语言模型01: 从Bigram、n-gram到GPT——它们如何回应语言建模的核心问题?

Language Model 01: From Bigram, n-gram to GPT — How do they respond to the core issues of language modeling?

璟明 Jingming

2025-03-28 05:08:32 已编辑 美国

2025-03-28 05:08:32 Edited United States

前言 Preface

在这篇文章中[对恋爱建模：马尔可夫决策过程、Bellman方程、价值迭代和策略迭代](www.douban.com/note/871389966/)，我们已经知道了Markov模型和Markov性——即，未来状态只依赖于当前状态，而与过去的历史状态无关。那么，自然语言处理（Natural Language Process, NLP）的问题能否用Markov建模？会面临什么麻烦？更好的方法是什么？

In this article [Modeling Love: Markov Decision Process, Bellman Equation, Value Iteration and Policy Iteration](www.douban.com/note/871389966/), we have learned about the Markov model and Markov property, that is, the future state depends only on the current state and has nothing to do with the past historical state. So, can the problem of Natural Language Processing (NLP) be modeled using Markov? What troubles will it encounter? What is a better way?

从这些问题出发，文章将谈到几种语言模型：Bigram、n-gram和GPT，并着重介绍GPT所基于的Transformer架构，以及其中的注意力机制（Attention mechanism），还有一些别的细节。

Based on these questions, this article will discuss several language models: Bigram, n-gram and GPT, and focus on the Transformer architecture on which GPT is based, as well as the attention mechanism and some other details.

阅读这篇文章，patience is all you need!

Read this article, patience is all you need!

一，什么是语言建模 1. What is Language Modeling?

语言建模是说建立一个模型，用于估计一段文本序列 **w_1,w_2,...,w_n** 的联合概率 **P(w_1,w_2,...,w_n)** 。（太抽象了！什么鬼！）

Language modeling is about building a model to estimate the joint probability **P(w_1,w_2,...,w_n**) of a text sequence w_1,w_2,...,w_n. (Too abstract! What the hell!)

让我们来看一个例句： Let's look at an example sentence:

“鱼，我所欲也；熊掌，亦我所欲也。二者不可得兼，舍鱼而取熊掌者也。”

"Fish is what I want; bear's paw is also what I want. I cannot have both, so I will give up fish and take bear's paw."

换句话说，就是要建立一个 **模型** ，它可以算出这些 **字符同时出现（** 也就是形成这个句子/文本序列）的概率。

In other words, to establish a **Model** , which can calculate these **Characters appear simultaneously (** That is, the probability of forming this sentence/text sequence).

因为有了模型，我们不仅能算出上面这个句子的概率，也能算出别的：

Because we have the model, we can not only calculate the probability of the sentence above, but also others:

“青春，我所欲也；学习，亦我所欲也。二者不可得兼，舍青春而取学习者也。”

"Youth is what I want; learning is also what I want. I cannot have both, so I will give up youth and choose learning."

如果我们的模型可靠，那么相比上面的原文，这个句子的概率显然会低很多，因为它是我瞎说的。

If our model is reliable, then the probability of this sentence will obviously be much lower than the original text above, because it is just something I made up.

还有别的： And others:

“ 粉本是刷我领匠强”

" The fan is to brush my leader's strength"

这个句子根本不通，如果我们的模型可靠，它的概率应该非常接近零。

This sentence doesn't make sense at all, and if our model is reliable, its probability should be very close to zero.

好的，现在我们明白语言建模是什么意思了。 Ok, now we understand what language modeling means.

让我们回到 **P(w_1,w_2,...,w_n)** ，根据链式法则它可以继续分解：

Let's go back to **P(w_1,w_2,...,w_n)** , which can be further decomposed according to the chain rule:

P(w_1,w_2,...,w_n) = P(w_1) · P(w_2|w_1) · P(w_3|w_1,w_2) · ... · P(w_n|w_1,...,w_n−1)

因此，只要知道这些条件概率 **P(w_n|w_1,...,w_n−1)** ，就能计算整个式子了！

Therefore, as long as we know these conditional probabilities **P(w_n|w_1,...,w_n−1)** , we can calculate the entire formula!

好的，如果整篇文章只能记住一句话，那就是这一句： OK, if you can only remember one sentence from this entire article, it should be this:

不同的语言建模方式，本质上就是以不同的方式回答同一个问题： **P(w_n|w_1,...,w_n−1)** 该怎么算。我们后面称这个问题为**核心问题**。

Different language modeling methods essentially answer the same question in different ways: **P(w_n|w_1,...,w_n−1)** How to calculate. We will call this problem **the core problem** later.

二，用Markov做语言建模：Bigram 2. Language Modeling with Markov: Bigram

未来状态只依赖于当前状态，而与过去的历史状态无关——这是我们已经熟悉的Markov性。

The future state depends only on the current state and has nothing to do with the past historical state - this is the Markov property we are already familiar with.

如果直接用Markov性建模，用1个字符定义状态，就会得到**Bigram模型（其实就是2-gram）**：

If we use Markov modeling directly and define the state with one character, we will get **a Bigram model (actually 2-gram)** :

P(w_n|w_1,...,w_n−1) = P(w_n|w_n−1)

即，下一个字符（未来状态）出现的概率，仅仅依赖于当前字符（当前状态）出现的概率，而与过去的历史字符（历史状态）无关。

That is, the probability of the next character (future state) appearing depends only on the probability of the current character (current state) appearing, and has nothing to do with the past historical characters (historical states).

回到最开始的例句：“鱼，我所欲也；熊掌，亦我所欲也。二者不可得兼，舍鱼而取熊掌者也。”

Let’s go back to the original example sentence: “Fish is what I want; bear’s paw is also what I want. I cannot have both, so I will give up fish and take bear’s paw.”

在Bigram模型中， 熊->掌，“掌”的出现只和“熊”有关，而与更早的字符无关。

In the Bigram model, in bear->pal, the appearance of “palm” is only related to “bear” and has nothing to do with earlier characters.

于是，我们只需要统计语料库中字符对出现的频率（例如“熊掌”这个字符对出现了多少次，“熊”出现了多少次），就能构建概率表（ P(掌|熊) = Count("熊掌") / Count(“熊”) ） 来回答核心问题。

Therefore, we only need to count the frequency of occurrence of character pairs in the corpus (for example, how many times does the character pair "熊掌" appear, and how many times does "熊" appear), and we can construct a probability table (P(掌|熊) = Count("熊掌") / Count("熊")) to answer the core question.

我们再做一点**空间复杂度**计算，假设词汇表有V个不同的词：

Let's do some more **space complexity** calculations, assuming that the vocabulary has V different words:

1. 状态表是O(V)。（V种词汇，即V种可能性，填入1个状态中 = V^1 ）

1. The state table is O(V). (V words, or V possibilities, are filled into 1 state = V^1)

2. 概率表是O(V)。（每种状态对应除自己以外，剩下所有状态 = V^2 ）。

2. The probability table is O(V). (Each state corresponds to all states except itself = V^2).

从上下文的角度，这看起来是很糟糕的模型，对吧？现实情况就像高考的阅读理解，肯定需要依赖更多的上下文。

From a contextual perspective, this seems like a pretty bad model, right? The reality is like the reading comprehension test in the college entrance examination, which definitely needs to rely on more context.

三，用Markov做语言建模：n-gram 3. Language Modeling with Markov: n-gram

为了优化Bigram，我们自然想到：有没有办法让下一个字符不仅依赖于当前字符，也依赖于更多的历史字符？但这样似乎不符合Markov性了...怎么办？

In order to optimize Bigram, we naturally think: Is there a way to make the next character depend not only on the current character, but also on more historical characters? But this does not seem to conform to the Markov property... What should we do?

我们可以通过重新定义“状态”来解决这个问题。 We can fix this by redefining what we mean by "state".

在Bigram中，预测下一个字符只依赖于前一个字符，比如 “熊”、“掌”，预测“掌”只依赖于 “熊” 。

In Bigram, predicting the next character only depends on the previous character, such as "熊" and "掌", predicting "掌" only depends on "熊".

但如果把“状态”定义为前几个字符的组合呢？ But what if we define "state" as the combination of the first few characters?

在**n-gram**模型中，我们用 n-1 个字符定义状态。假设下一个字符 w_n 的出现概率依赖于前 n−1 个字符，而不是仅仅一个字符：

In the **n-gram** model, we use n-1 characters to define the state. Assume that the probability of the next character w_n depends on the previous n−1 characters, not just one character:

P(w_n|w_1,...,w_n−1) = P(w_n|w_(n−(n−1)),...,w_n−1)

（Tips：细心的同学会发现，这里的P不是Markov的状态转移概率了。条件是当前状态，但结果不是下一个状态。不用困惑，下一状态中，起始词为W_n的状态转移概率相加就可以得到这个结果。）

(Tips: Careful students will find that P here is not Markov's state transition probability. The condition is the current state, but the result is not the next state. Don't be confused, in the next state, the state transition probability with the starting word W_n is added to get this result.)

回到例句：“鱼，我所欲也；熊掌，亦我所欲也。二者不可得兼，舍鱼而取熊掌者也。”

Let’s go back to the example sentence: “Fish is what I want; bear’s paw is also what I want. I cannot have both, so I will give up fish and take bear’s paw.”

如果是 3-gram 模型，下一个字符的出现概率依赖于前 3-1=2 个字符——P(掌|也,熊)

If it is a 3-gram model, the probability of the next character depends on the previous 3-1=2 characters - P(掌|也,熊)

如果是 4-gram 模型， 下一个字符的出现概率依赖于前 4-1=3 个字符—— P(掌|欲,也,熊)

If it is a 4-gram model, the probability of the next character depends on the previous 4-1=3 characters - P(掌|欲,也,熊)

同样，只需要统计语料库中字符组出现的频率，就能构建概率表来回答核心问题。

Similarly, we can construct a probability table to answer the core question simply by counting the frequency of character groups in the corpus.

同样做**空间复杂度**计算，假设词汇表有V个不同的词：

We also calculate **the space complexity** , assuming that the vocabulary has V different words:

1. 状态表是O(V^n)。（V种词汇，即V种可能性，填入n-1个状态中 = V^(n-1) ）

1. The state table is O(V^n). (V kinds of words, that is, V possibilities, filled in n-1 states = V^(n-1))

2. 概率表是O(V^n)。(即V^(n-1) * V^(n-1) ~ V^n ）

2. The probability table is O(V^n). (i.e. V^(n-1) * V^(n-1) ~ V^n)

理论上，只要n-gram的n足够大，就可以理解足够长的上下文。

In theory, as long as the n of n-gram is large enough, a sufficiently long context can be understood.

问题是，当n增大时，状态表和概率表的空间复杂度 O(V^n) 指数增加，我们称之为 **状态爆炸** 。这导致n无法非常大，上下文就不能很长，概率就不够准，构建的语言模型就不够可靠。此外还有一些别的问题，这些麻烦让我们想寻找更优的方法。

The problem is that as n increases, the space complexity of the state table and probability table increases exponentially, which we call **State explosion** This means that n cannot be very large, the context cannot be very long, the probability is not accurate enough, and the constructed language model is not reliable enough. There are also some other problems, which make us want to find a better way.

四，用Transformer架构建模：GPT Fourth, modeling with Transformer architecture: GPT

要之，我们希望找到一种方法，它主要解决状态爆炸的问题，而且还有很多别的优点，这就是GPT。（如果你了解强化学习，此时脑海中应该有这样的类似：TD(0) —> Q* 类比 n-gram —> GPT）

In short, we hope to find a method that mainly solves the problem of state explosion and has many other advantages, which is GPT. (If you know reinforcement learning, you should have something like this in your mind: TD(0) —> Q* analogy n-gram —> GPT)

面对语言建模的核心问题： **P(w_n|w_1,...,w_n−1)** 该怎么算?

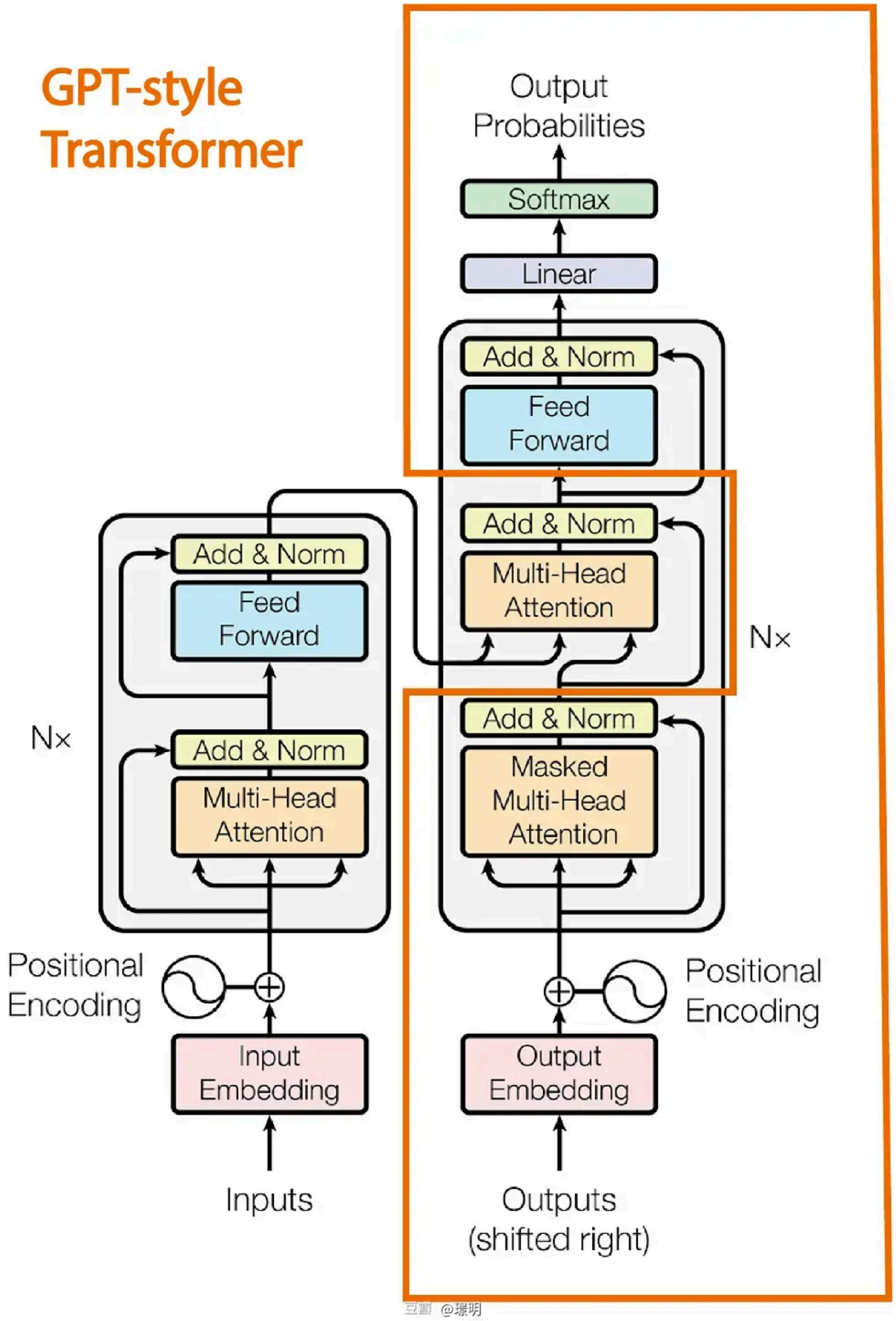
The core problem of language modeling is: How to calculate **P(w_n|w_1,...,w_n−1)** ?

GPT的回应是： 通过 **decoder-only的Transformer架构** ，用 **神经网络** 结合 **注意力机制** 来计算。

GPT's response: **Decoder-only Transformer architecture** ,use **Neural Networks** Combination **Attention Mechanism** to calculate.

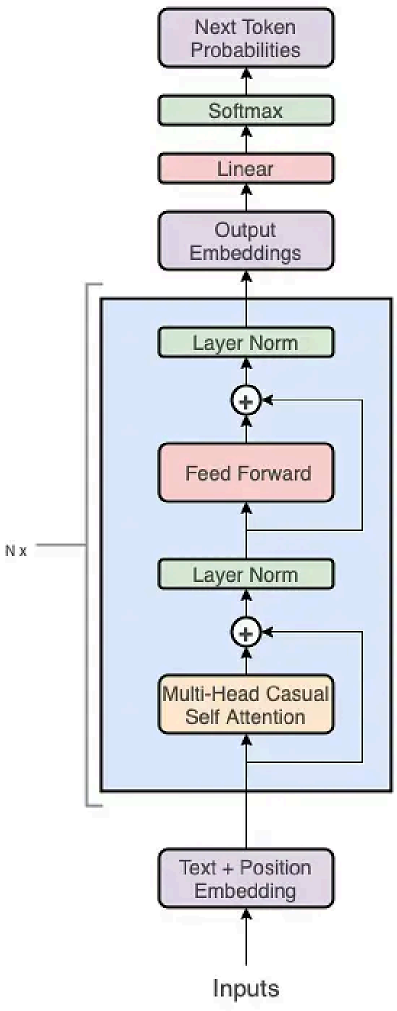
GPT（Generative Pre-trained Transformer）是基于decoder-only Transformer架构的语言模型。完整的Transformer架构包含两部分：左半边是encoder，右半边是decoder——GPT只取了右半部分。

GPT (Generative Pre-trained Transformer) is a language model based on the decoder-only Transformer architecture. The complete Transformer architecture consists of two parts: the left half is the encoder, and the right half is the decoder - GPT only takes the right half.



Transformer(encoder+decoder)

单独看decoder-only是这个样子： The decoder-only view looks like this:



decoder-only

下一篇文章，我们将从头开始，讲解如何搭建一个简化版本的GPT，希望这样可以帮助理解GPT到底做了什么。

In the next article, we will start from scratch and explain how to build a simplified version of GPT, hoping that this will help you understand what GPT actually does.

投诉 complaint

178人浏览 编辑 | 设置 | 删除

178 people viewed Edit | Settings | delete

回应 转发 赞 收藏

Reply Retweet Like Collection