GPT is a model for sequences.

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

```
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
```

TEXT TOKEN IDS

```
[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]
```

TEXT TOKEN IDS

GPT tokenizer: https://platform.openai.com/tokenizer

GPT is a probability model for sequences of tokens

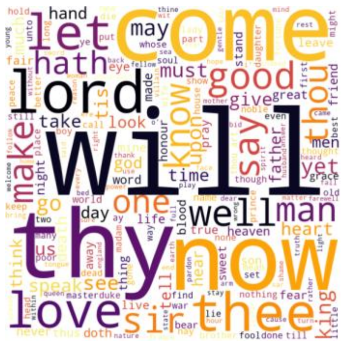
- Let $\underline{X} = X_0 X_1 X_2 \cdots X_N$ be a random sequence of tokens, of random length N
- What's a good probability model for $\underline{X}$
  and how do we fit it to a training dataset $\{\underline{x}^{(1)}, \underline{x}^{(2)}, \dots, \underline{x}^{(n)}\}$?

- Once we have a trained probability model, we can use it for completion.
  We give it an input prompt $\underline{x} = x_0 x_1 \cdots x_m$ and it generates a sample from

$$(\underline{X} \mid X_0 = x_0, \dots, X_m = x_m)$$

`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
incardinate take him worthy quick generals □”
```

end-of-sentence token

Probability model: generate $\underline{X}$ by producing random words until we produce $\square$.

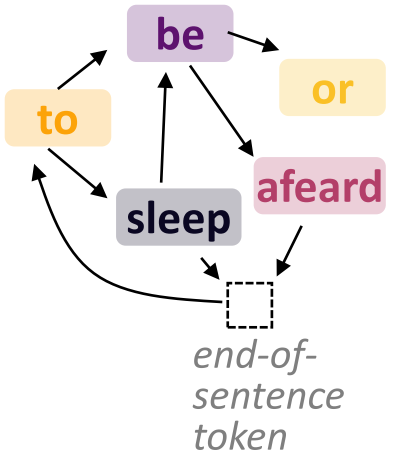
$X_1, X_2, \ldots, X_N, \square$

$$\Pr_{\underline{X}}(x_1 x_2 \cdots x_n) = \Pr(x_1)\Pr(x_2) \times \cdots \times \Pr(x_n)\Pr(\square)$$

Let’s let $\Pr(w) = p_w$ where $p = [p_{w_1}, p_{w_2}, \ldots, p_{w_V}, p_\square]$ 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 $\{\underline{x}^{(1)}, \underline{x}^{(2)}, \ldots, \underline{x}^{(n)}\}$.

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 $\underline{X}$ by starting at $\square$ and jumping from word to word until we hit $\square$ again.

$$\square \to X_1 \to X_2 \to \cdots \to X_N \to \square$$

$$\Pr_{\underline{X}}(x_1 x_2 \cdots x_n) = \Pr(x_1|\square) \times \Pr(x_2|x_1) \times \cdots \times \Pr(x_n|x_{n-1}) \times \Pr(\square|x_n)$$

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)

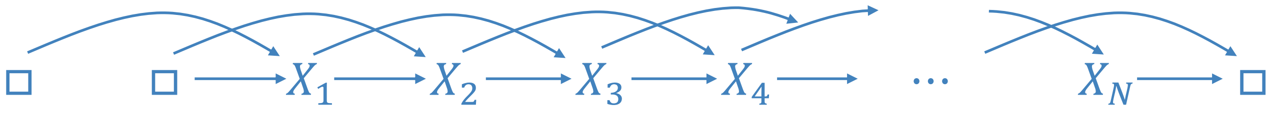
be contented to be what they
who is to be executed this
in him to be truly touched
took occasion to be quickly woo'd

Markov's trigram model

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

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

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

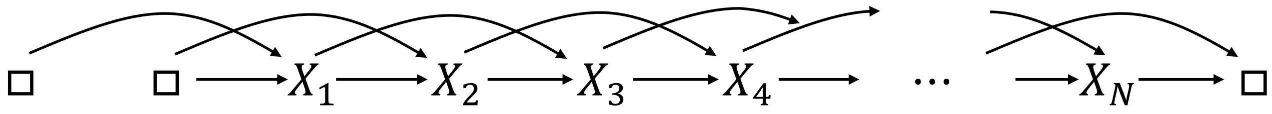


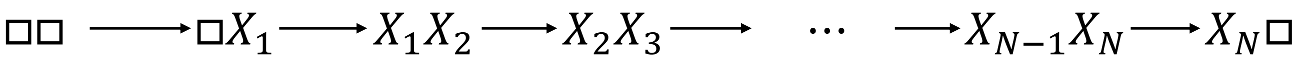
Let's let $\Pr(w|uv) = P_{(uv)w}$

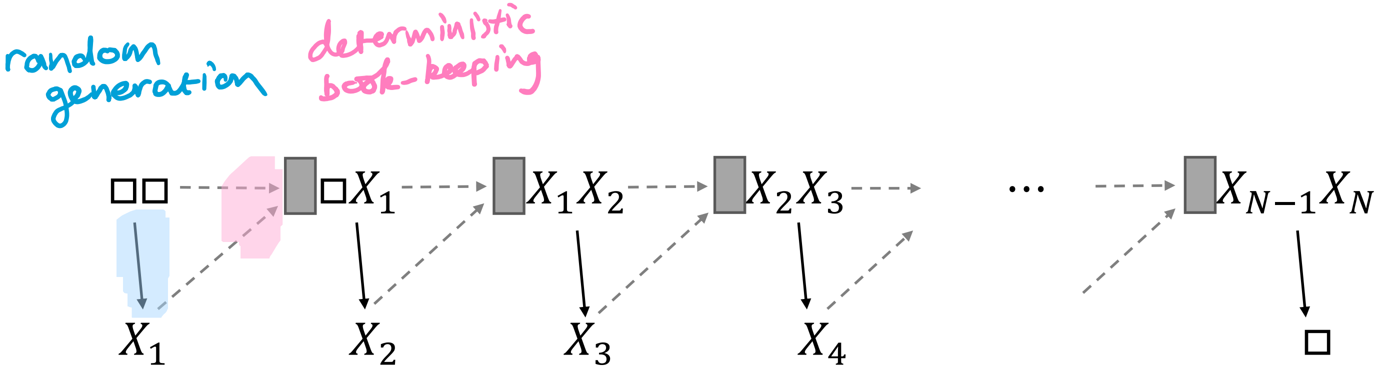
It's easy to estimate P, the (word,word)-to-word transition probabilities, by simple counting.

(Before counting, preprocess the dataset by putting □□ at the start and □ at the end of every sentence.)

Different ways to write the trigram model:







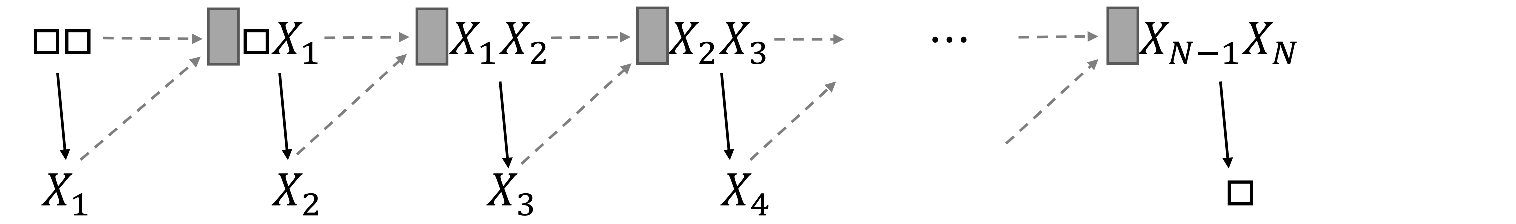
A *Markov Chain* is a sequence in which each item is generated based only on the preceding item.

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

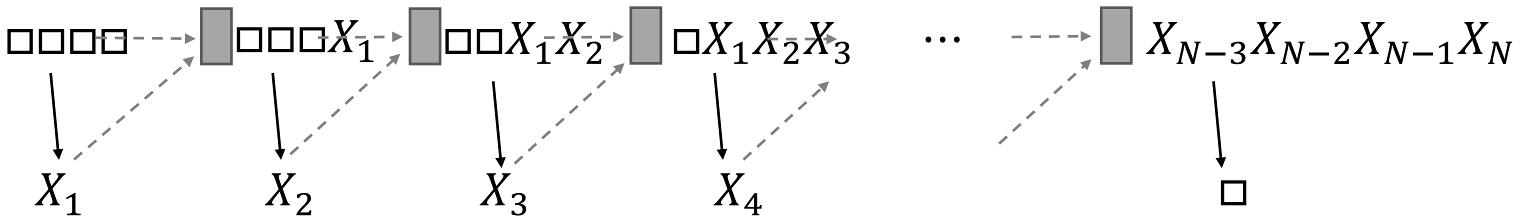
deterministic bookkeeping function $f((x, y), z) = (y, z)$

$(x, y) \to X_{\text{new}}$ random generation

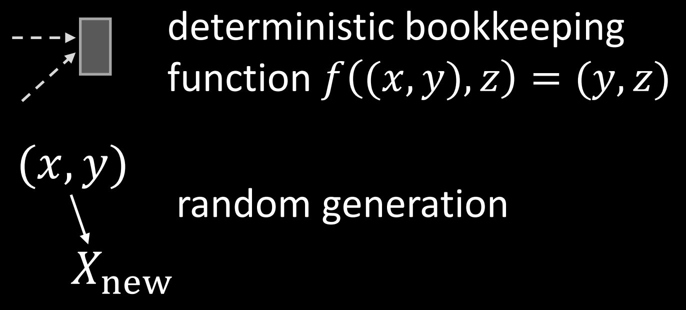
Can we get a better model by using more history?



Trigram character-by-character model trained on Shakespeare:
`"on youghtlee for vingiond do my not whow'd no crehout withal deepher forand a but thave a doses?"`



5-gram character-by-character model trained on Shakespeare:
`"once is pleasurely. though the the with them with comes in hand. good. give and she story tongue."`

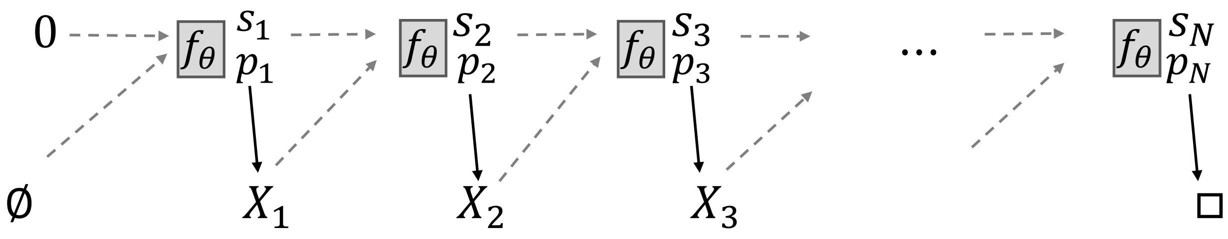


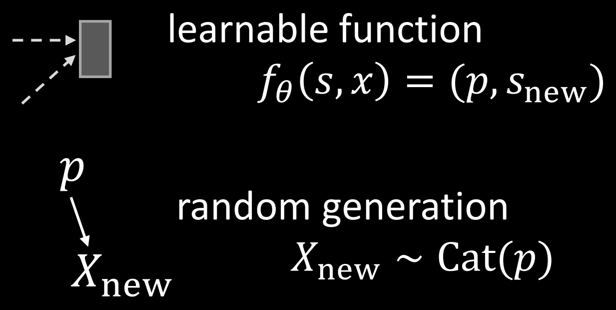
QUESTION. What are the advantages and disadvantages of a long history window?

QUESTION. Can we do better than using a fixed history window?

Recurrent Neural Network (RNN)

Let's use a neural network to learn an appropriate history digest. This is more flexible than choosing a fixed history window.

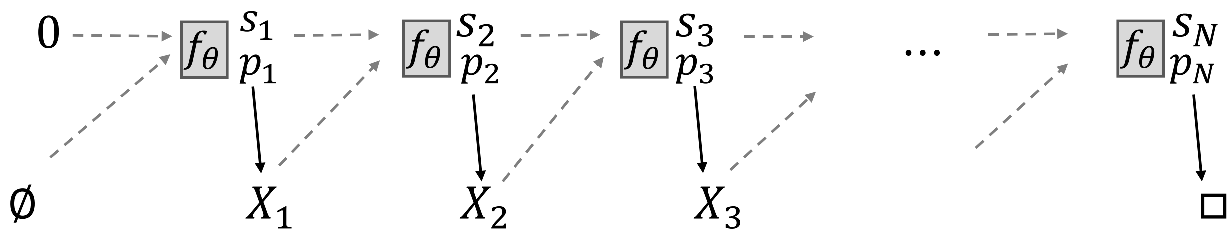




RNN character-by-character model trained on Shakespeare
[due to Andrej Karpathy]:

```
“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.”
```

A Recurrent Neural Network (RNN) is a probability model for generating a random sequence $\underline{X}$.



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

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

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:

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

by the chain rule for probability

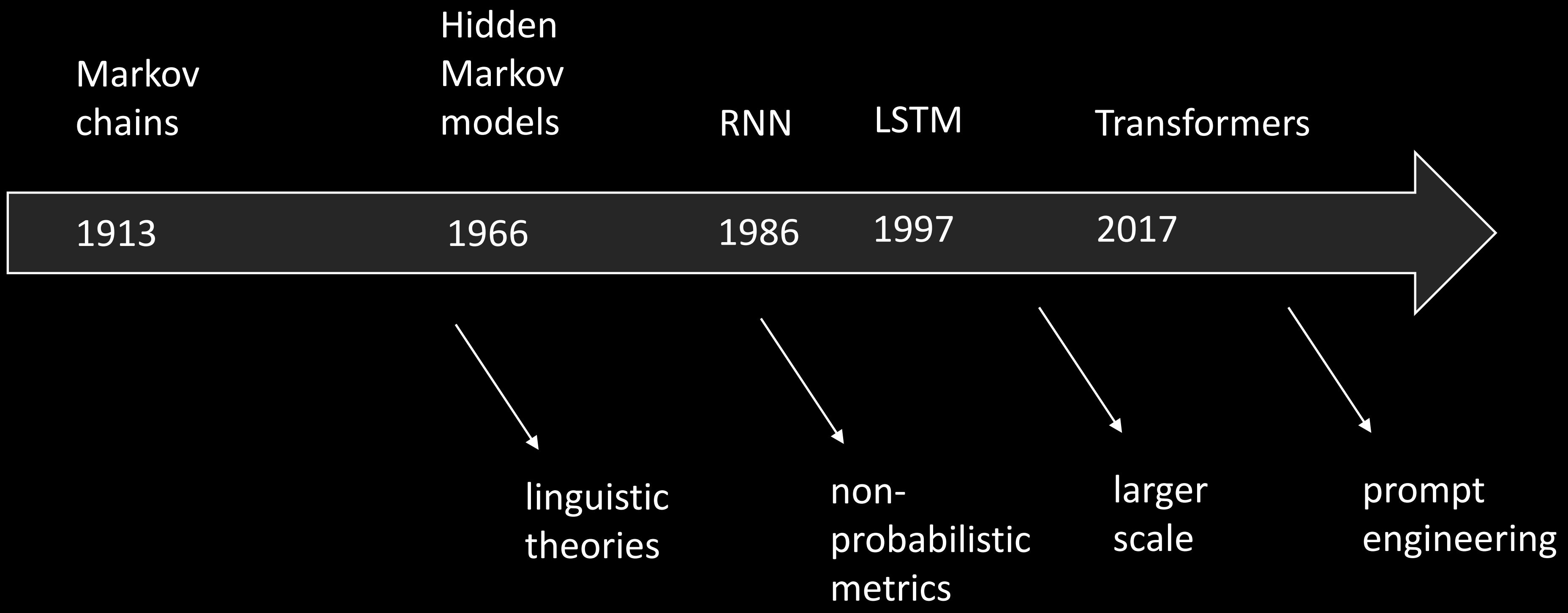
$$= [p_1]_{x_1}\, [p_2]_{x_2} \times \cdots \times [p_n]_{x_n}\, [p_{n+1}]_\square$$

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

$\mathbb{P}(A \text{ and } B \text{ and } C) = \mathbb{P}(A)\, \mathbb{P}(B|A)\, \mathbb{P}(C|A,B)$

```
def loglik(xstr):
    res = 0
    s,x = 0,□
    for x_next in xstr + "□":
        s,p = f_θ(s,x)
        res += log(p[x_next])
        x = x_next
    return res
```

The history of random sequence models



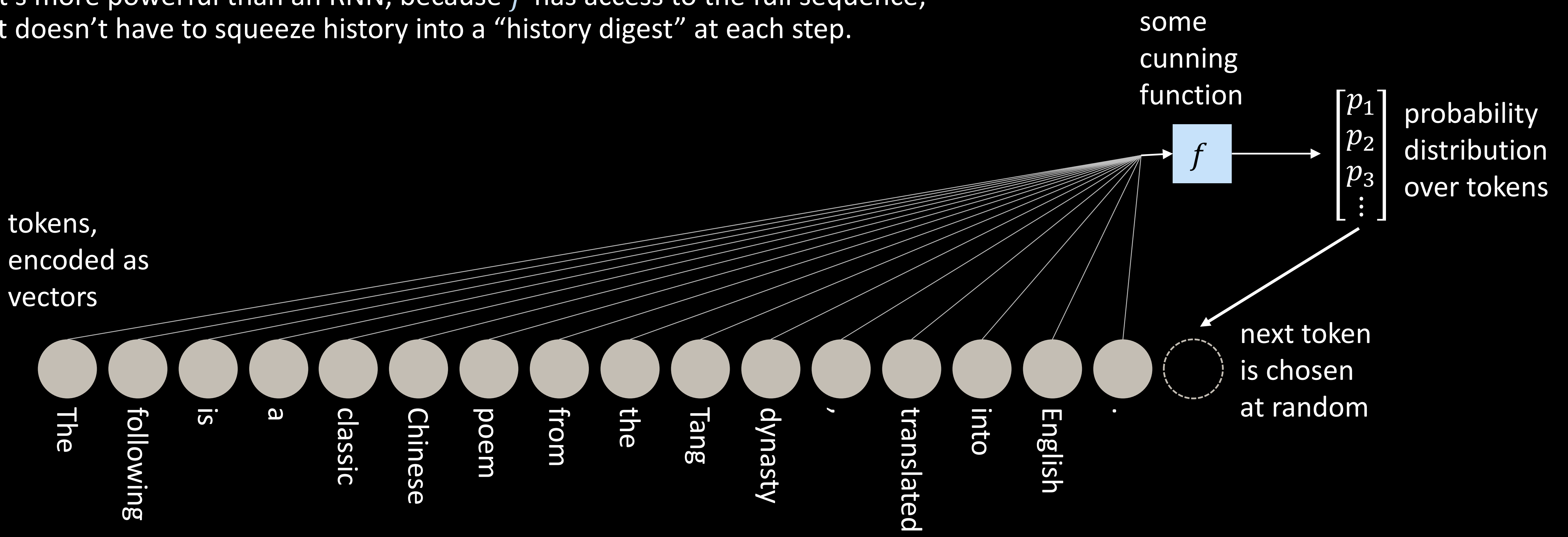
Better models of the data
All trained by maximizing the log likelihood of the data

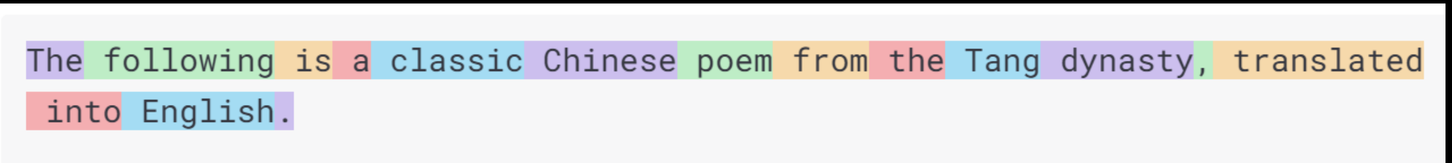
Transformer architecture

This is a probability model for a random sequence $\underline{X}$.

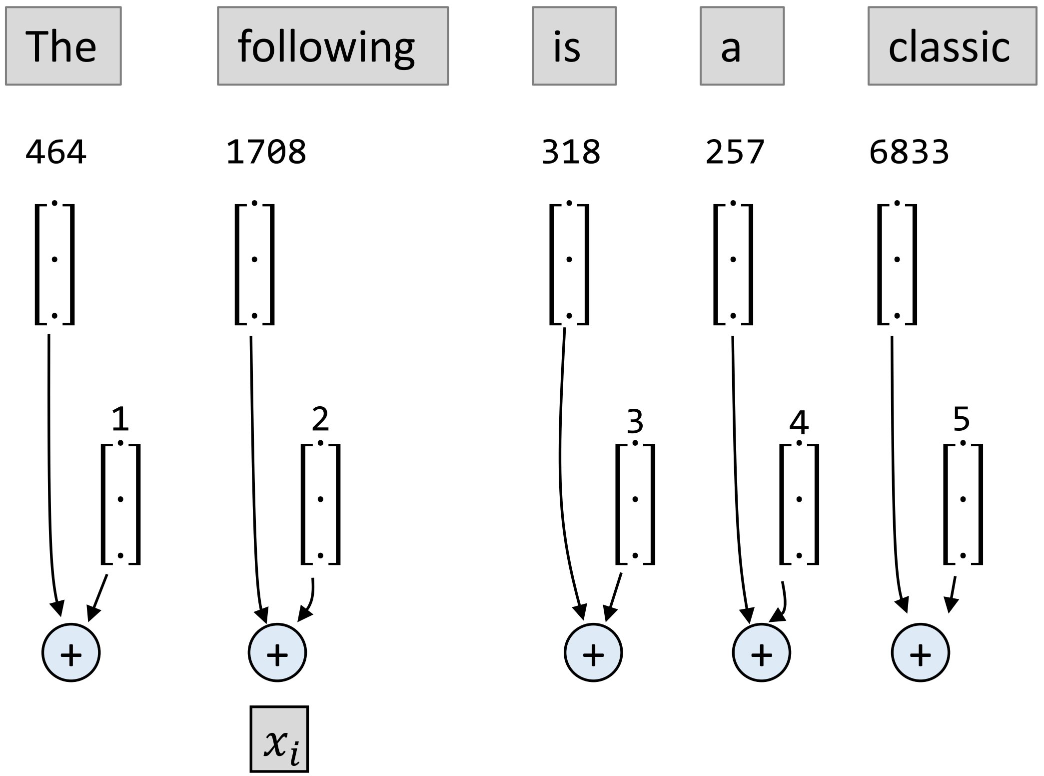
Like the RNN, there's a simple explicit formula for the log likelihood $\Pr_{\underline{X}}(\underline{x})$, so it's easy to train.

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.





What does f look like? How is it built out of differentiable functions?



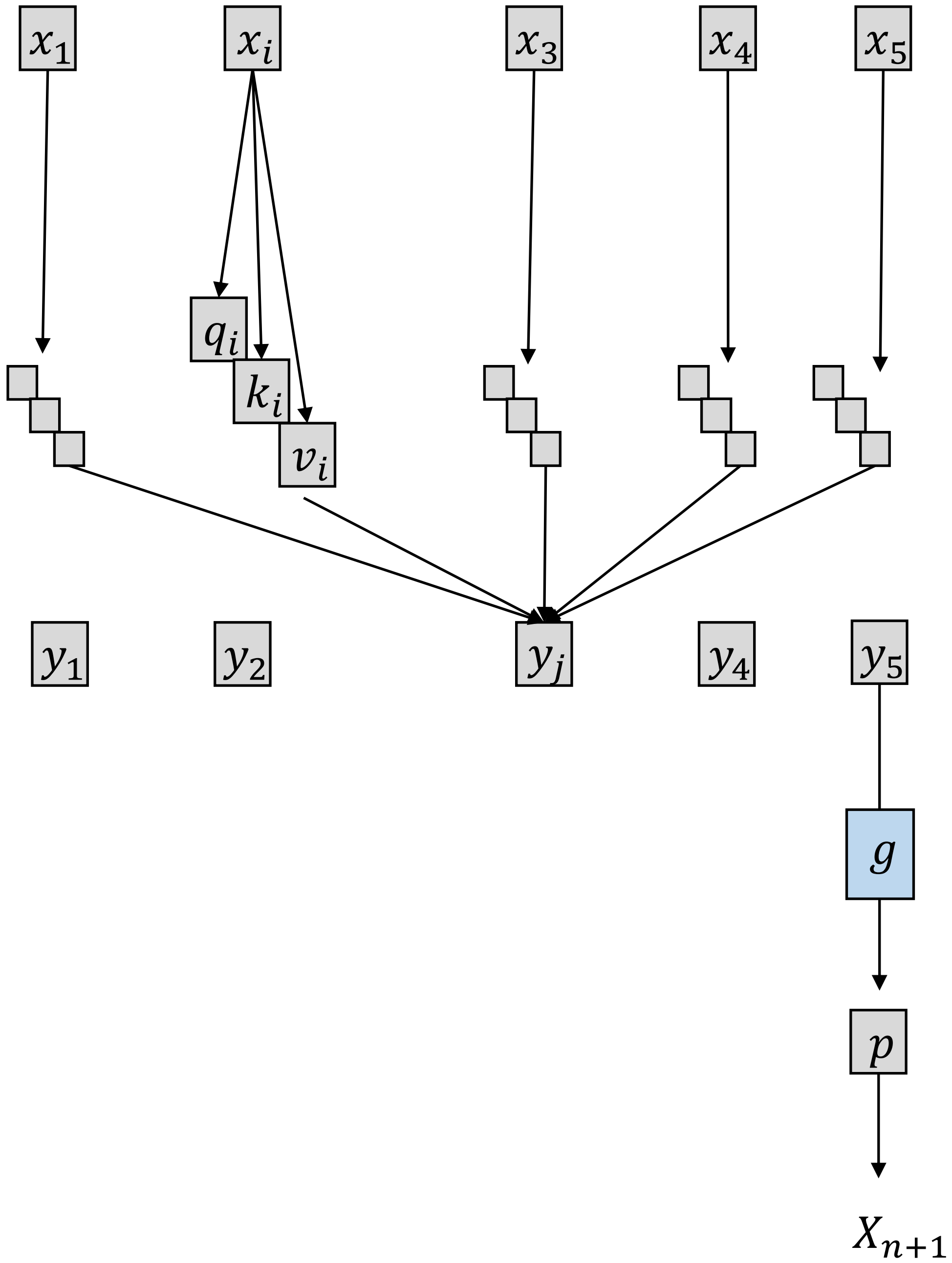
Split the text into tokens $t_i \in \{1, \ldots, W\}$

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 $t_i \in \mathbb{R}^d$

$$\begin{bmatrix} \sin(i) \\ \cos(i) \\ \sin(i/2) \\ \cos(i/2) \\ \vdots \end{bmatrix}$$

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 $v_i = Vx_i$
$\in \mathbb{R}^e$ $\in \mathbb{R}^e$ $\in \mathbb{R}^d$

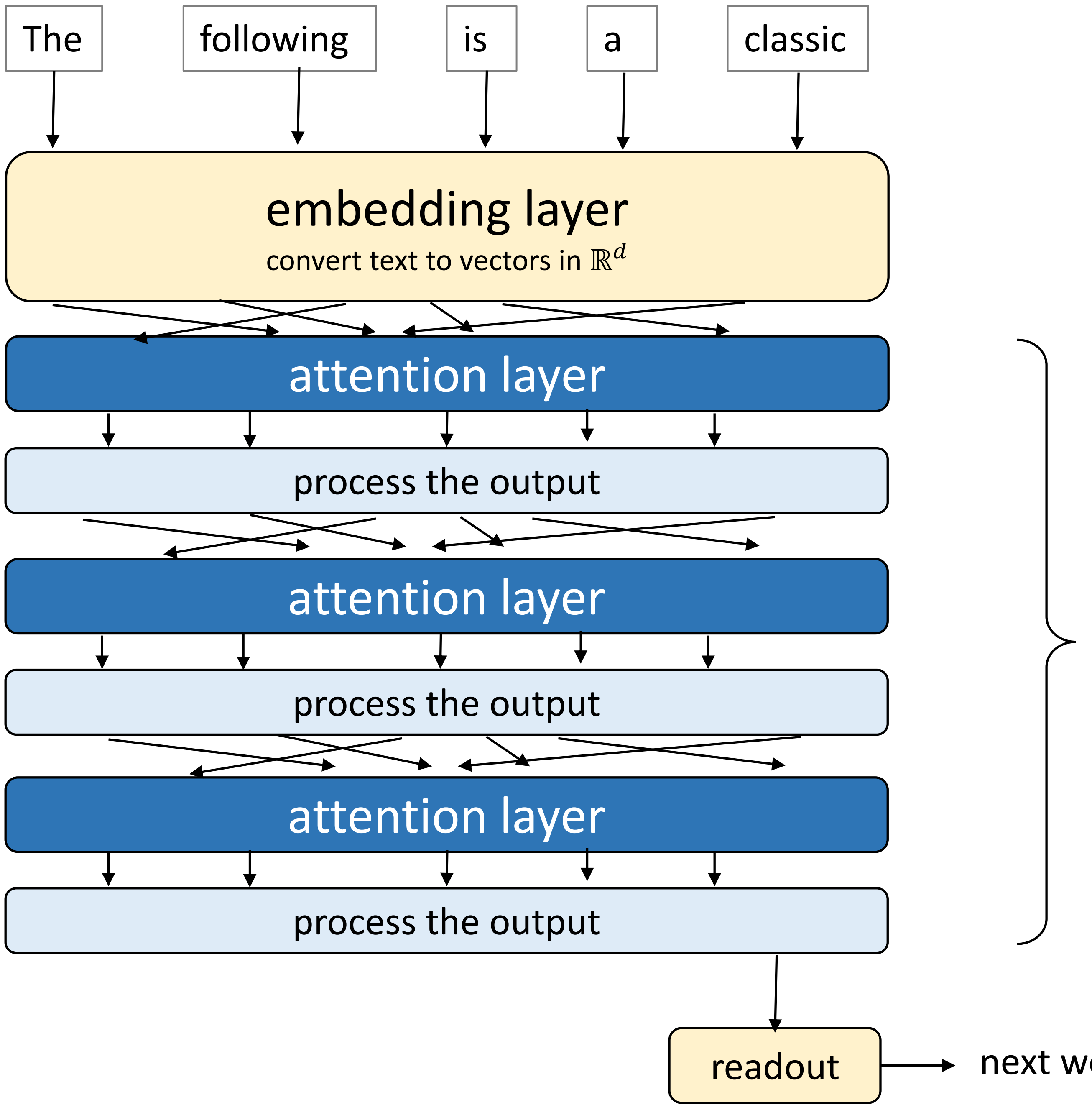
For each position $j \in \{1, \ldots, n\}$ we'll produce an output vector $y_j \in \mathbb{R}^d$, as follows:

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

a_{ji} is the attention that we should give to input x_i when computing output y_j

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 $X_{n+1} \sim \mathrm{Cat}(p)$



In practice, it's useful to use several passes of the attention mechanism.