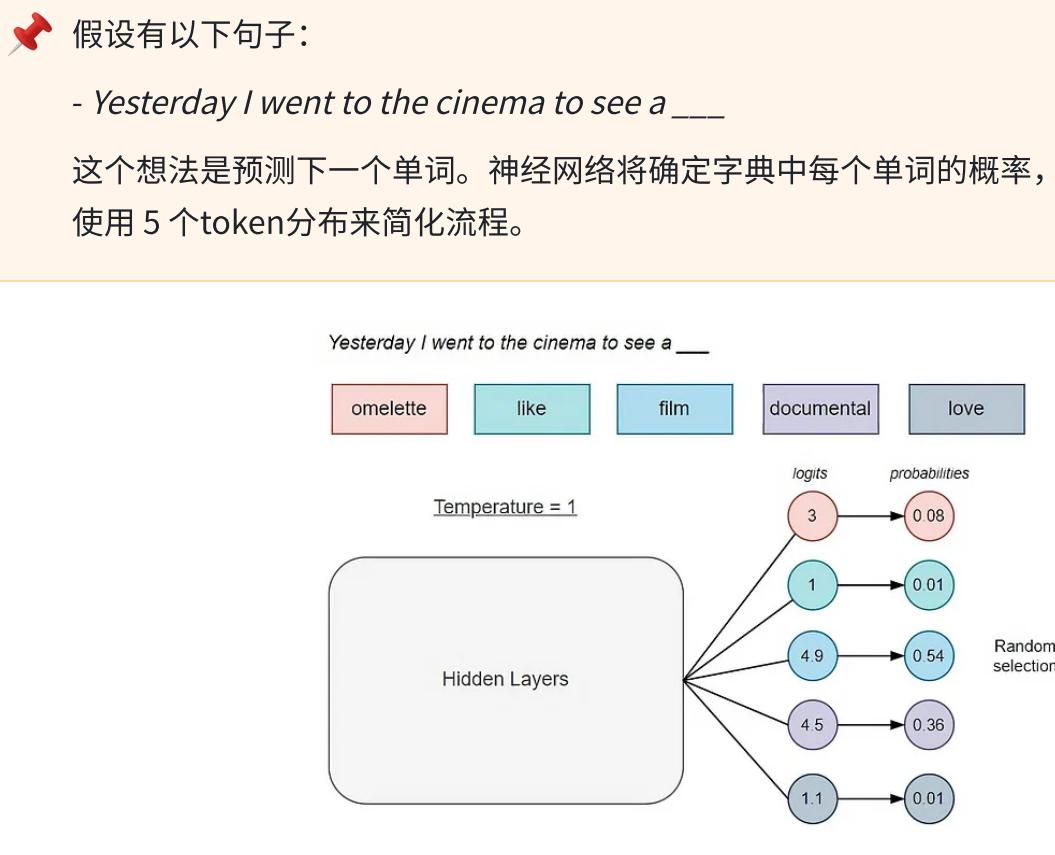
温度

温度是微调 GPT-3 等大型语言模型 (LLM) 输出的关键超参数。它在**控制生成文本的随机性和创造性方面**起着至关重要的作用。这些大型语言模型的输出是单词出现概率的函数。换句话说，要生成一个单词，词典中的每个单词都与一个概率相关联，并在此基础上确定如何进行。**这个超参数的主要思想是调整这些概率以强制随机性或确定性。**

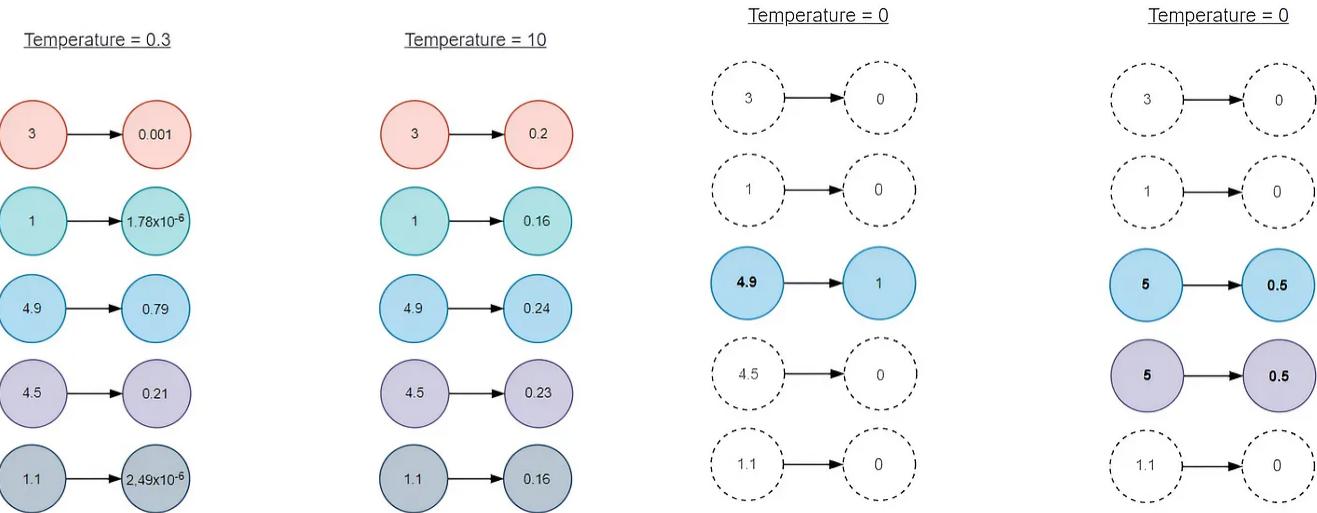
字典中每个单词的概率生成是在最后一层通过应用 *softmax* 作为激活函数完成的。回想一下，*softmax* 作用于 logits 以将它们转换为概率。这正是温度发挥作用的地方。

$p(x_i) = \frac{e^{x_i}}{\sum_{j=1}^V e^{x_j}}$	$p(x_i) = \frac{e^{\frac{x_i}{T}}}{\sum_{j=1}^V e^{\frac{x_j}{T}}}$
Softmax (temperature = 1)	Softmax with temperature

从上面的公式可以看出，softmax对每个 logit (x)求幂，然后将每个求幂的值除以所有求幂的值之和。此步骤确保输出是概率分布，这意味着值介于 0 和 1 之间，并且总和为 1。温度超参数是定义为“ T ”的值，应用于每个 logit，使低温使概率更加偏向极端。



从生成的概率来看，随机性将负责确定后面的单词。如上所示，在上面的示例中，应用了普通 $softmax$ （温度 = 1）。下面，我们将看到温度值为 0.3、10 和 0 时的结果。



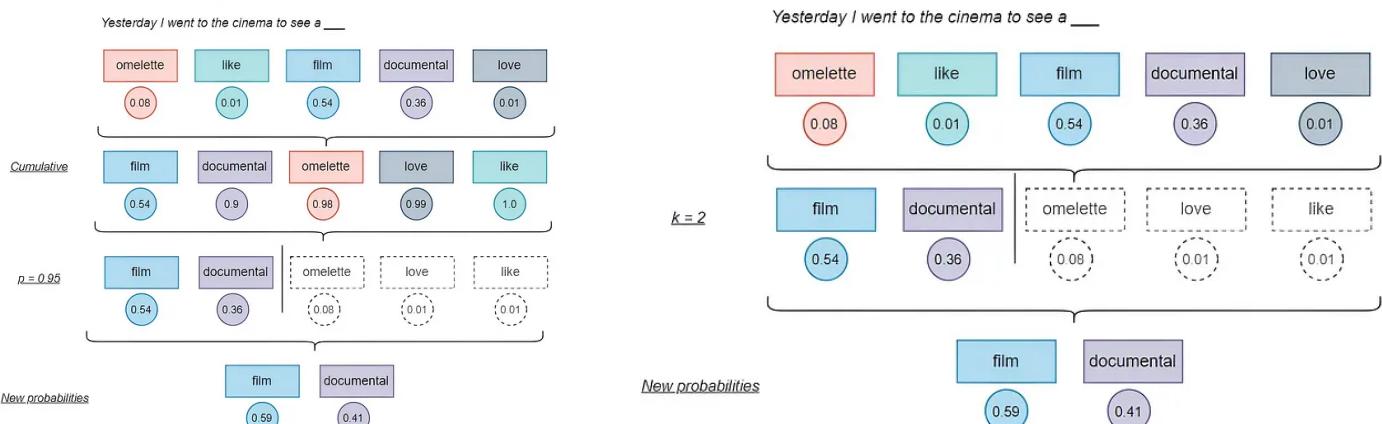
随着温度值接近 0，概率会进一步增加，从而使得选择的可能性更大。相反，当温度变得更高时，概率会降低，从而使得更多意想不到的单词更有可能被选中。

因此，当温度值等于 0 时，它就变成一个确定性解。但是，如果有两个词具有相同的 logit 值，因此概率也相同，则温度为 0 将使这两个词被选中的可能性相等，并且加起来为 1。

📌 Top_p和Top_k

- 1. Top_p (核采样) :** 它从概率分布中选择最可能的token，考虑累积概率，直到达到预定的阈值 “ p ”。这限制了选择的数量，并有助于避免过于多样化或无意义的输出。
- 2. Top_k (Top-k 采样) :** 它根据概率将 token 的选择限制为 “ k ” 个最可能的选项。这可以防止模型考虑概率非常低的 token，从而使输出更加集中和连贯。

Top_p和Top_k的核心思想是缩小采样空间，避免采样到过于多样化和无意义的token



在上图中，我们可以看到top_p的工作原理。您可以看到它如何经历排序过程，然后是概率的累积。利用这些，建立一个阈值并选择这些单词。最后，重新计算概率。假设 “ p ” 值小于最大概率，则选择最可能的单词。

Top_k的工作原理相同，但仅将参数 “ k ” 视为最可能的单词。与top_p过程一样，对概率进行排序，但在这种情况下，无需累积。排序后，设置 “ k ” 的阈值，最后重新计算概率。

- 现代大模型常用采样组合是Temperature+Top_p采样