GPT is a model for sequences.

- It sees text as a sequence of tokens $x = x_0 x_1 x_2 \cdots x_N$
- Its training dataset is a collection of sequences $\{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$

---

The following is a classic Chinese poem from the Tang dynasty, translated into English.

The dawn light strikes the head of my bed
I see leaves

<table>
<thead>
<tr>
<th>TEXT</th>
<th>TOKEN IDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>The dawn light strikes the head of my bed</td>
<td>464, 1708, 318, 257, 6833, 3999, 21247, 422, 262, 18816, 30968, 11, 14251, 656, 3594, 13, 198, 198, 464, 17577, 1657, 8956, 262, 1182, 286, 616, 3996, 198, 40, 766, 5667, 220</td>
</tr>
</tbody>
</table>

---

GPT tokenizer: [https://platform.openai.com/tokenizer](https://platform.openai.com/tokenizer)
GPT is a probability model for sequences of tokens

- Let $X = X_0X_1X_2 \cdots X_N$ be a random sequence of tokens, of random length $N$
- What's a good probability model for $X$ and how do we fit it to a training dataset $\{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$?

- Once we have a trained probability model, we can use it for completion. We give it an input prompt $x = x_0x_1 \cdots x_m$ and it generates a sample from

$$ (X | X_0 = x_0, \ldots, X_m = x_m) $$

GPT playground: https://platform.openai.com/playground?mode=complete
§12. What’s a good probability model for sequences, and how can we fit it?
Bag-of-words text generation
Choose each word randomly, independently.
“us the incite o'er a land-damn are peace
inardinate take him worthy quick generals □”

Probability model: generate $X$ by producing random words until we produce □.
$X_1, X_2, ..., X_N, □$

$\Pr_X(x_1 x_2 \cdots x_n) = \Pr(x_1) \Pr(x_2) \times \cdots \times \Pr(x_n) \Pr(□)$

Let’s let $\Pr(w) = p_w$ where $p = [p_{w_1}, p_{w_2}, ..., p_{w_V}, p_{□}]$ is a probability vector with an entry for each word in the vocabulary.

We can learn the $p$ vector by maximizing the likelihood of the dataset $\{x^{(1)}, x^{(2)}, ..., x^{(n)}\}$. The mle is simple: $p_w =$ fraction of occurrences of word $w$ in the dataset
Markov model
Based on a graph of word-to-word transitions.

“to foreign princes lie in your blessing god who shall have the prince of rome □”

Probability model: generate $X$ by starting at □ and jumping from word to word until we hit □ again.

$$\square \rightarrow X_1 \rightarrow X_2 \rightarrow \cdots \rightarrow X_N \rightarrow \square$$

$$\Pr_X(x_1x_2 \cdots x_n) = \Pr(x_1|\square) \times \Pr(x_2|x_1) \times \cdots \times \Pr(x_n|x_{n-1}) \times \Pr(\square|x_n)$$

Let’s let $\Pr(w|v) = P_{vw}$ for some matrix $P$ that denotes the word-to-word transition probabilities. The maximum likelihood estimate for $P$ is easy to find, by simple counting of word pairs.
Andrei Markov (1856–1922)
Markov’s trigram model

“to be wind-shaken we will be glad to receive at once for the example of thousands □”

Probability model: Generate $X$ by starting with □□ and repeatedly generating the next word based on the preceding two, until we produce □.

$$
\Pr_X(x_1x_2 \cdots x_n) = \Pr(x_1|\square\square) \Pr(x_2|x_1) \Pr(x_3|x_1x_2) \times \cdots \times \Pr(x_n|x_{n-2}x_{n-1}) \Pr(\square|x_{n-1}x_n)
$$

Let’s let $\Pr(w|uv) = P_{(uv)w}$

It’s easy to estimate $P$, the (word,word)-to-word transition probabilities, by simple counting. (Before counting, preprocess the dataset by putting □□ at the start and □ at the end of every sentence.)
Different ways to write the trigram model:

A Markov Chain is a sequence in which each item is generated based only on the preceding item.

The trigram model is a Markov chain, whose items are word-pairs.

A deterministic bookkeeping function $f((x, y), z) = (y, z)$.

Random generation

Deterministic bookkeeping

$(x, y) \

X_{\text{new}}$
Can we get a better model by using more history?

**Trigram character-by-character model trained on Shakespeare:**
“on youghtlee for vingiond do my not whow’d no crehout withal deeper forand a but thave a doses?”

**5-gram character-by-character model trained on Shakespeare:**
“once is pleasurerly. though the the with them with comes in hand. good. give and she story tongue.”

**Deterministic bookkeeping function** $f((x, y), z) = (y, z)$

**Random generation** $X_{\text{new}}$

**QUESTION.** What are the advantages and disadvantages of a long history window?

**QUESTION.** Can we do better than using a fixed history window?
Let’s use a neural network to learn an appropriate history digest. This is more flexible than choosing a fixed history window.

RNN character-by-character model trained on Shakespeare
[due to Andrej Karpathy]:

“PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain’d into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.”

\[
\begin{align*}
  f_\theta(s_1, x_1) &= (p_1, s_{new}) \\
  f_\theta(s_2, x_2) &= (p_2, s_{new}) \\
  f_\theta(s_3, x_3) &= (p_3, s_{new}) \\
  \vdots \\
  f_\theta(s_N, x_N) &= (p_N, s_{new}) \\
\end{align*}
\]
A Recurrent Neural Network (RNN) is a probability model for generating a random sequence $X$.

$$X_i \sim \text{Cat}(p_i)$$

$$(s_{i+1}, p_{i+1}) = f_\theta(s_i, X_i)$$

We can train it in the usual way, by maximizing the log likelihood of our dataset. This is easy, because there’s a simple explicit formula for the likelihood of a datapoint:

$$\Pr_X(x_1, \ldots, x_n) = \Pr_{X_1}(x_1) \Pr_{X_2}(x_2|x_1) \times \cdots \times \Pr_{X_n}(x_n|x_1 \cdots x_{n-1}) \Pr_{X_{n+1}}(\square|x_1 \cdots x_n)$$

by the chain rule for probability

$$= [p_1]_{x_1} [p_2]_{x_2} \times \cdots \times [p_n]_{x_n} [p_{n+1}]_{\square}$$

where each $p_i$ is a function of $x_1 \cdots x_{i-1}$

```python
def loglik(xstr):
    res = 0
    s, x = 0, \square
    for x_next in xstr + "\square":
        s, p = f_\theta(s, x)
        res += log(p[x_next])
        x = x_next
    return res
```
The history of random sequence models

- **1913**: Markov chains
- **1966**: Hidden Markov models
- **1986**: RNN
- **1997**: LSTM
- **2017**: Transformers

Better models of the data
All trained by maximizing the log likelihood of the data

- Linguistic theories
- Non-probabilistic metrics
- Larger scale
- Prompt engineering
Transformer architecture

This is a probability model for a random sequence $X$.

Like the RNN, there’s a simple explicit formula for the log likelihood $\Pr_{X}(x)$, so it’s easy to train.

It’s more powerful than an RNN, because $f$ has access to the full sequence; it doesn’t have to squeeze history into a “history digest” at each step.

The following is a classic Chinese poem from the Tang dynasty, translated into English.
What does $f$ look like? How is it built out of differentiable functions?

Split the text into tokens $t_i \in \{1, \ldots, W\}$

Turn each token into a vector $e_i \in \mathbb{R}^d$
by looking up an embedding matrix $E \in \mathbb{R}^{W \times d}$

For each position $i \in \{1, \ldots, n\}$
create a position-embedding vector $t_i \in \mathbb{R}^d$

Let $x_i = e_i + t_i \in \mathbb{R}^d$
For each position $i \in \{1, \ldots, n\}$, let $q_i = Qx_i$, let $k_i = Kx_i$, let $v_i = Vx_i 
\in \mathbb{R}^e \quad \in \mathbb{R}^e \quad \in \mathbb{R}^d$

For each position $j \in \{1, \ldots, n\}$ we’ll produce an output vector $y_j \in \mathbb{R}^d$, as follows:

1. let $s_{ji} = q_j \cdot k_i$ and $a_{ji} = \text{softmax}(s_{ji}/\sqrt{e})$
2. let $y_j = \Sigma_i a_{ji} v_i$

From the final value $y_n$, compute $p = g(y_n) \in \mathbb{R}^W$ where $g$ is some straightforward neural network

Generate the next token by $X_{n+1} \sim \text{Cat}(p)$

$a_{ji}$ is the attention that we should give to input $x_i$ when computing output $y_j$
In practice, it’s useful to use several passes of the attention mechanism.