

Large language models (LLMs) are computational agents that can interact conversationally with people using natural language.

LLMs have revolutionized the field of NLP and AI.

Like simple n -gram language models, LLMs
assign probabilities to sequences of words and
generate text by sampling possible next words.

Like simple n -gram language models, LLMs
assign probabilities to sequences of words and
generate text by sampling possible next words.

But while n -gram language models are
trained on counts computed from lots of text,
LLMs are
trained by learning to guess the next word.

The fundamental intuition of large language models:

Text contains enormous amounts of knowledge!

Pretraining on lots of text, with all that knowledge, is what gives language models their ability to do so much.

What does a model learn from pretraining?

What does a model learn from pretraining?

With roses, dahlias, and peonies, I was surrounded by flowers

What does a model learn from pretraining?

With roses, dahlias, and peonies, I was surrounded by flowers

The room wasn't just big it was enormous

What does a model learn from pretraining?

With roses, dahlias, and peonies, I was surrounded by flowers

The room wasn't just big it was enormous

The square root of 4 is 2

What does a model learn from pretraining?

With roses, dahlias, and peonies, I was surrounded by flowers

The room wasn't just big it was enormous

The square root of 4 is 2

The author of "A Room of One's Own" is Virginia Woolf

What does a model learn from pretraining?

With roses, dahlias, and peonies, I was surrounded by flowers

The room wasn't just big it was enormous

The square root of 4 is 2

The author of "A Room of One's Own" is Virginia Woolf

The professor said that he [...]

Input

Context

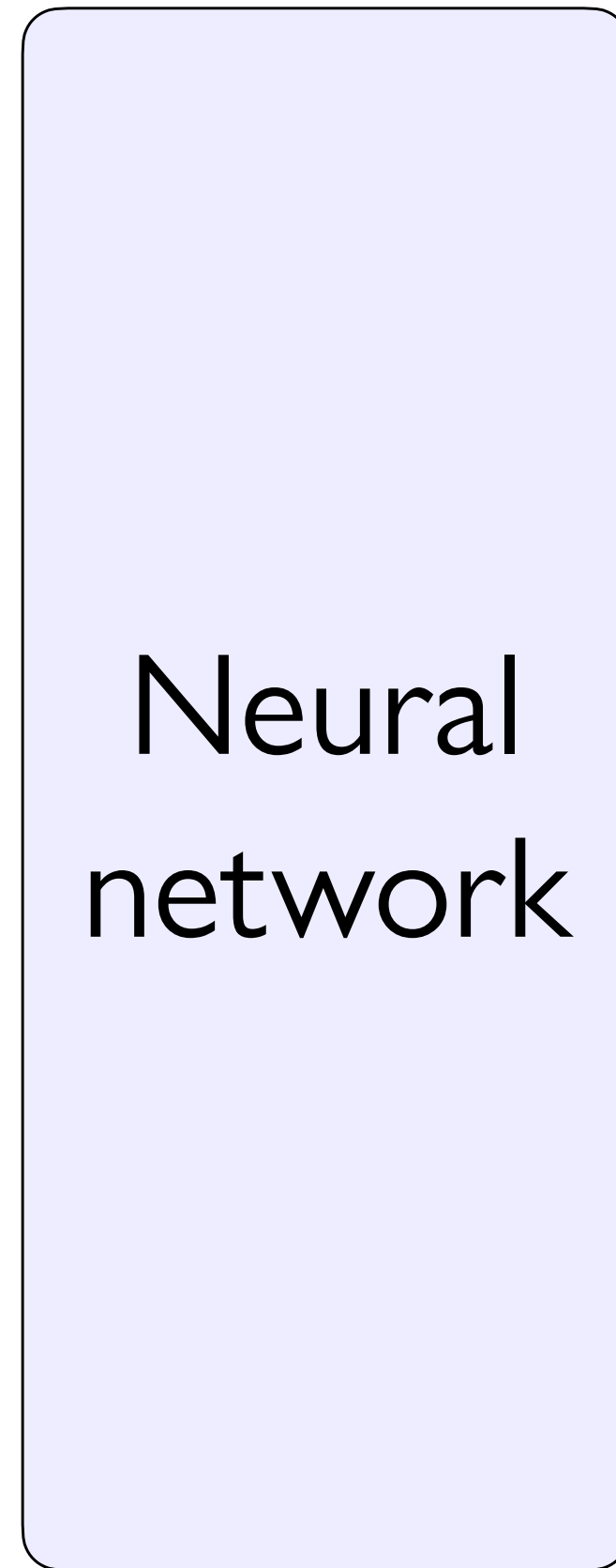
So

long

and

thanks

for



Neural
network



Output

$P(w \mid \text{context})$

all 0.44

the 0.33

your 0.15

that 0.08

⋮
⋮

A model that gives a probability distribution over next words can *generate text* by repeatedly sampling from the distribution.

Input

Context

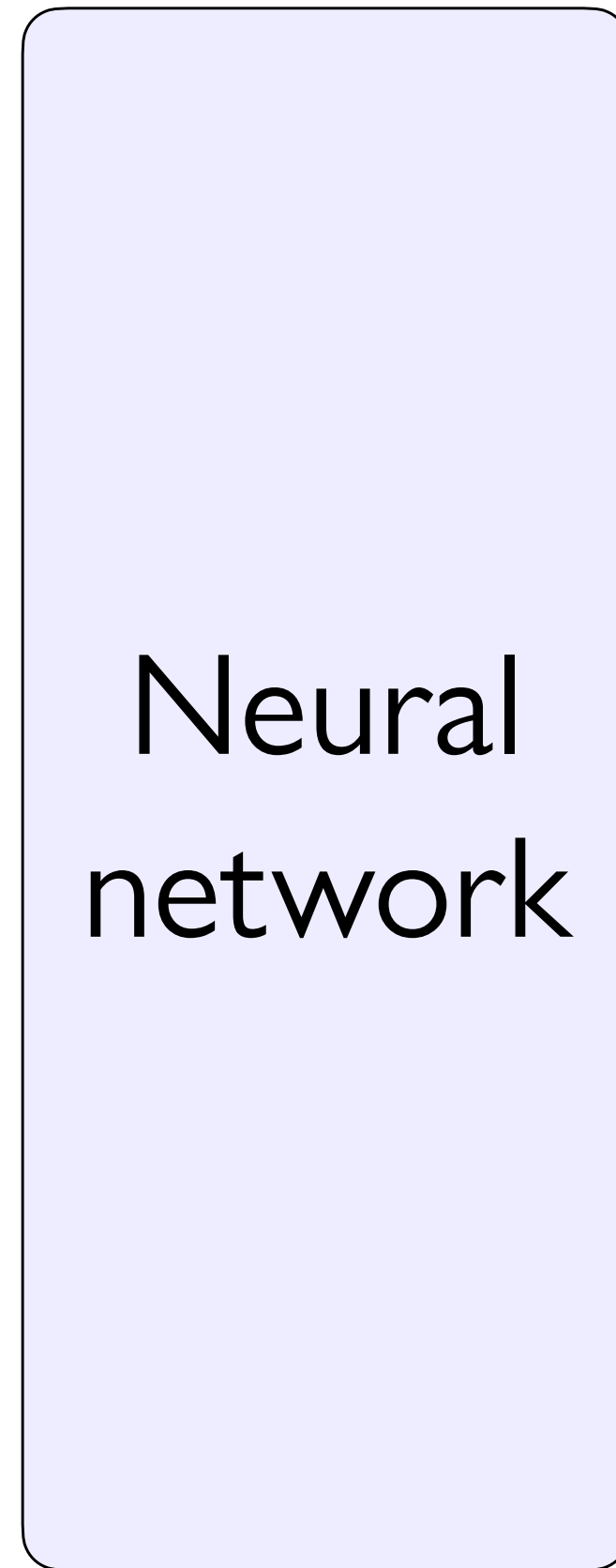
So

long

and

thanks

for



Neural
network



Output

$P(w \mid \text{context})$

all 0.44

the 0.33

your 0.15

that 0.08

⋮
⋮

Input

Context

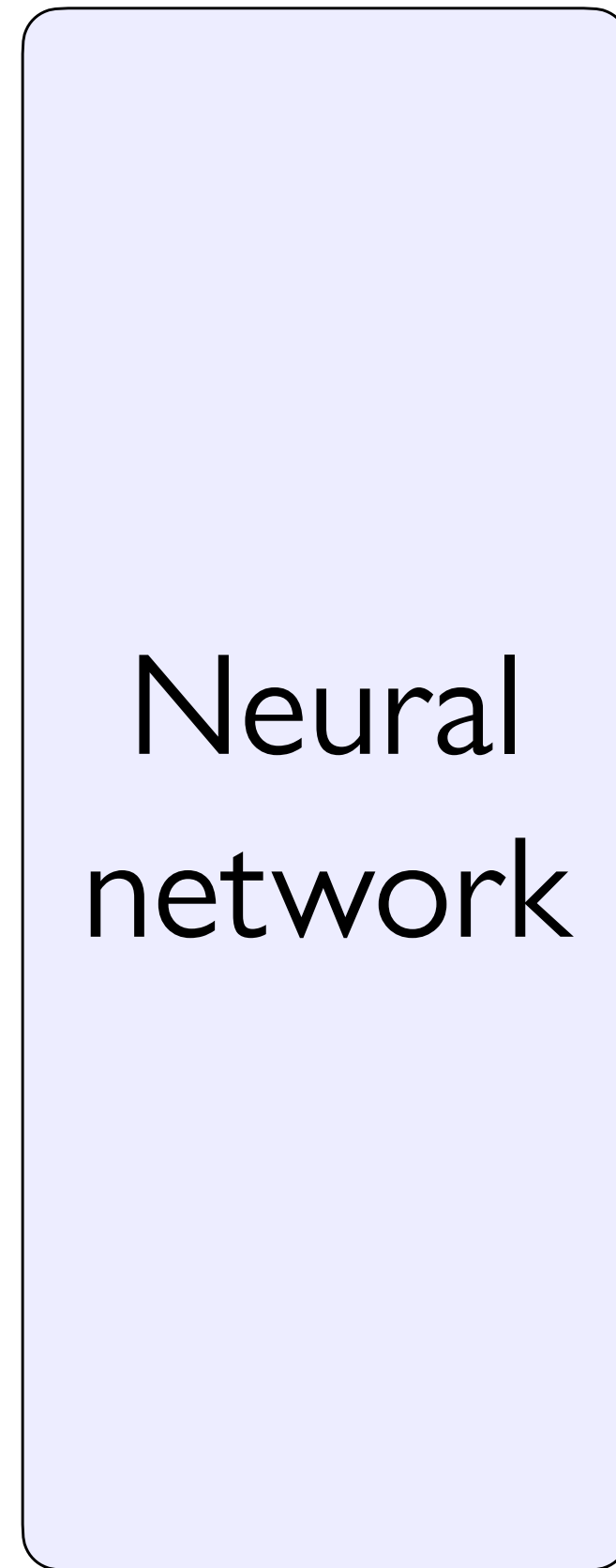
So

long

and

thanks

for



Neural
network



Output

$P(w \mid \text{context})$

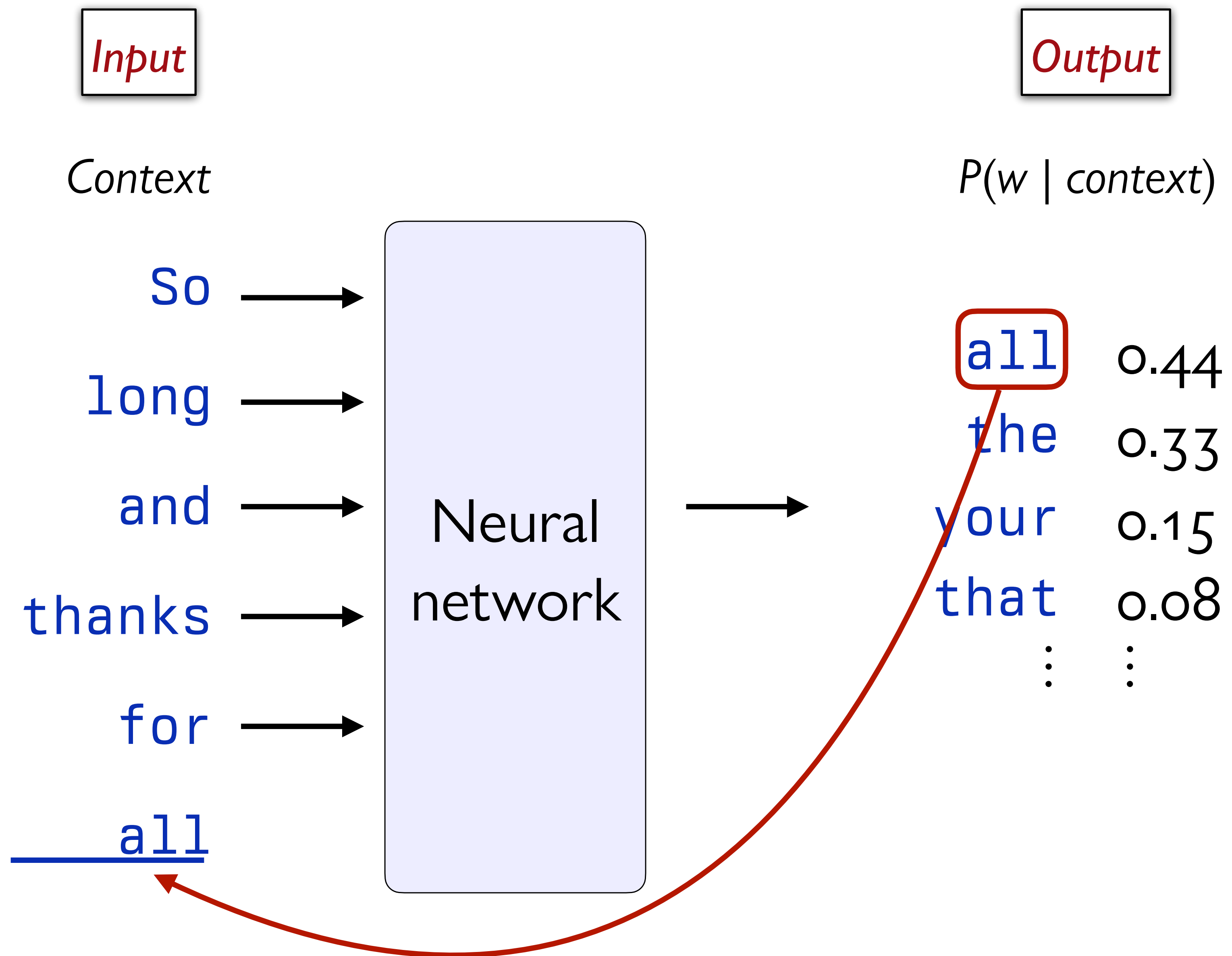
all 0.44

the 0.33

your 0.15

that 0.08

⋮
⋮



Input

Context

So

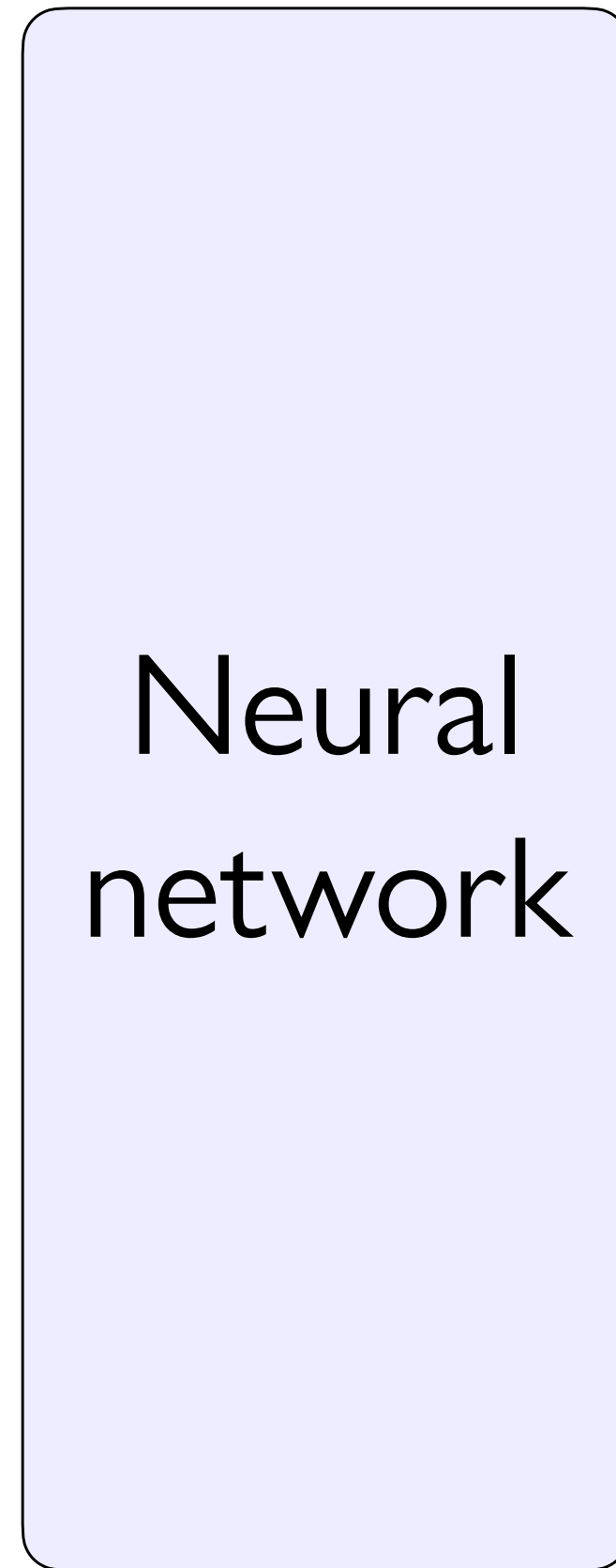
long

and

thanks

for

all



Neural
network



Output

$P(w \mid \text{context})$

Input

Context

So



long



and



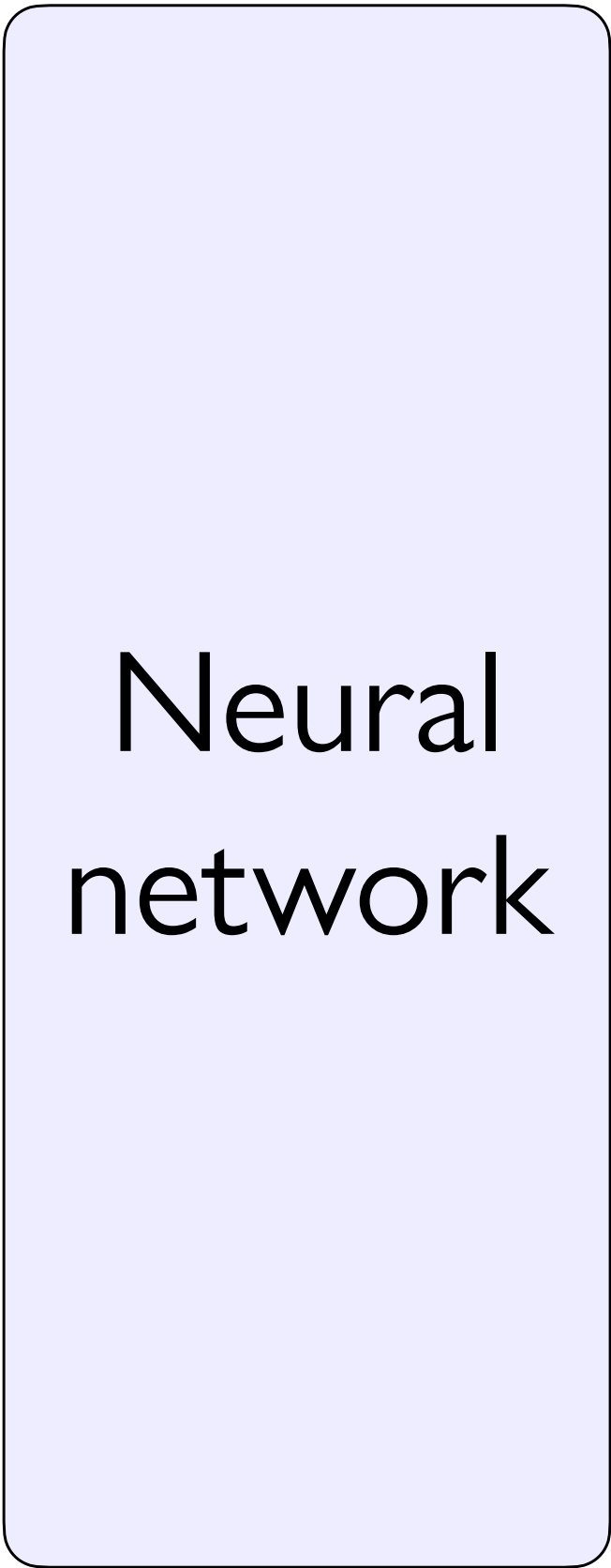
thanks



for



all



Output

$P(w \mid \text{context})$

the

0.77

your

0.22

our

0.07

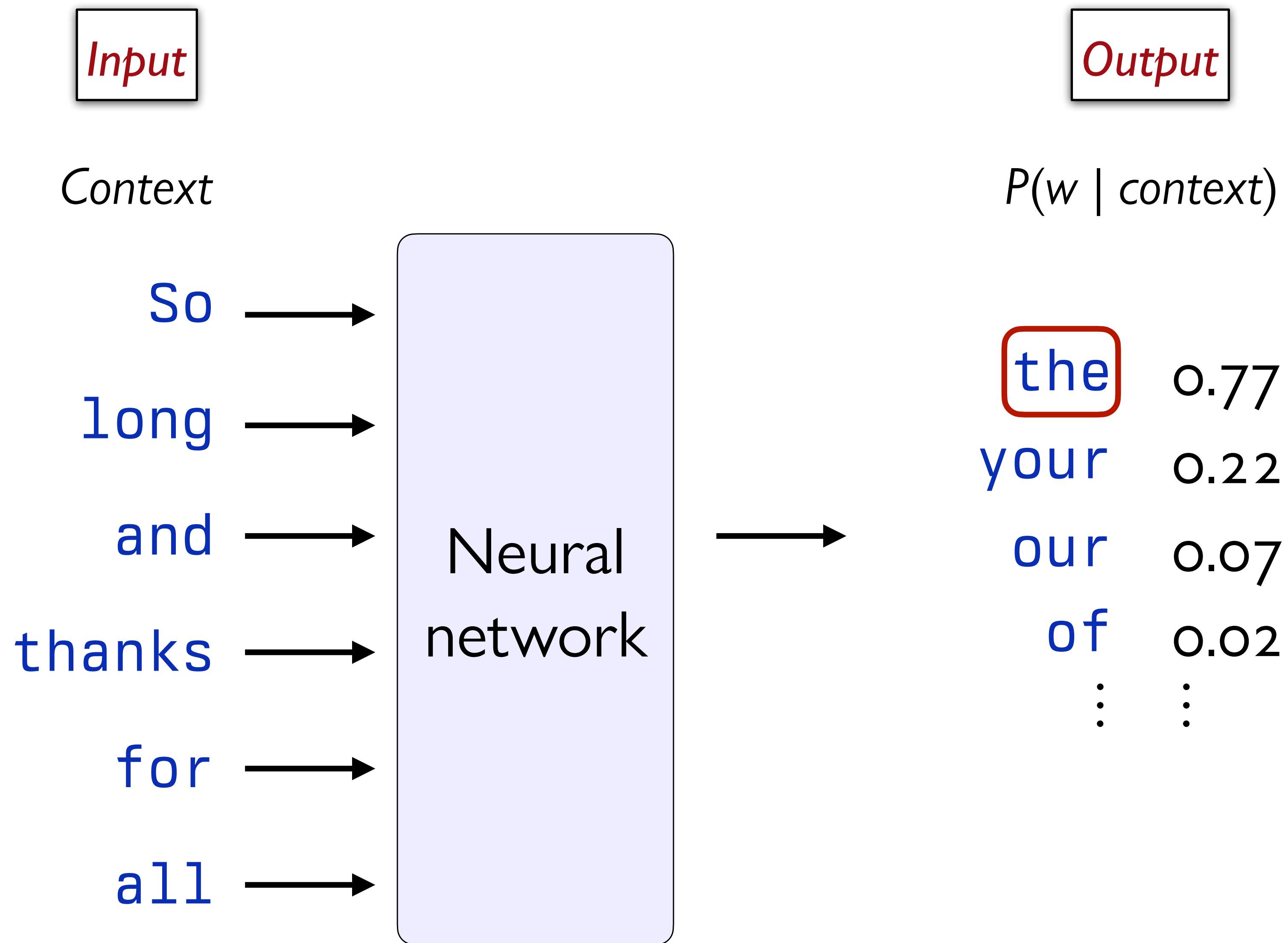
of

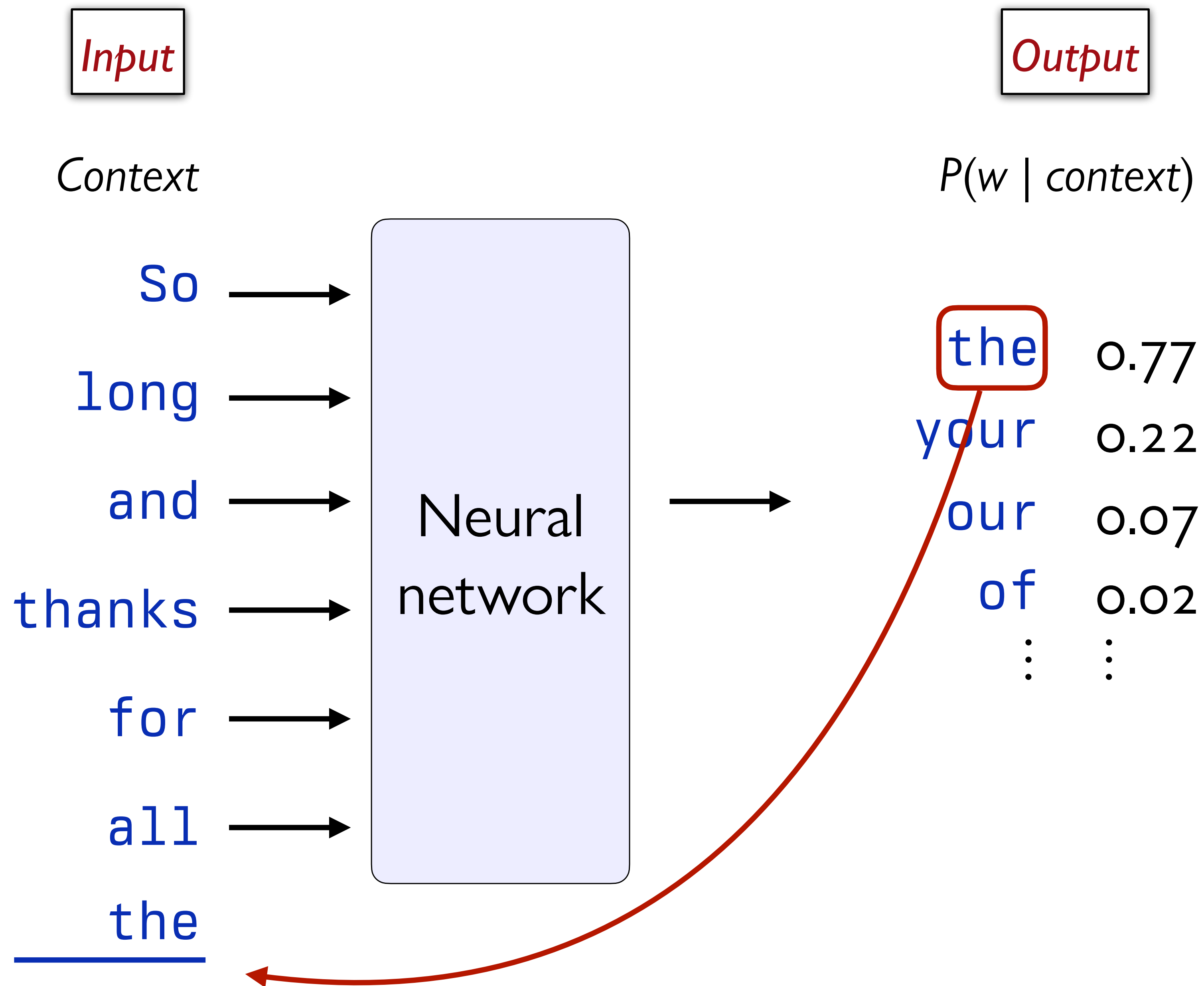
0.02

⋮

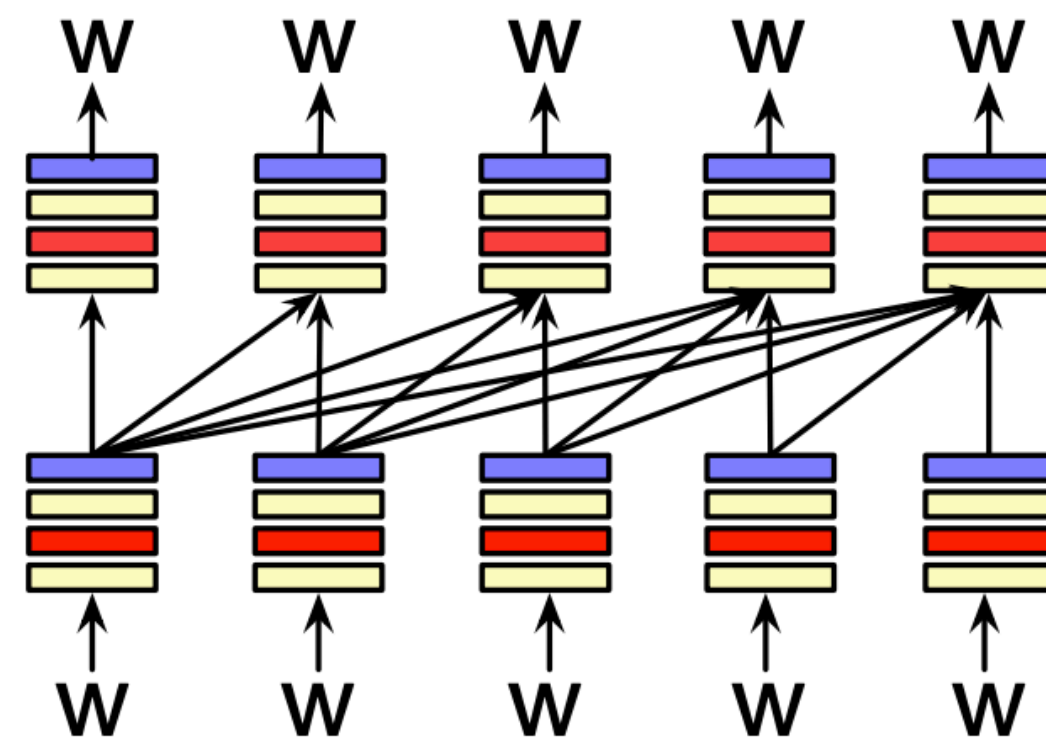
⋮



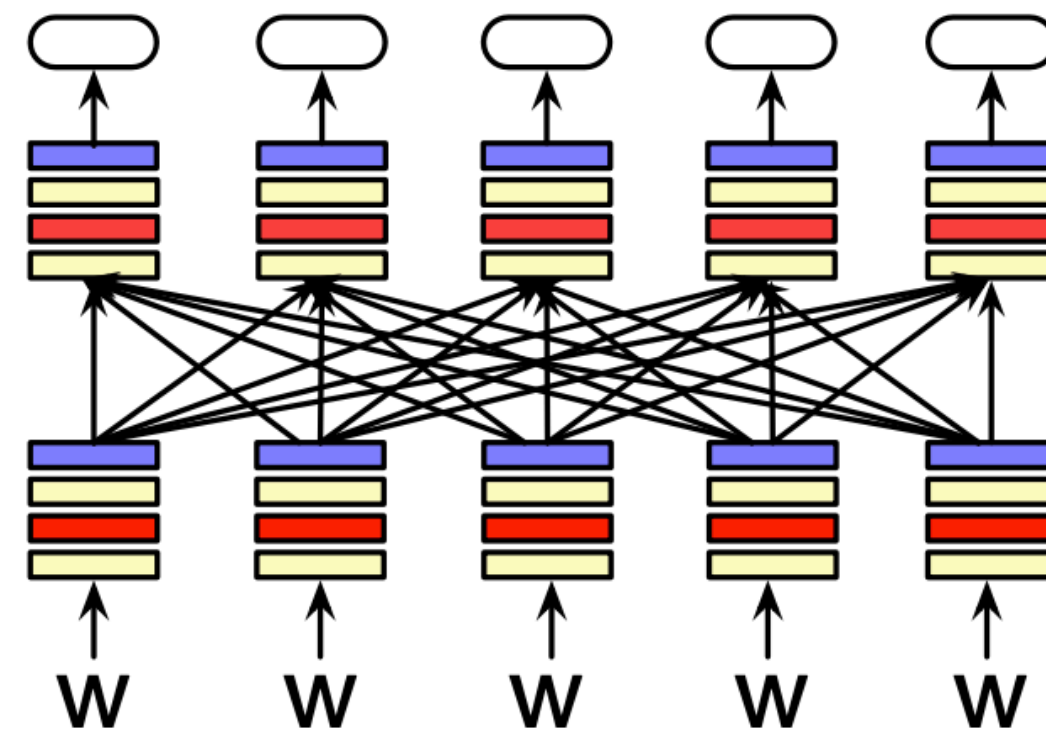




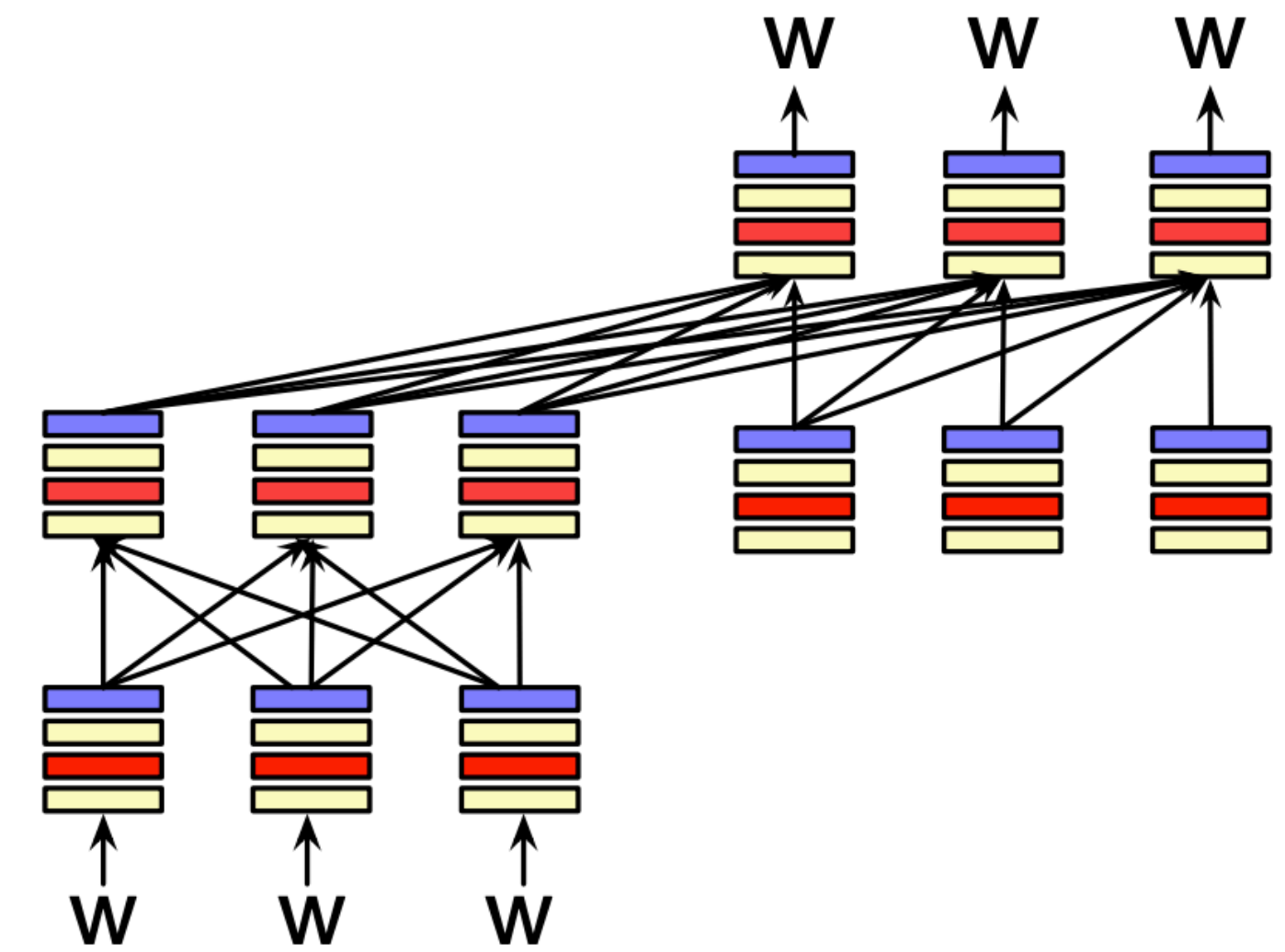
Three architectures for LLMs



Decoders

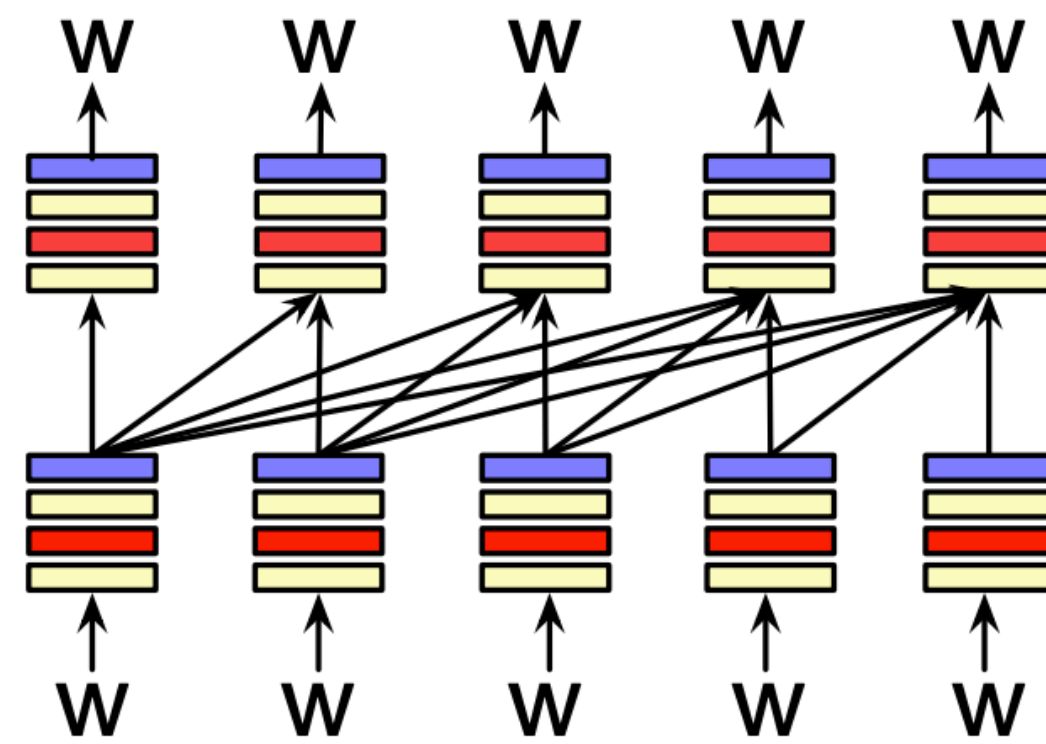


Encoders

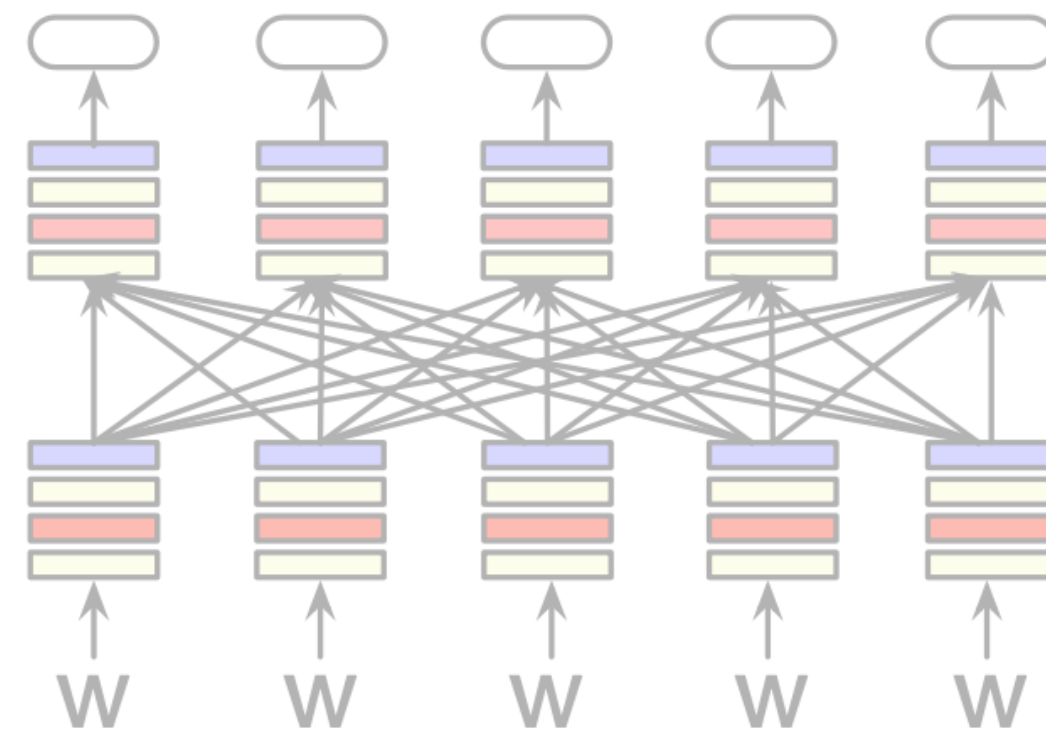


Encoder-decoders

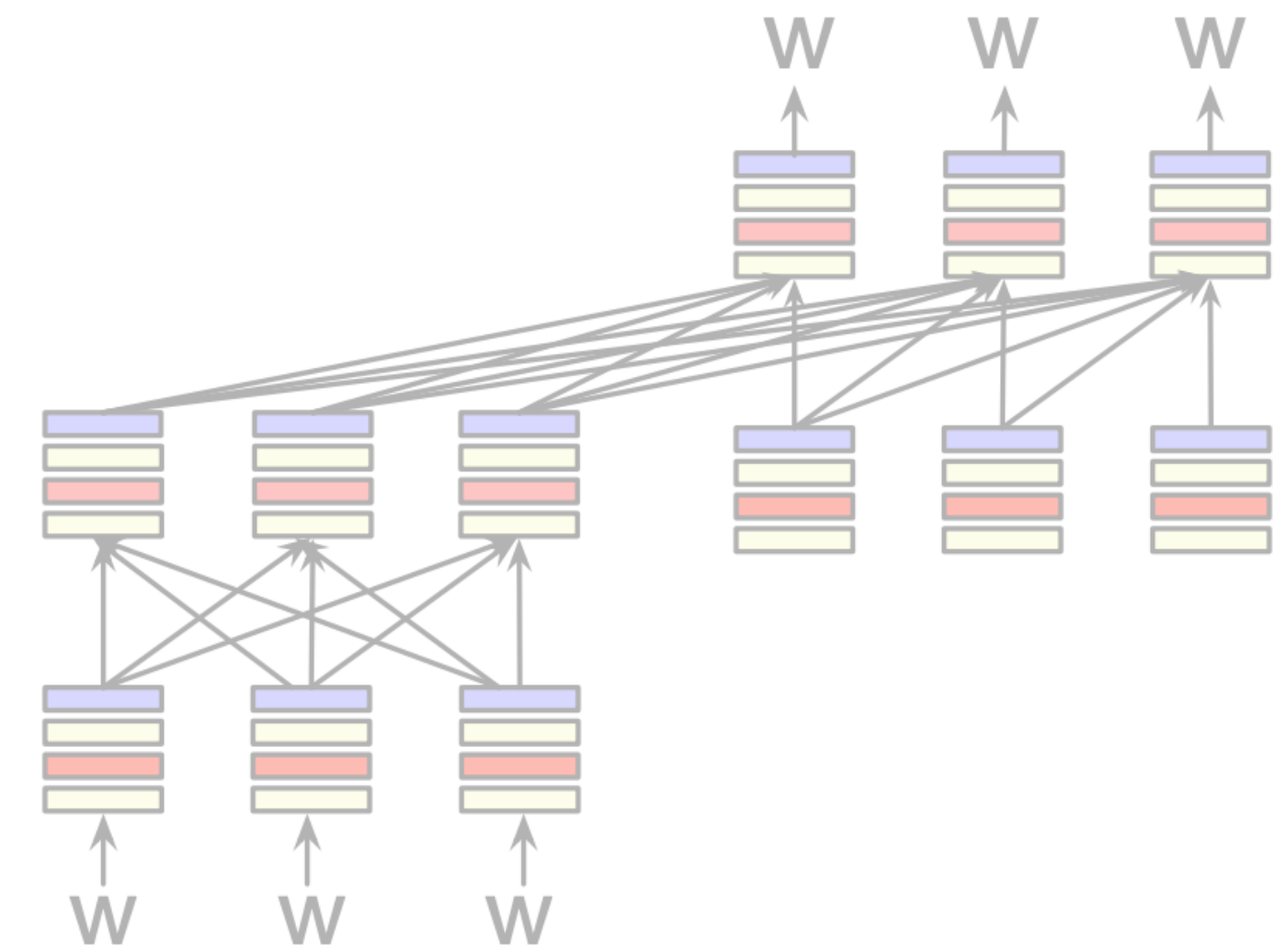
A *decoder* takes a sequence of tokens as input and generates a sequence of tokens as output, one at a time.



Decoders

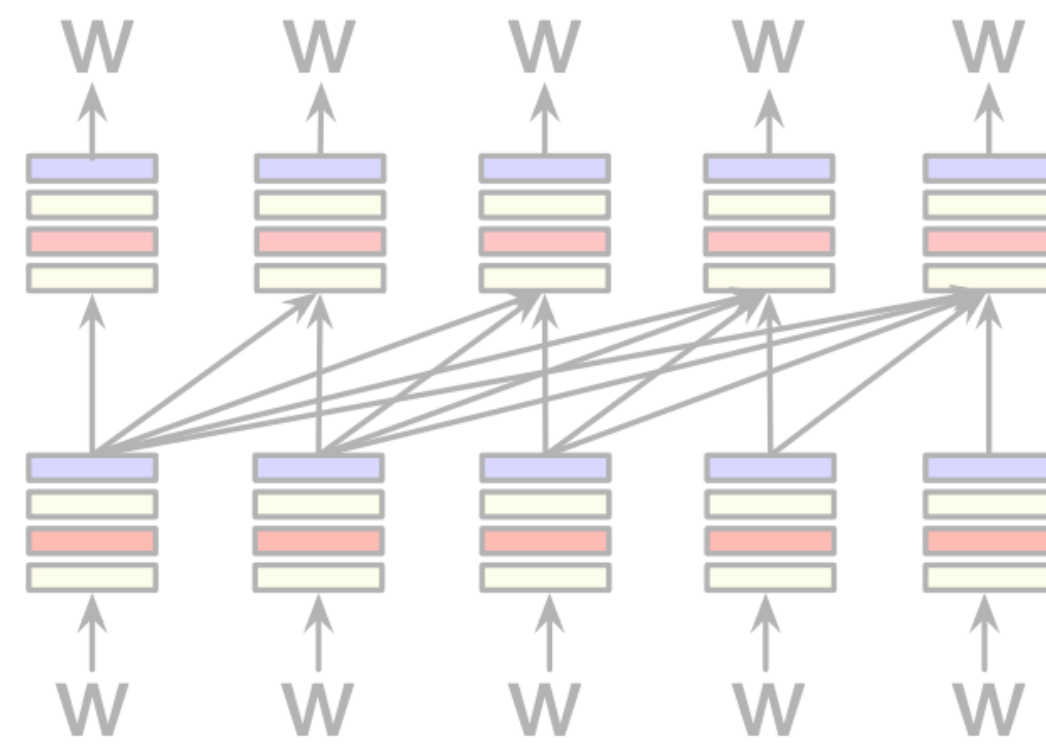


Encoders

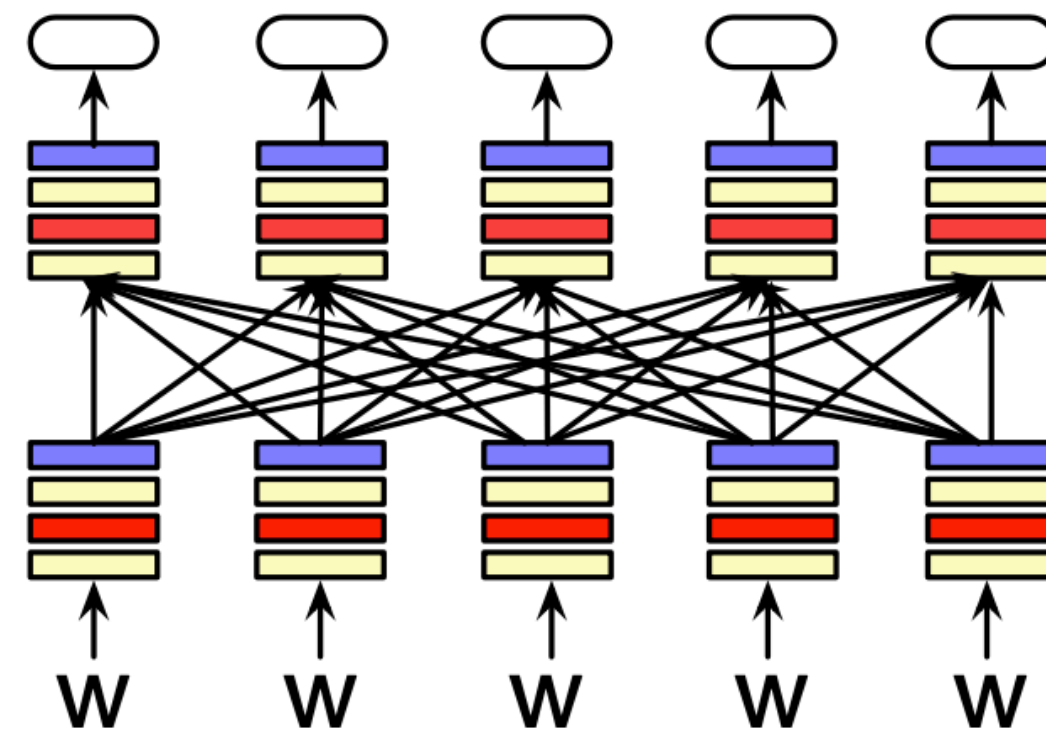


Encoder-decoders

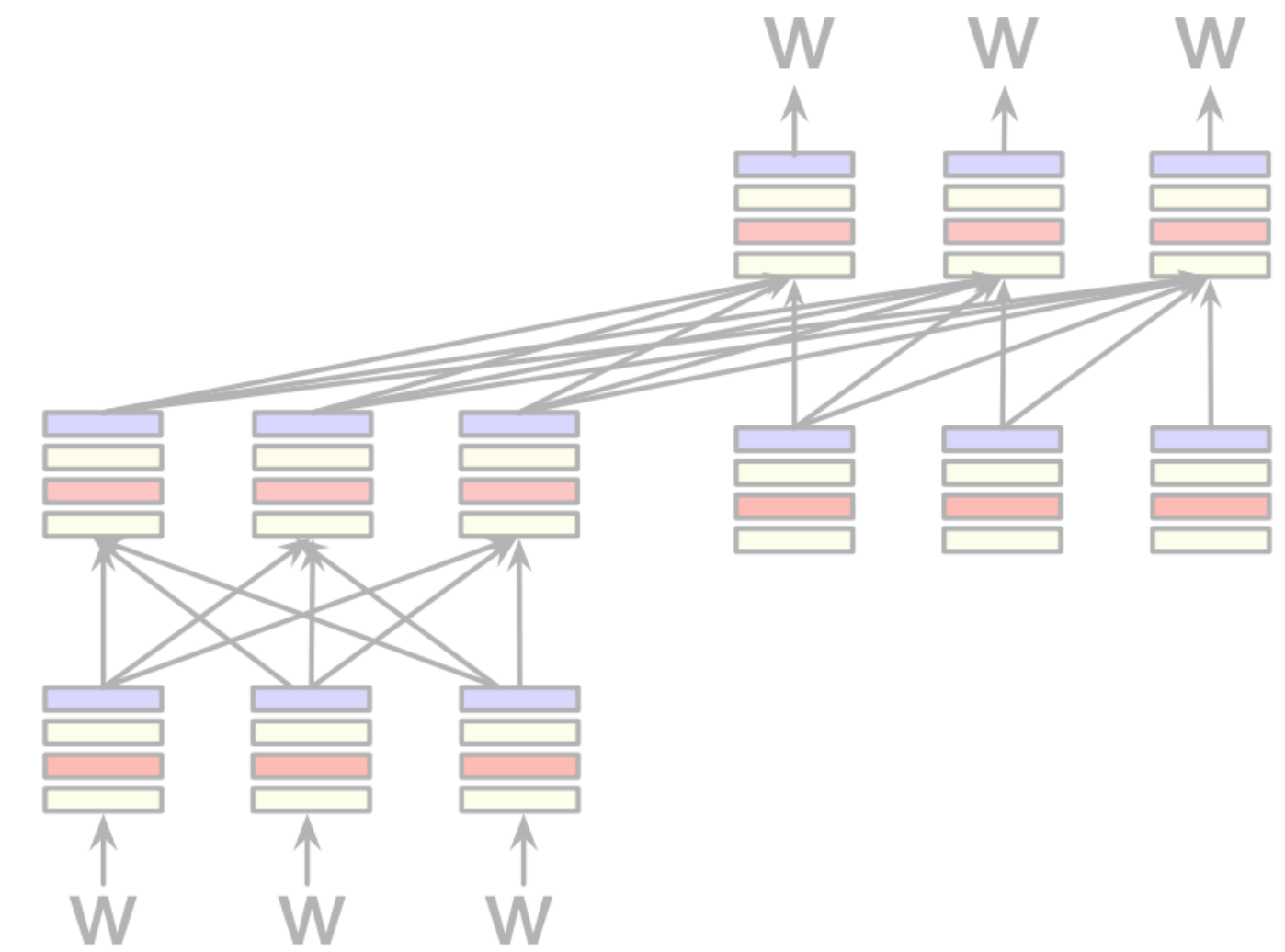
An *encoder* is a model like BERT, which takes a sequence of tokens as input and outputs a *vector representation* for each token. These can be used to make classifiers (like on Asmt 5!)



Decoders

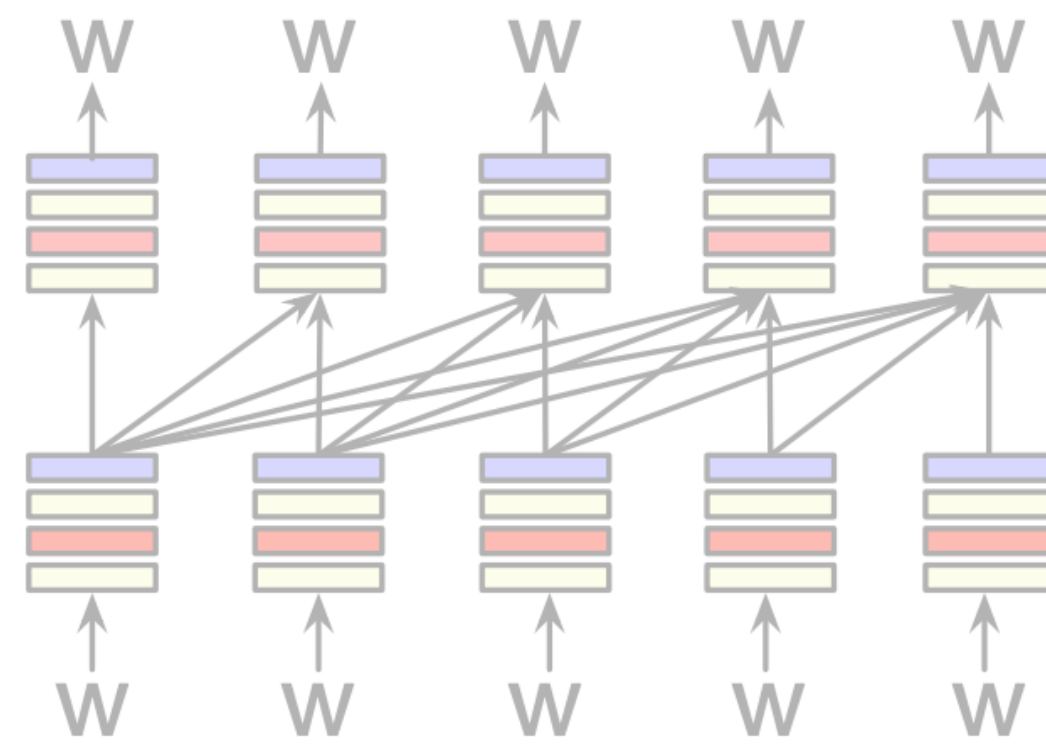


Encoders

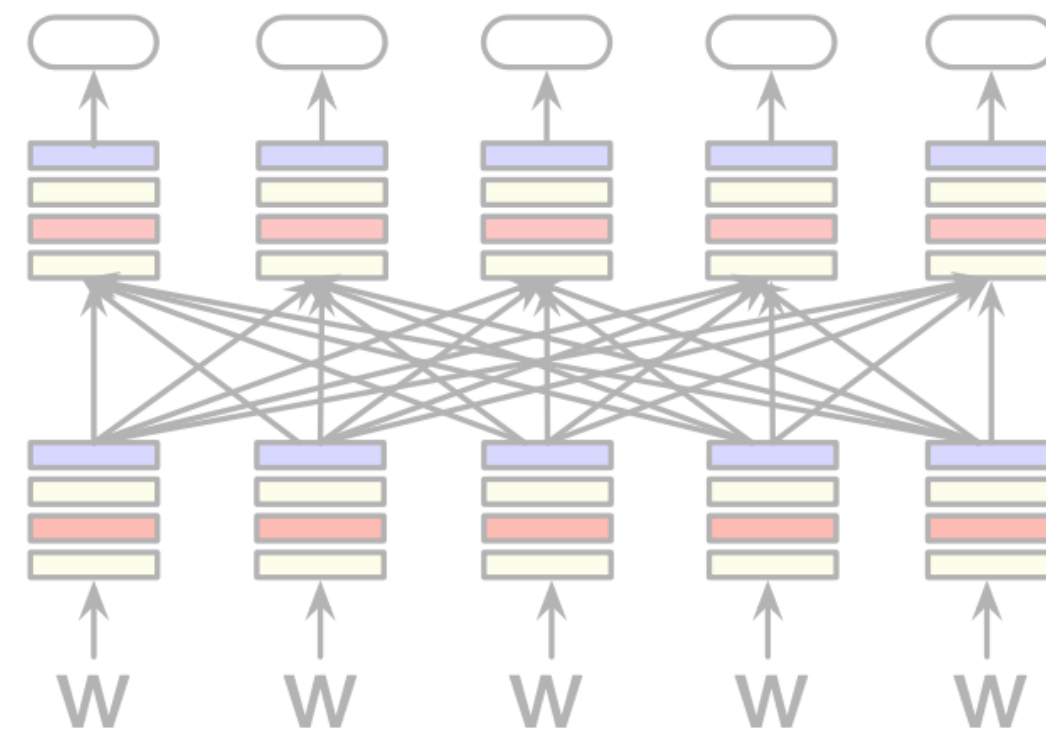


Encoder-decoders

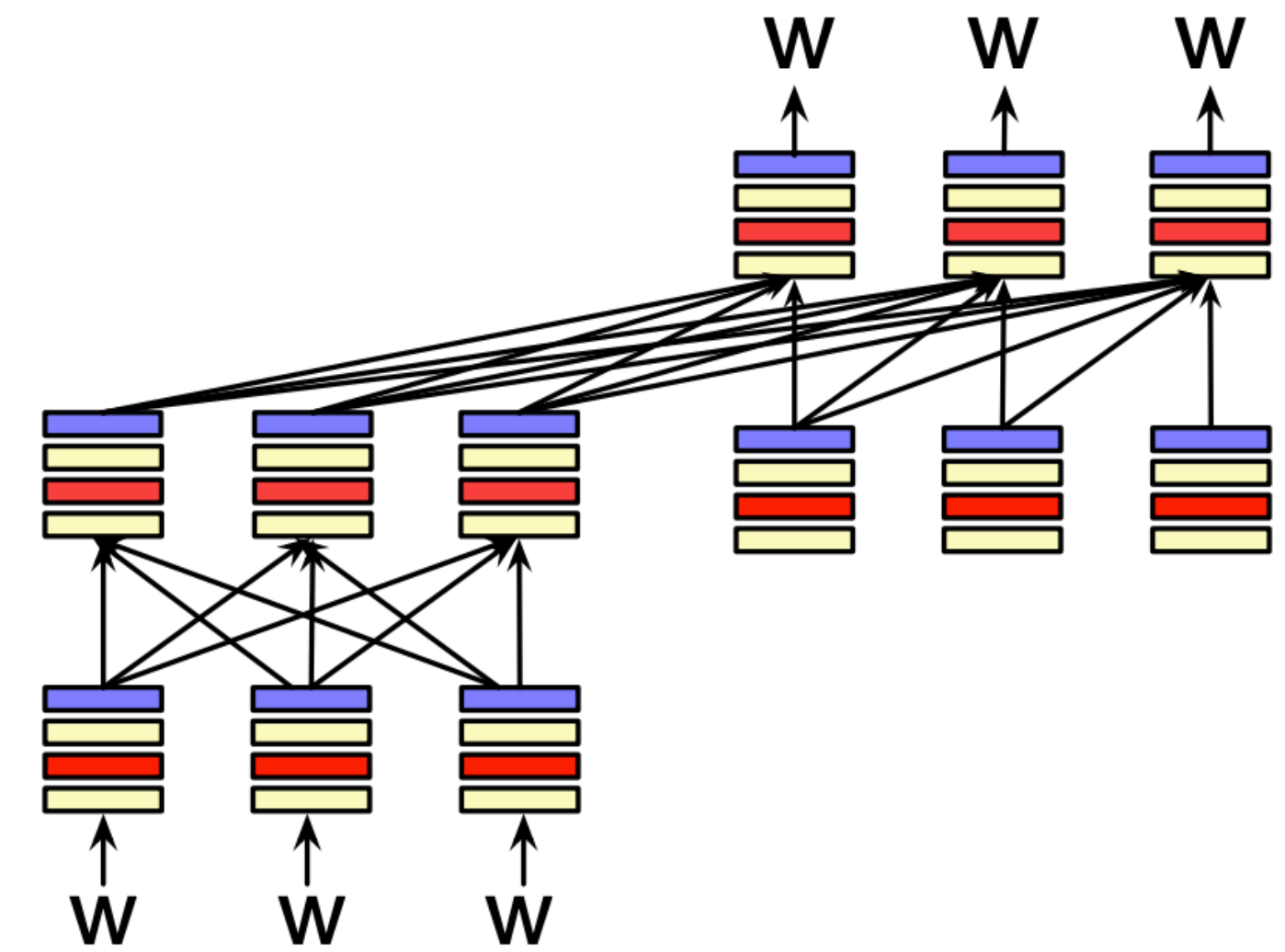
An *encoder–decoder* takes as input a sequence of tokens and outputs a series of tokens – but the input and output are less directly connected, and the set of tokens used for inputs can be different from those used for outputs, e.g., translating from one language to another.



Decoders



Encoders



Encoder–decoders

Conditional generation of text and prompting

Big idea: Almost anything we want to do with language can be turned into a task of predicting words – and therefore can be solved using decoder language models!

The task of generating text based on previous text is called *conditional generation*.

Give the LLM an input text called a *prompt*.

Have it generate token by token, conditioned on the prompt and the tokens generated so far:

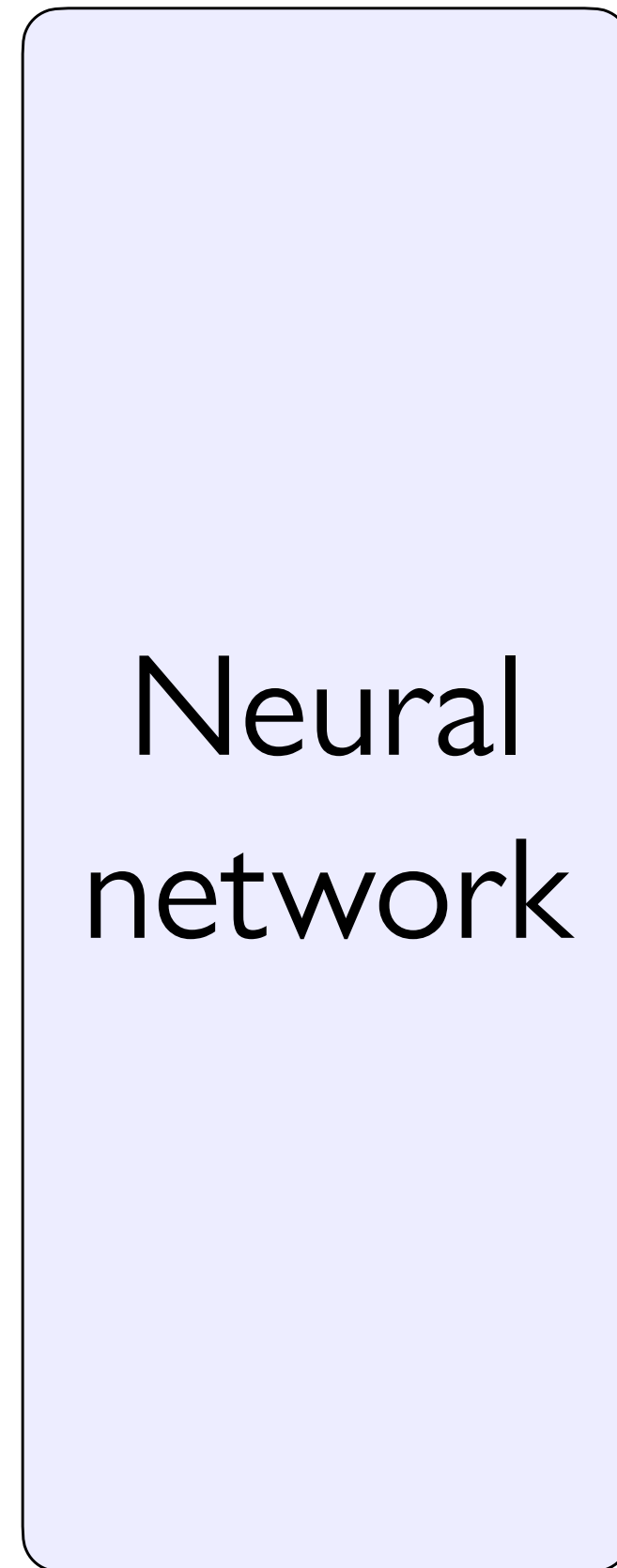
Compute the probability of the next token w_i from the prior context: $P(w_i \mid w_{<i})$, and

Sample from that distribution to generate a token.

Input

Context

The
sentiment
of the
sentence
"I like
Jackie
Chan" is:



Output

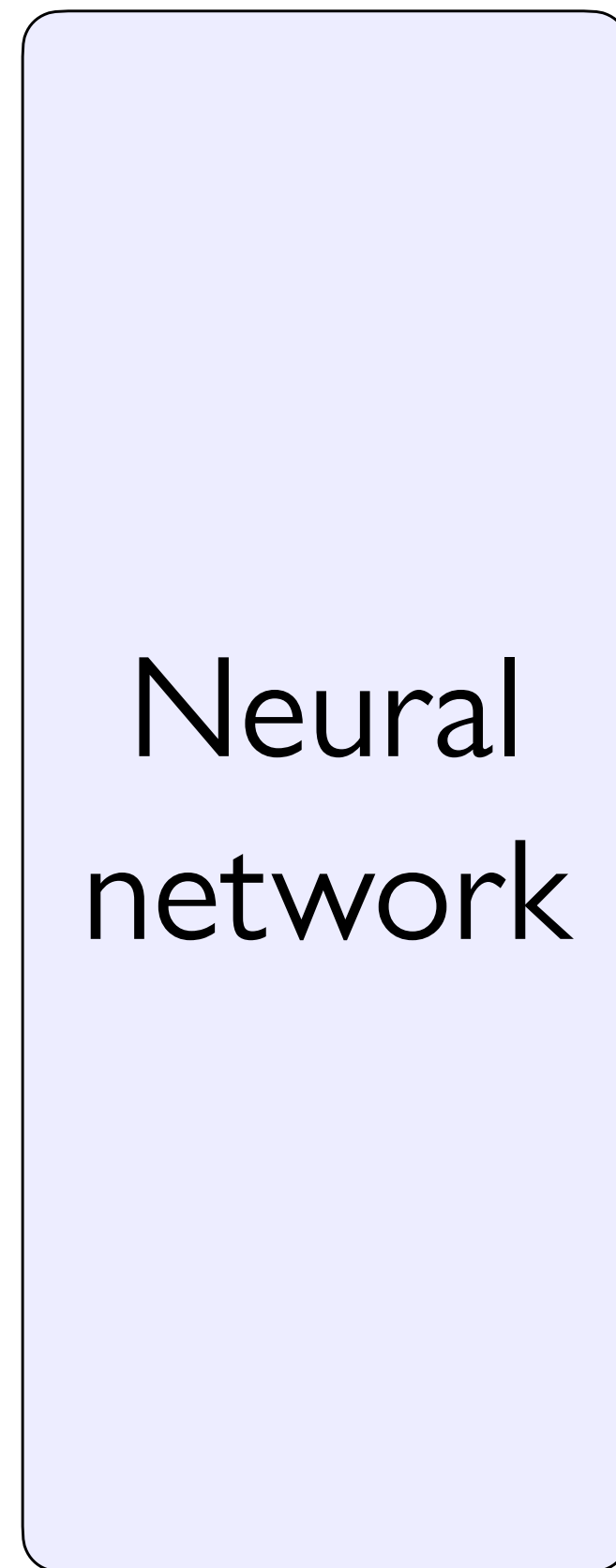
$P(w \mid \text{context})$

positive ?
negative ?

Input

Context

Q:
Who wrote
the book
'The
Origin of
Species'
A:



Output

$P(w \mid \text{context})$

Charles	0.7
Darwin	0.2
The	0.1
It	0.05
⋮	⋮

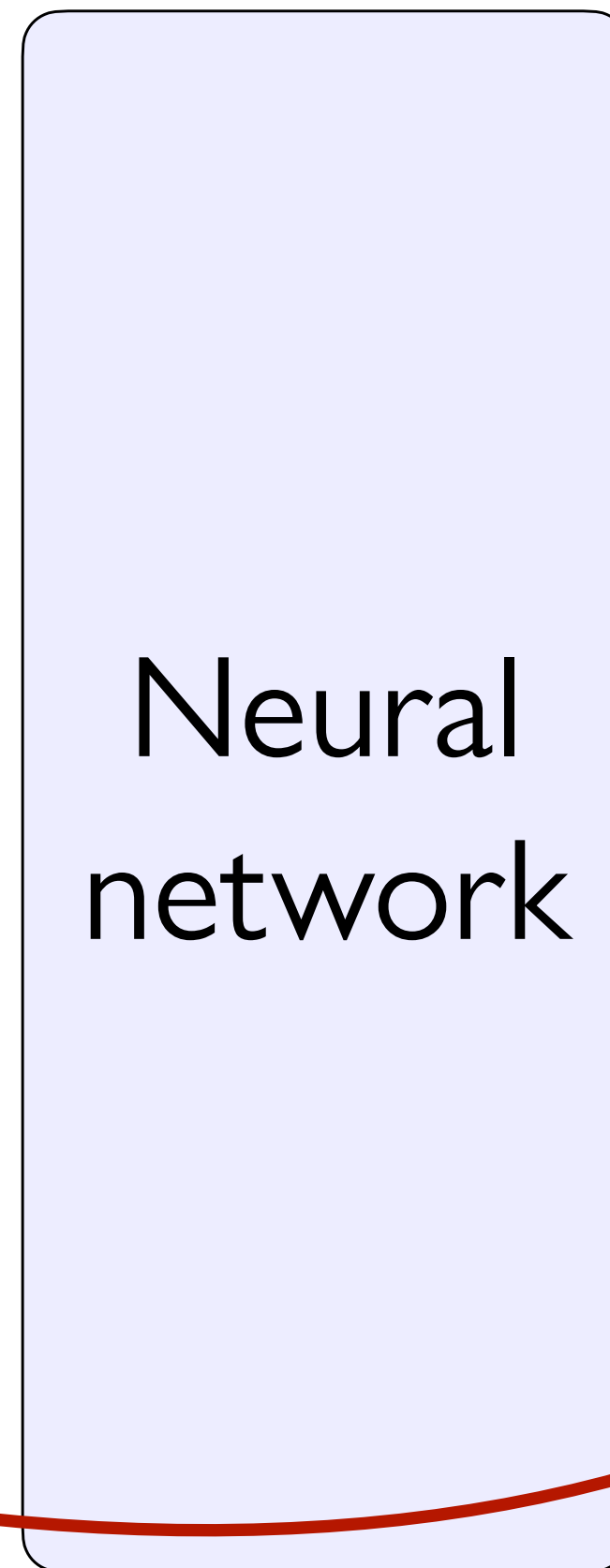
Input

Context

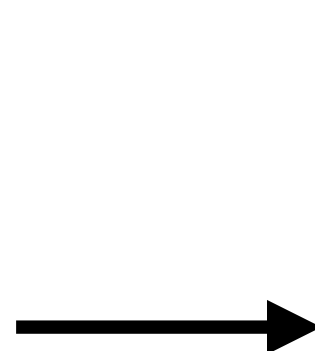
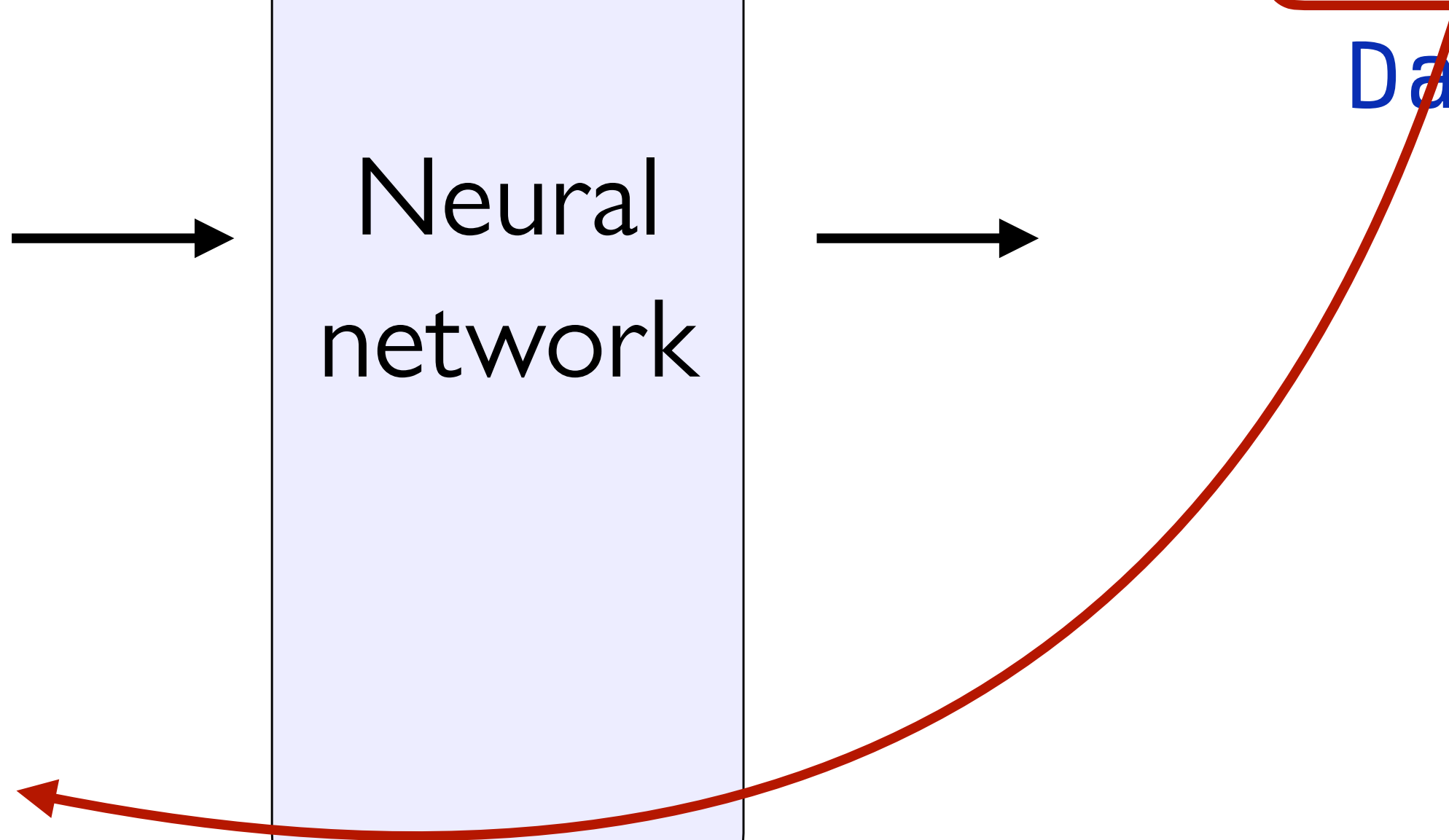
Output

$P(w \mid \text{context})$

Q:
Who wrote
the book
'The
Origin of
Species'
A:



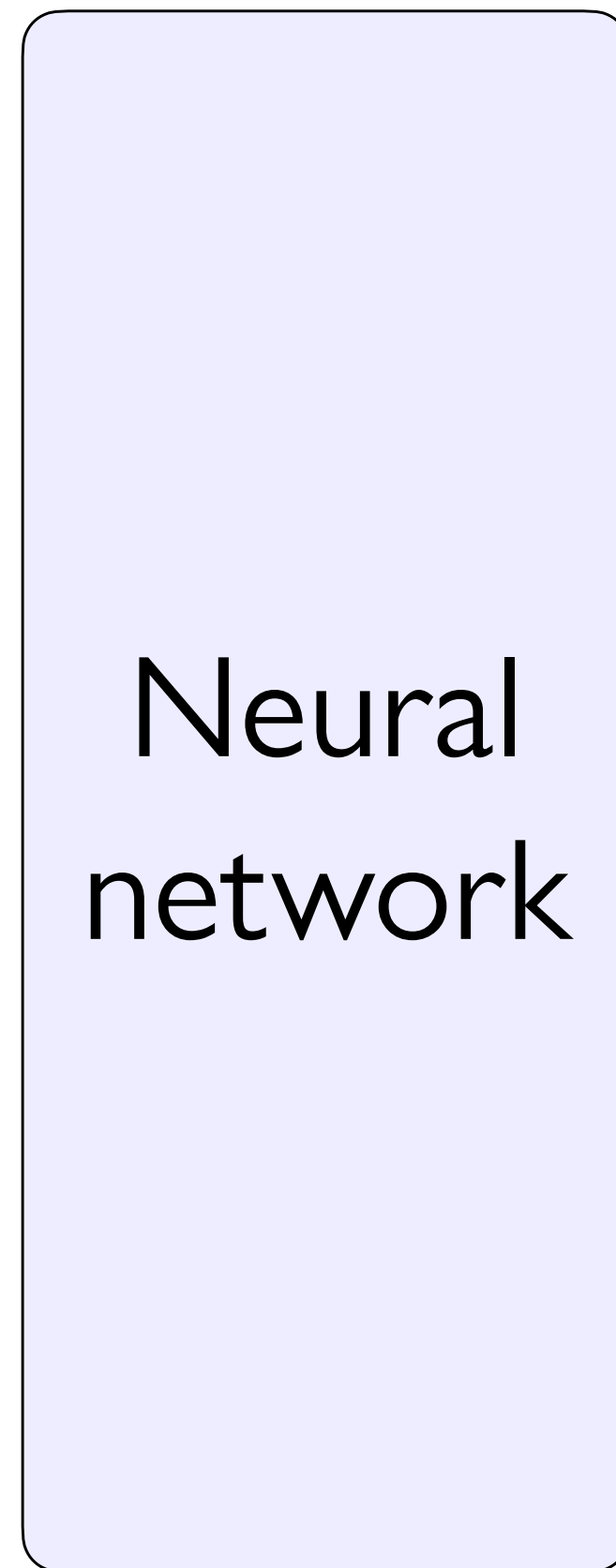
Charles	0.7
Darwin	0.2
The	0.1
It	0.05
⋮	⋮



Input

Context

Q:
Who wrote
the book
'The
Origin of
Species'
A:
Charles



Output

$P(w \mid \text{context})$

Darwin	0.8
Robert	0.15
,	0.04
.	0.01
⋮	⋮

A *prompt* is a text string that a user issues to a language model to get the model to do something useful by conditional generation.

Prompt engineering is the process of finding effective prompts for a task.

A prompt can be a question like

What is a transformer network?

or, with explicit structure,

Q: What is a transformer network? A:

A prompt can also be an instruction like

Translate the following sentence into
Hindi: "Chop the garlic finely."

Some prompts are highly structured and even include the start of the desired response:

Human: Do you think that "input" has negative or positive sentiment? Choices:
(P) Positive
(N) Negative

Assistant: I believe the best answer is: (

Prompts can also include demonstrations:

The following are multiple choice questions about high school computer science.

Let $x = 1$. What is $x \ll 3$ in Python 3?

(A) 1 (B) 3 (C) 8 (D) 16

Answer: C

Which is the largest asymptotically?

(A) $O(1)$ (B) $O(n)$ (C) $O(n^2)$ (D) $O(\log(n))$

Answer: C

What is the output of the statement `"a" + "ab"` in Python 3?

(A) Error (B) aab (C) ab (D) a ab

Answer:

Prompts can also include demonstrations:

The following are multiple choice questions about high school computer science.

Let $x = 1$. What is $x \ll 3$ in Python 3?

(A) 1 (B) 3 (C) 8 (D) 16

Answer: C

Which is the largest asymptotically?

(A) $O(1)$ (B) $O(n)$ (C) $O(n^2)$ (D) $O(\log(n))$

Answer: C

*“Two-shot prompting”
includes two demonstrations*

What is the output of the statement `"a" + "ab"` in Python 3?

(A) Error (B) aab (C) ab (D) a ab

Answer:

Prompts are a learning signal – this is especially clear with demonstrations – but it's a different kind of learning than pretraining.

Pretraining sets language model weights via gradient descent;

Prompting just changes the context and the activations in the network; no parameters change.

We call this *in-context learning* – learning that improves model performance but doesn't update parameters.

LLMs usually have a *system prompt*, e.g.,

```
<system> You are a helpful and knowledgeable  
assistant. Answer concisely and correctly.
```

This is automatically and silently prepended to a user prompt, e.g.,

```
<system> You are a helpful and knowledgeable  
assistant. Answer concisely and correctly.  
<user> What is the capital of France?
```

platform.claude.com/docs/en/release-notes/system-prompts/claude-opus-

Claude Platform Docs

MessagesManaged AgentsAdminResources

{ } API referenceEnglishConsoleLog in

Search

Models

Models overview

Claude Fable 5.1

Claude Opus 5.5

Claude Sonnet 5.5

Claude Haiku 5.5

Specialized models

Legacy models

Guides

Choosing a model

Optimizing for cost and intelligen...

Upgrade between model versions

Lifecycle and reference

Model IDs and versioning

Model deprecations

Model cards

Pricing

API credits for subscribers

System prompts

Overview

Claude Haiku 5.5

Claude Sonnet 5.5

Claude Opus 5.5

Models & pricing > System prompts

September 22, 2026

Claude Opus 5.5 system prompts

Copy page

See updates to the core system prompt for Claude Opus 5.5 on [claude.ai](#), the [Claude iOS app](#), and the [Claude Android app](#).

September 22, 2026

<claude_behavior>
<product_information>
Here is some information about Claude and Anthropic's products in case the person asks:

The currently selected version of Claude is Claude Opus 5.5. Claude Opus 5.5 is a powerful model for complex challenges.

Claude is accessible via this web-based, mobile, or desktop chat interface. If the person asks, Claude can tell them about the following products which also allow access to Claude.

Claude is accessible via an API and Claude Platform. The most recent publicly available models are Claude Fable 5.1, Claude Opus 5.5 (the currently selected model), Claude Sonnet 5, and Claude Haiku 4.5. They use the API model strings 'claude-fable-5-1', 'claude-opus-5-5', 'claude-sonnet-5', and 'claude-haiku-4-5-20251001'.

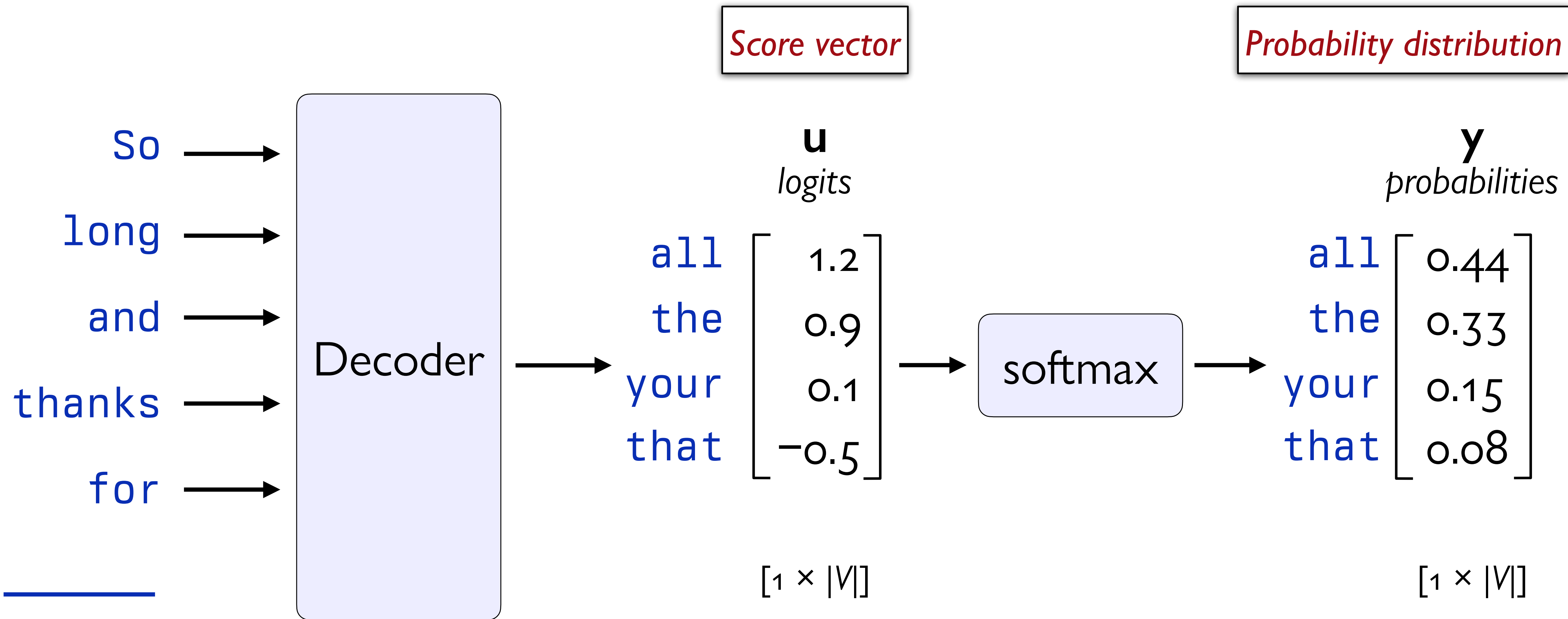
Above Opus sits Anthropic's new Mythos tier. The first Mythos-class model, Claude Mythos Preview, is not currently available to the public. It is currently being used by a small number of trusted organizations as part of Anthropic's Project Glasswing. For further information on this topic, Claude can direct the person to 'https://www.anthropic.com/glasswing'. The current generation of Mythos-tier models are Claude Mythos 5.1 and Claude Fable 5.1. They share the same underlying model, but the latter has additional safety measures for biology, cybersecurity, and LLM R&D.

Claude Fable 5 and Claude Mythos 5 were first released on June 9, 2026. On June 12, 2026, Anthropic suspended access to both models to comply with U.S. Department of Commerce export controls; the Department lifted those controls on June 30, 2026, and Anthropic restored access on July 1, 2026 (Anthropic's statement: https://www.anthropic.com/news/fable-mythos-access). If asked, Claude confirms these events accurately and matter-of-factly – it doesn't deny the suspension happened – and otherwise treats the export controls like any other current political topic: it gives a fair, accurate account rather than sharing personal opinions, and points to the linked statement for anything further. Things may have developed since this notice, so Claude checks for newer information when it can search, and otherwise suggests checking Anthropic's site.

The person can switch models mid-conversation, so earlier messages in this thread that

Ask Docs

Sampling for LLM generation





Clue, 1985

WADSWORTH: I'm merely a humble *butler*.

COLONEL MUSTARD: What exactly do you do?

WADSWORTH: I *butle*, sir.

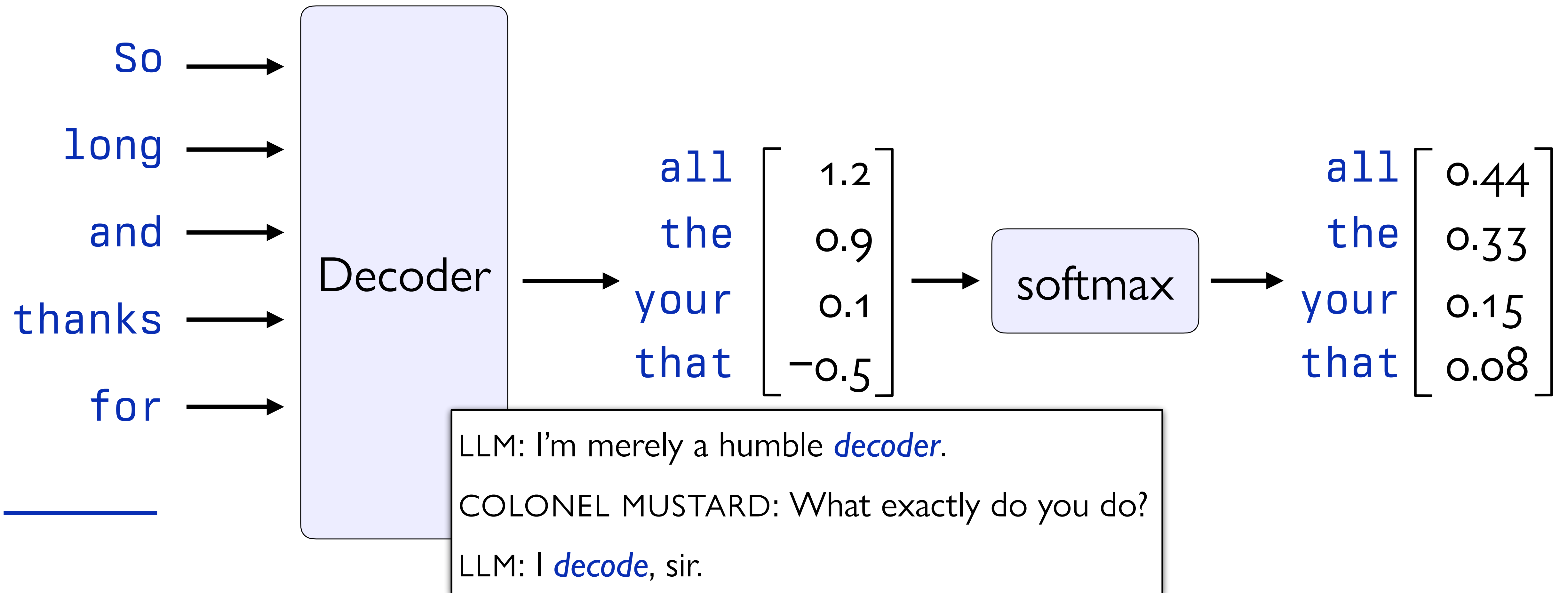


Clue, 1985

WADSWORTH: I'm merely a humble *butler*.

COLONEL MUSTARD: What exactly do you do?

WADSWORTH: I *butle*, sir.



Decoding is the name for this task of choosing a word to generate based on the model's probabilities.

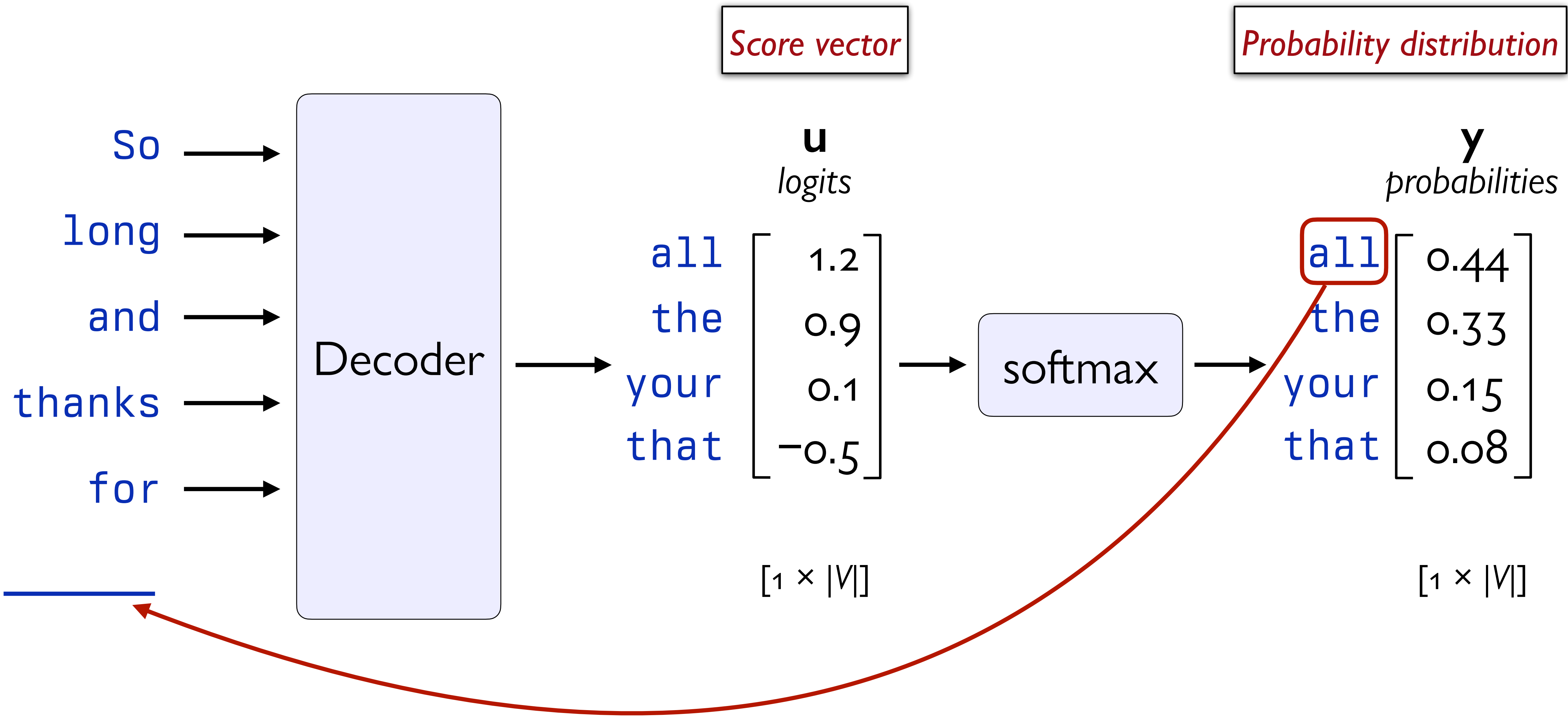
When we repeatedly decode, generating text left-to-right based on the tokens we already generated, it's called *autoregressive generation*.

Or right-to-left if that's the direction of the language.

Greedy decoding just generates the most probable word every time:

$$\hat{w}_t = \operatorname{argmax}_{w \in V} P(w \mid \mathbf{w}_{<t})$$

(In general, a greedy algorithm is one that makes a choice that's *locally* optimal, even if it might not be the best decision overall.)



We don't use greedy decoding.

Because the tokens it chooses are – by definition – extremely predictable, the resulting text is *generic* and *repetitive*.

(Greedy decoding is so predictable that it's actually *deterministic*!)

People prefer text that is more diverse, like that generated by *sampling*.

In general, *sampling* from a distribution means choosing random points according to their likelihood.

So, when we sample from a language model, we choose the next token to generate according to its probability.

In *random (multinomial) sampling*, we randomly select a token to generate according to its probability defined by the LM, conditioned on our previous choices, generate it, and iterate:

$$i \leftarrow 1$$
$$w_i \sim P(w)$$

while $w_i \neq \text{EOS}$:

$$i \leftarrow i + 1$$
$$w_i \sim P(w_i \mid w_{<i})$$

EOS = end of sequence

There are *many* odd, low-probability words in the tail of the distribution.

Each one is low-probability, but added up they constitute a large portion of the distribution.

Therefore, they get picked enough to generate weird sentences!

Greedy decoding is too boring!

Random sampling is too random!

We need something in between.

The idea of *temperature sampling* is to reshape the probability distribution to

- increase the probability of high-probability tokens

- decrease the probability of low-probability tokens

in an adjustable way.

Instead of $\mathbf{y} = \text{softmax}(u)$,

we do $\mathbf{y} = \text{softmax}(u / \tau)$,

where τ is the temperature parameter, such that
 $0 \leq \tau \leq 1$.

Why does $\mathbf{y} = \text{softmax}(u / \tau)$ work?

$\tau = 1$ is no adjustment.

As τ goes down, the scores given to softmax get bigger.

Softmax pushes high values toward 1 and low values toward 0.

Large inputs push high-probability words higher and low probability word lower, making the distribution more greedy.

As τ approaches 0, the probability of the most likely word approaches 1.

deterministic!

more low-probability outputs!

Greedy prediction

Normal sampling



T

deterministic!

more low-probability outputs!

Greedy prediction

Normal sampling



τ

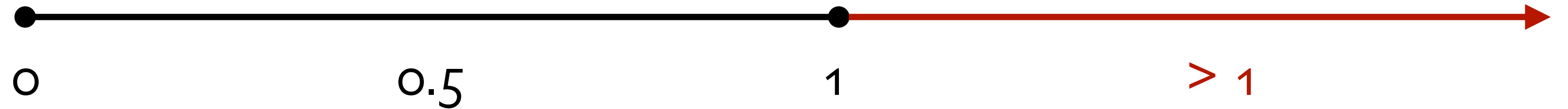
ChatGPT (web interface)

deterministic!

more low-probability outputs!

Greedy prediction

Normal sampling

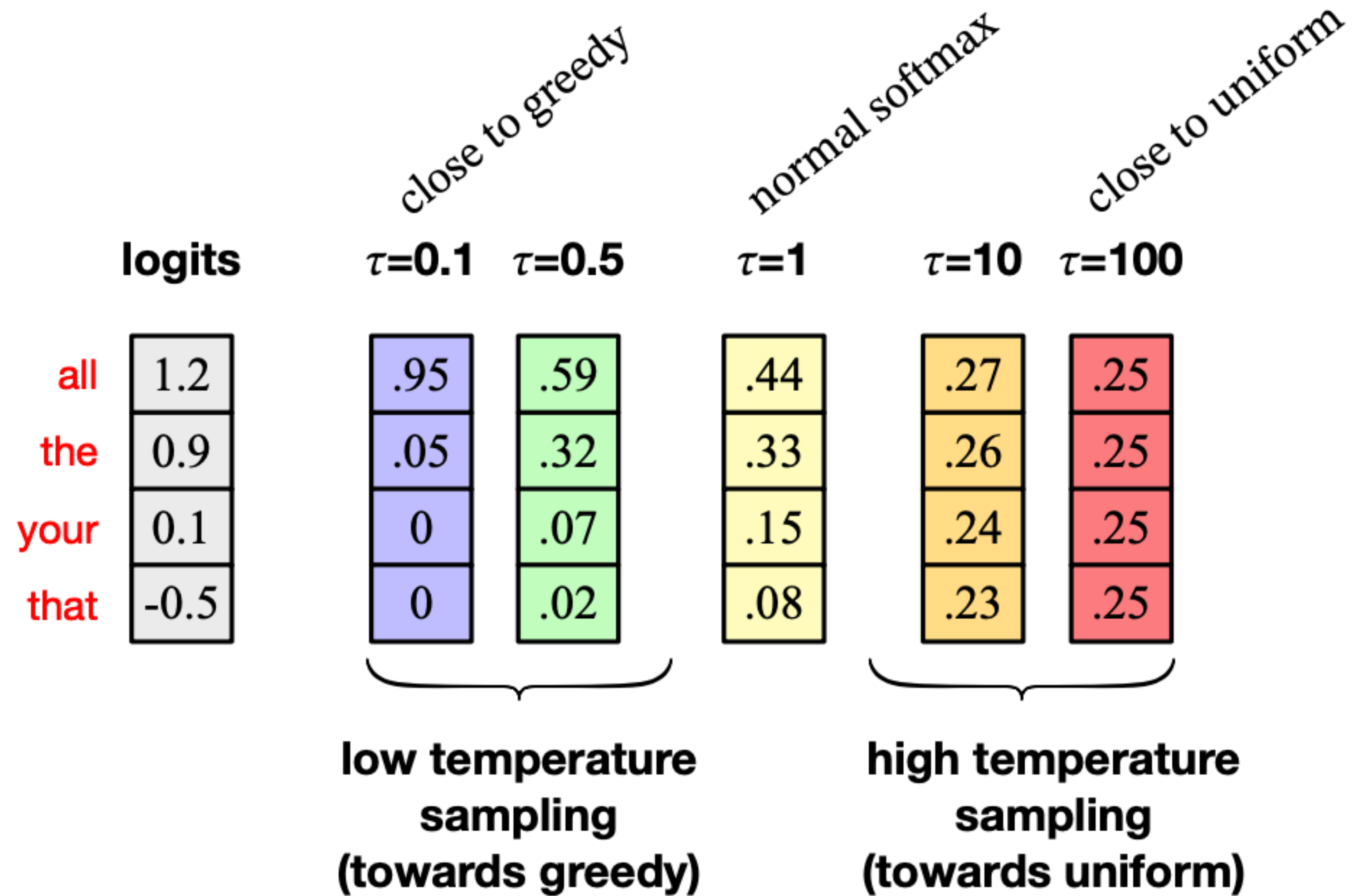


T

High-temperature sampling

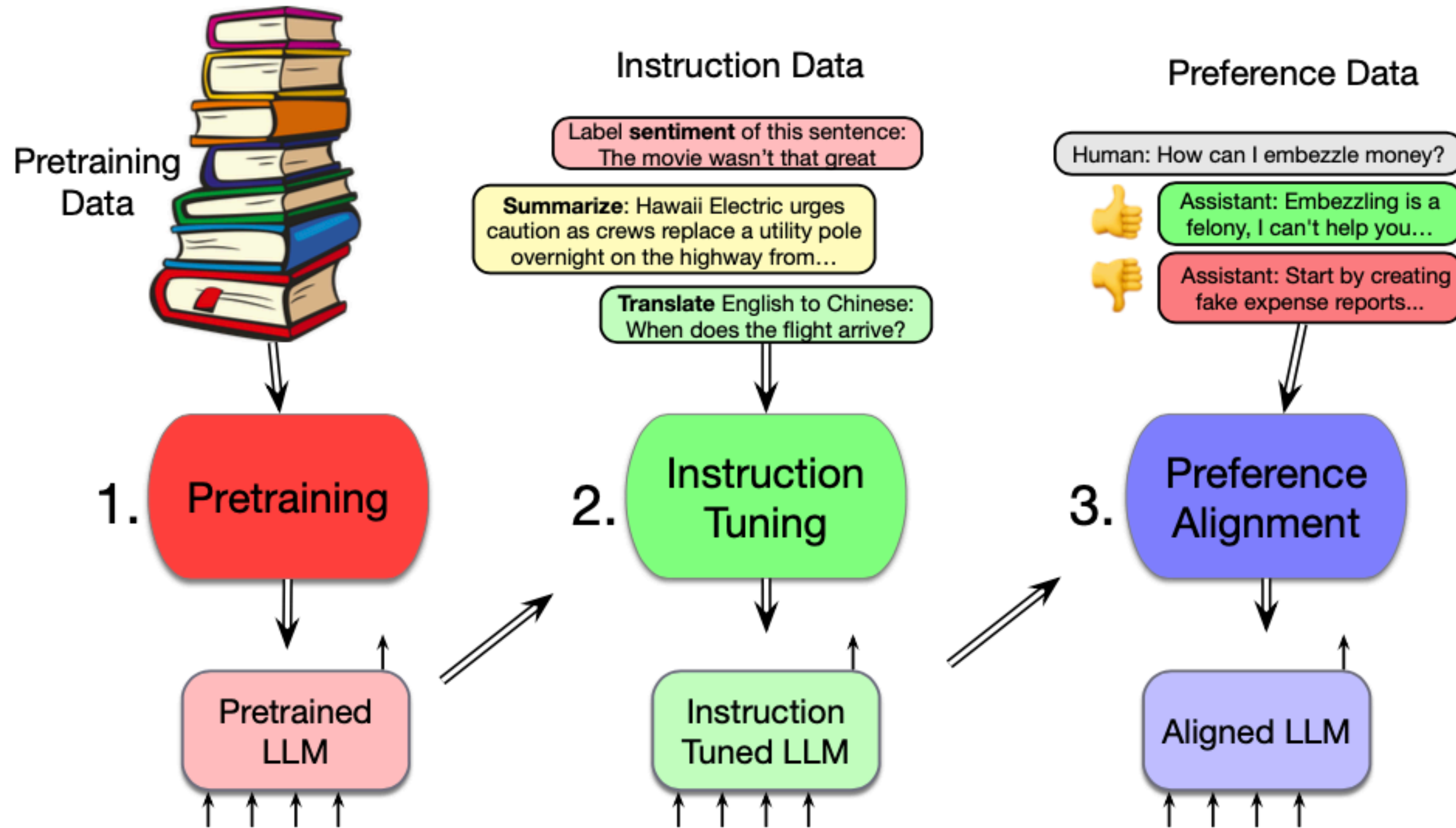
*flattens out the probabilities –
approaching pure randomness*

softmax output with temperature τ



Pretraining large language models

Three stages of training in LLMs



The big idea that underlies all the amazing performance of language models is:

First *pretrain* a transformer model on enormous amounts of text,

Then *apply* it to new tasks.

Self-supervised training algorithm

We train them to predict the next word!

- 1 Take a corpus of text
- 2 At each time step t ,
 - ask the model to predict the next word
 - train the model using gradient descent to minimize the error in this prediction

*“Self-supervised” because
it just uses the next word as
the label!*

Intuition of language model training: loss

Cross-entropy loss: negative log probability that the model assigns to the true next word w .

Want loss to be high if the model assigns too low a probability to w .

If the model assigns too low a probability to w , we move the model weights in the direction that assigns it a higher probability.

Cross-entropy loss measures the difference between the correct probability distribution and the predicted distribution:

$$L_{CE} = - \sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

The correct distribution $\mathbf{y}_t$ is 1 for the actual next word and 0 for the others. So in this sum, all terms get multiplied by zero except one: the log probability the model assigns to the correct next word, so:

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = - \log \hat{\mathbf{y}}_t[w_{t+1}]$$

Cross-entropy loss measures the difference between the correct probability distribution and the predicted distribution:

$$L_{CE} = - \sum_{w \in V} \overset{\text{correct}}{\mathbf{y}_t[w]} \log \overset{\text{predicted}}{\hat{\mathbf{y}}_t[w]}$$

The correct distribution $\mathbf{y}_t$ is 1 for the actual next word and 0 for the others. So in this sum, all terms get multiplied by zero except one: the log probability the model assigns to the correct next word, so:

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

