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

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

Preface

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?

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.

Read this article, patience is all you need!

1. What is Language Modeling?

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!)

Let's look at an 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."

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

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

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

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.

And others:

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

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

Ok, now we understand what language modeling means.

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)

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

OK, if you can only remember one sentence from this entire article, it should be this:

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.

2. Language Modeling with Markov: Bigram

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.

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)

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).

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.”

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

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.

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

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

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

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.

3. Language Modeling with Markov: n-gram

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?

We can fix this by redefining what we mean by "state".

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

But what if we define "state" as the combination of the first few characters?

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: 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.)

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.”

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)

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)

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

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

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

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

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.

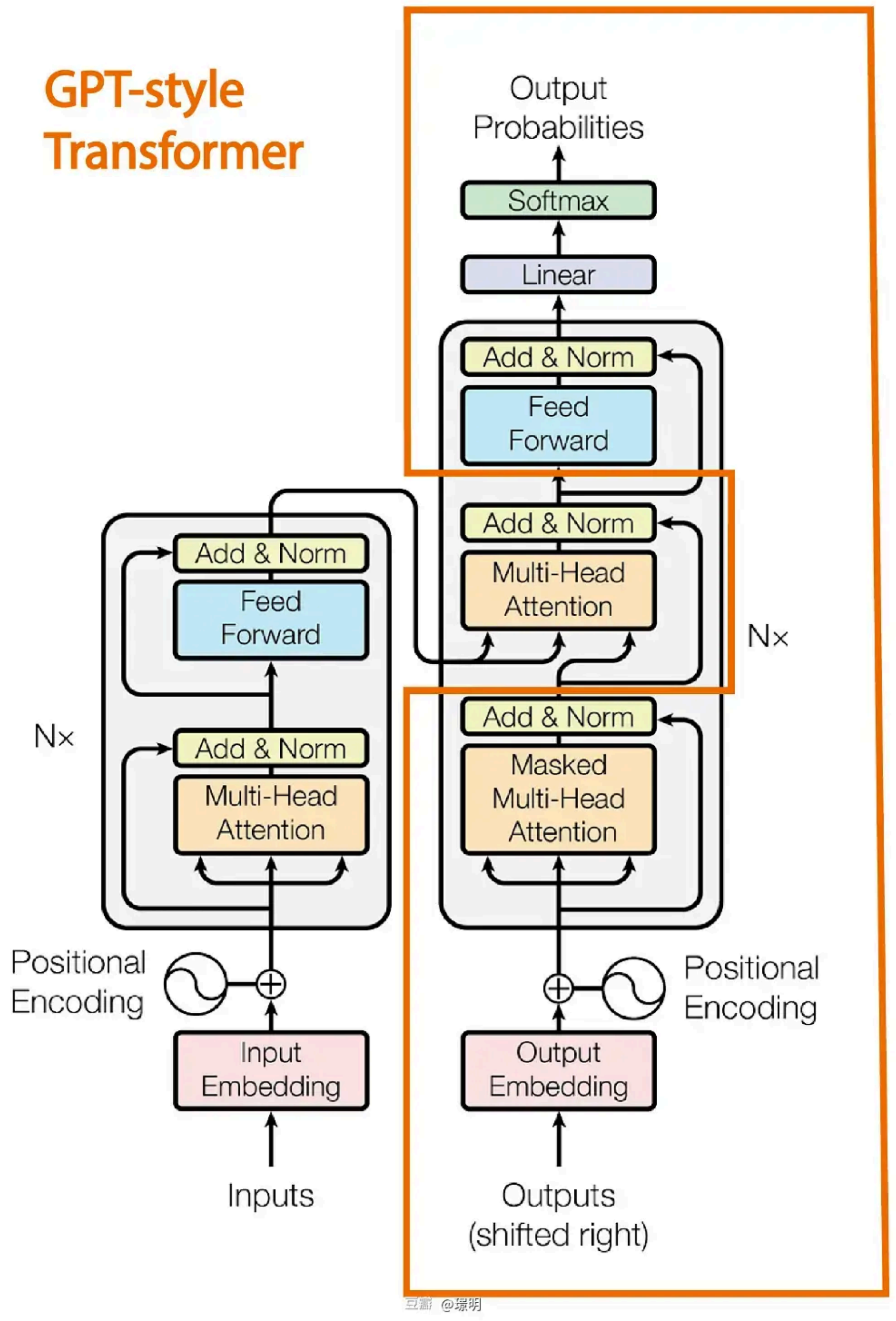
Fourth, modeling with Transformer architecture: GPT

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)

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

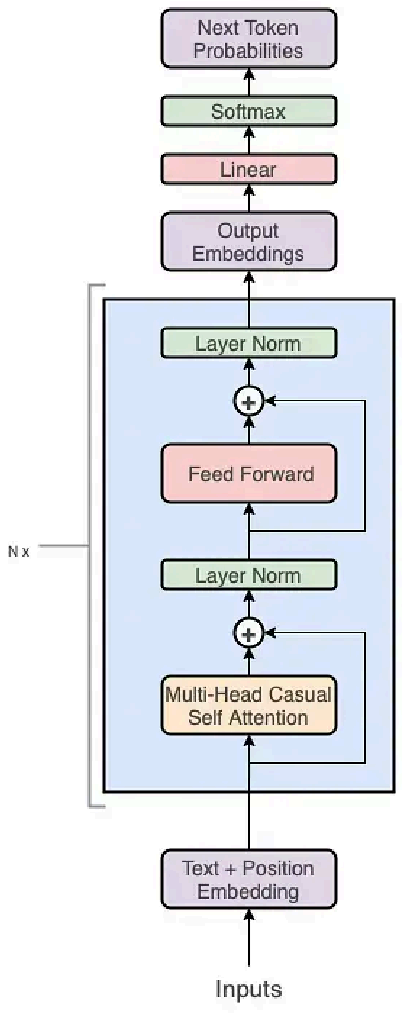
GPT's response: **Decoder-only Transformer architecture** ,use **Neural Networks** Combination **Attention Mechanism** to calculate.

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)

The decoder-only view looks like this:



decoder-only

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.

complaint

180 people viewed Edit | Settings | delete

Reply Retweet Like Collection