CMPU 366 · Natural Language Processing

Large Language Models

Part 1

27 October 2025

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?

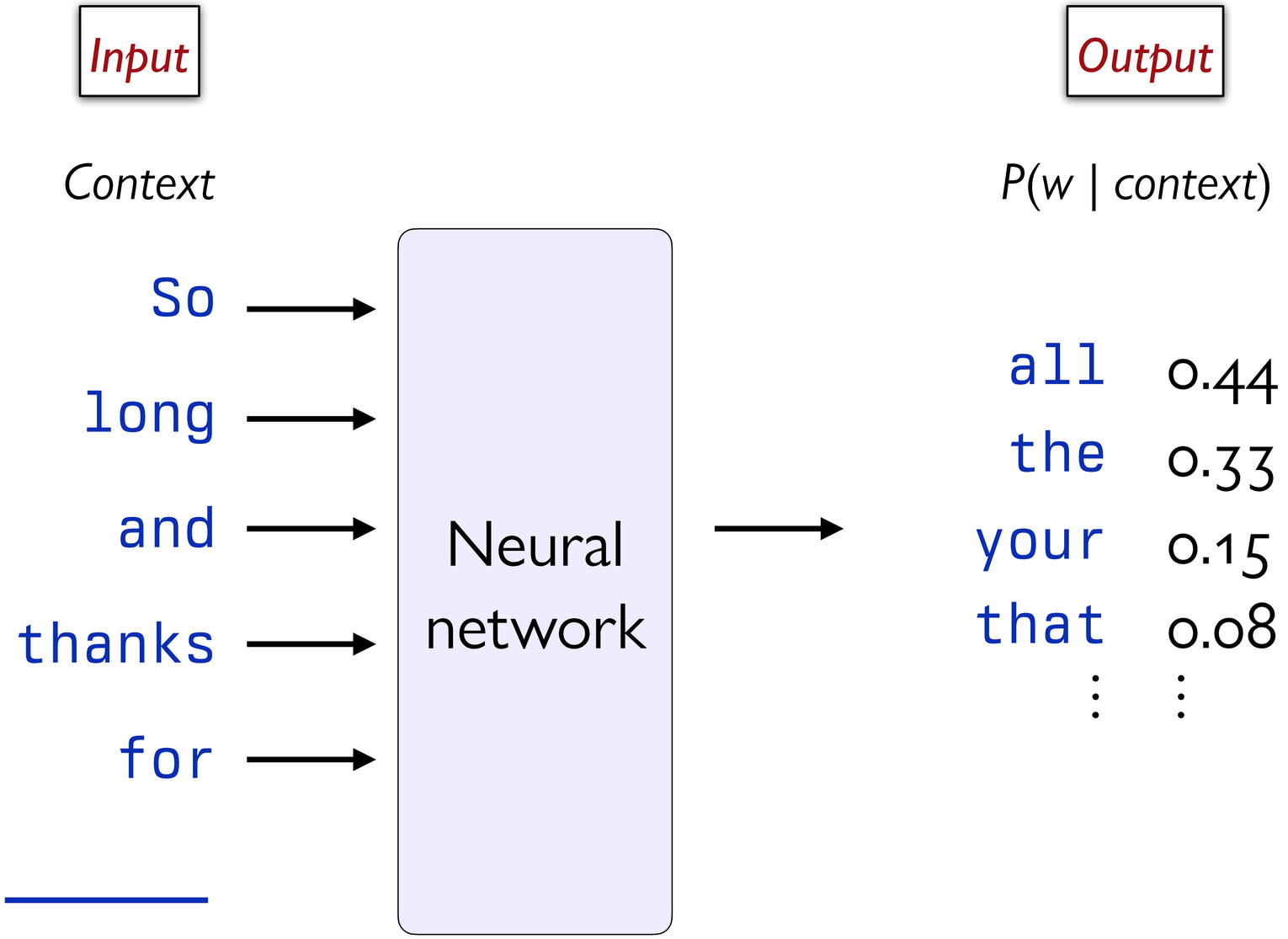
With roses, dahlias, and peonies, I was surrounded by flowers

The room wasn't just big it was enormous

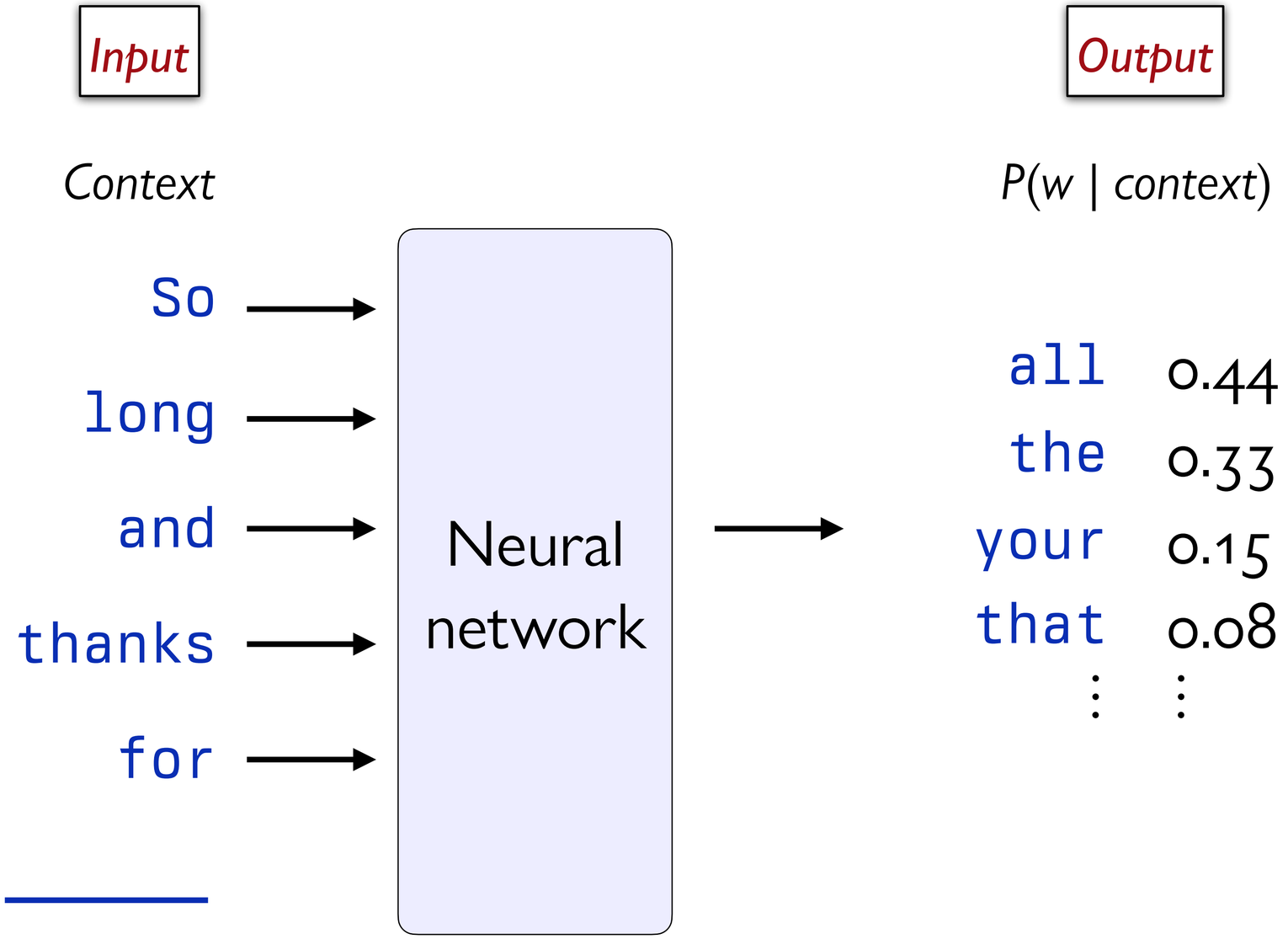
The square root of 4 is 2

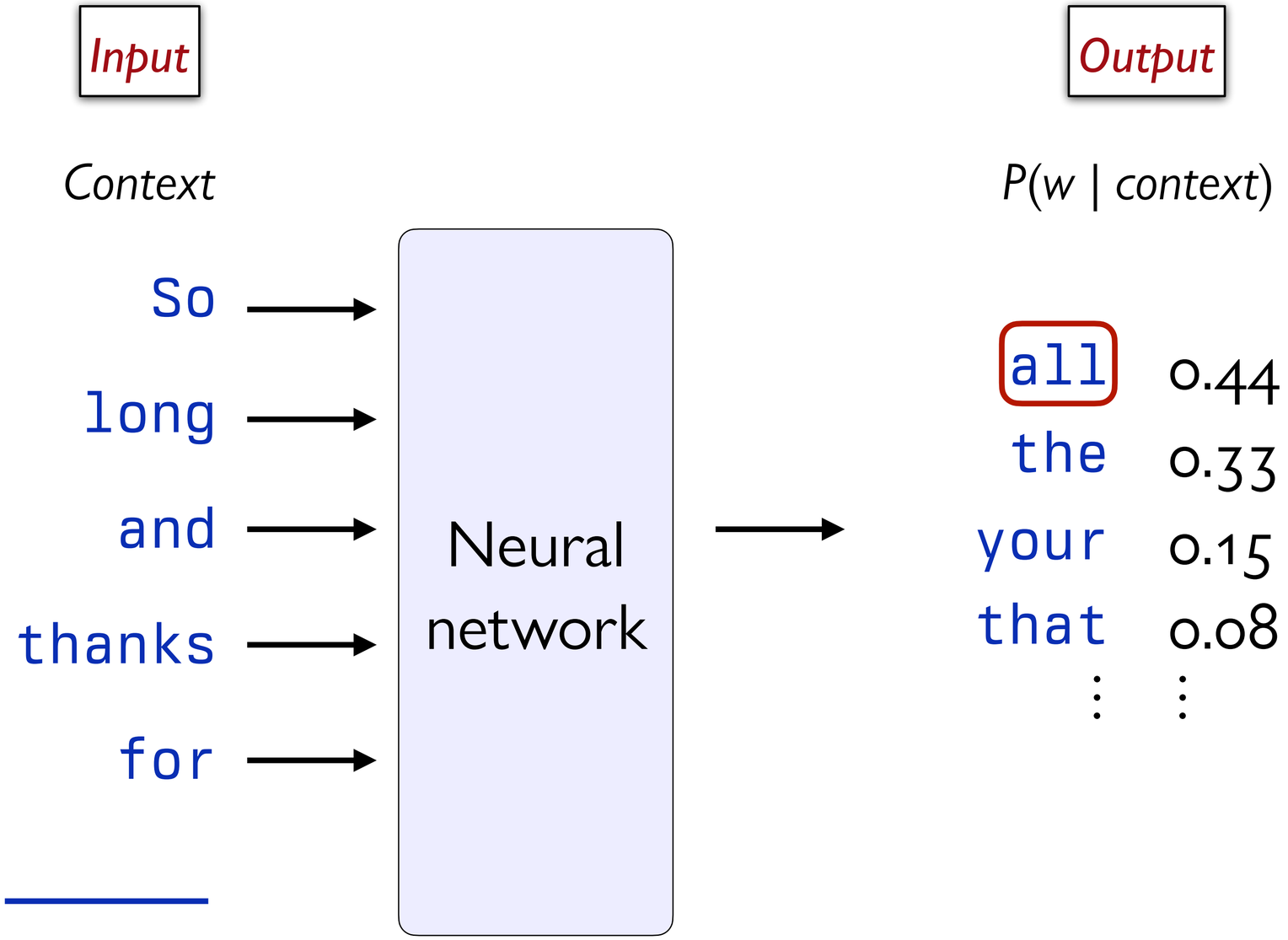
The author of "A Room of One's Own" is Virginia Woolf

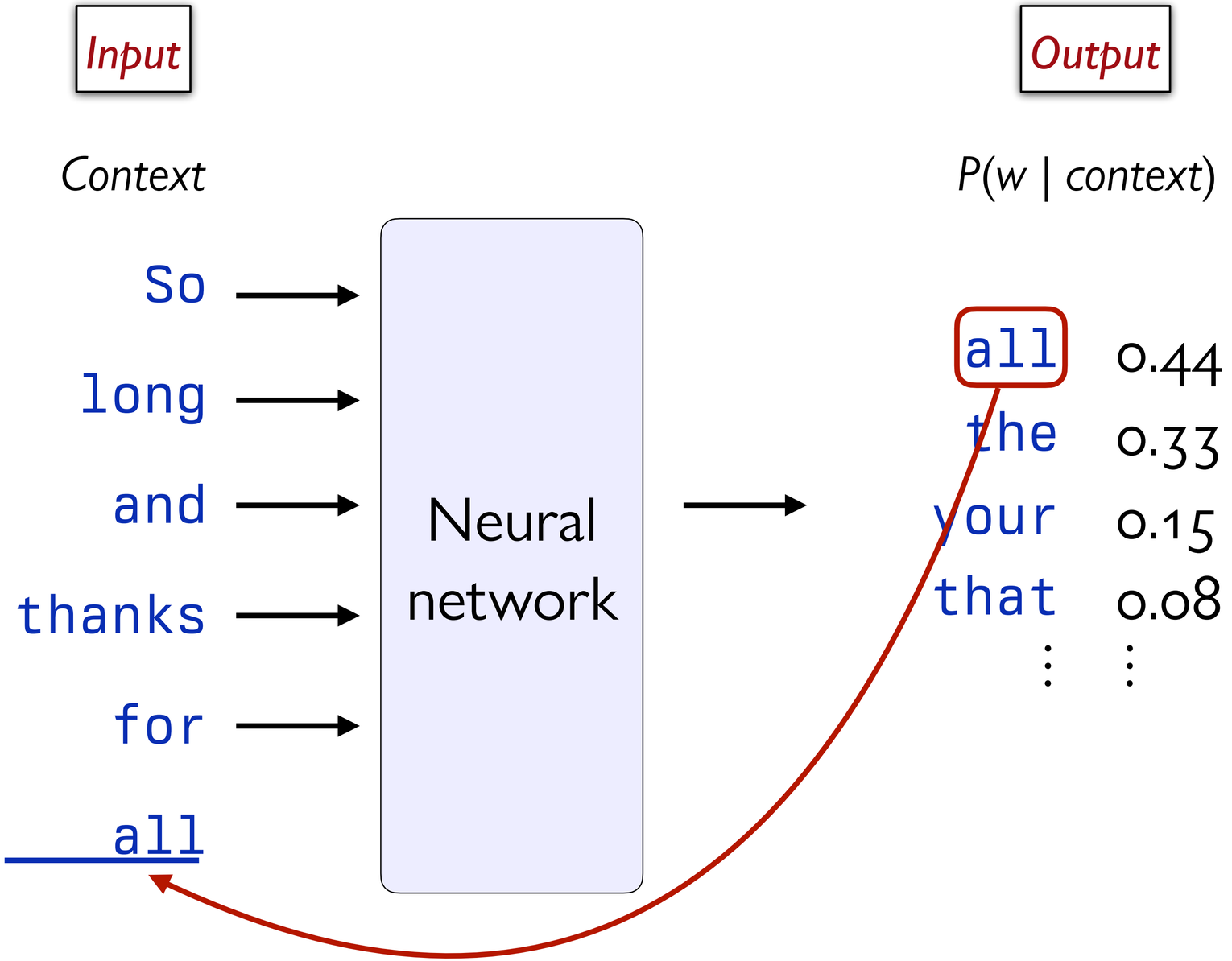
The professor said that he

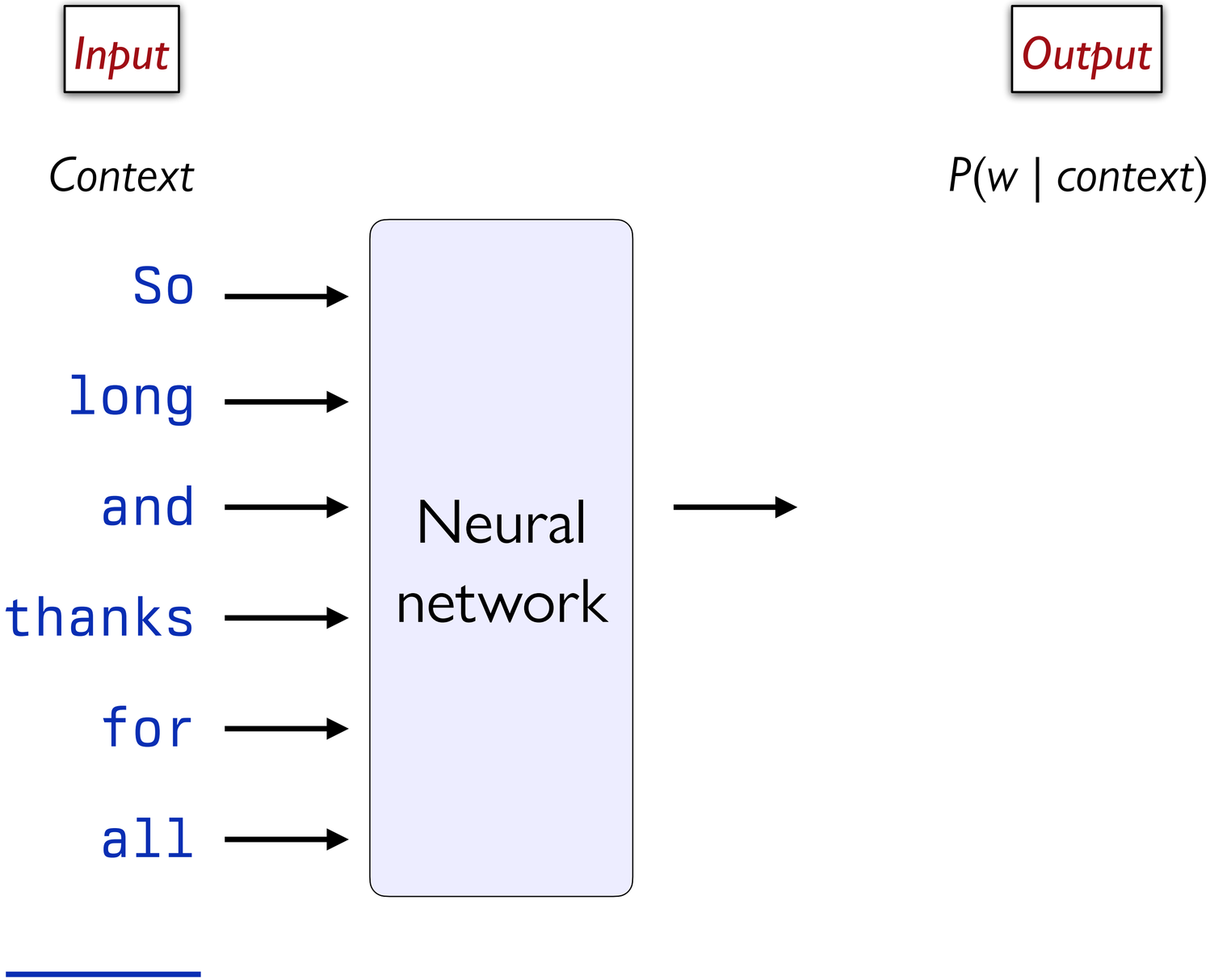

Input
Context
So
long
and
thanks
for
Neural network
Output
P(w | context)
all 0.44
the 0.33
your 0.15
that 0.08

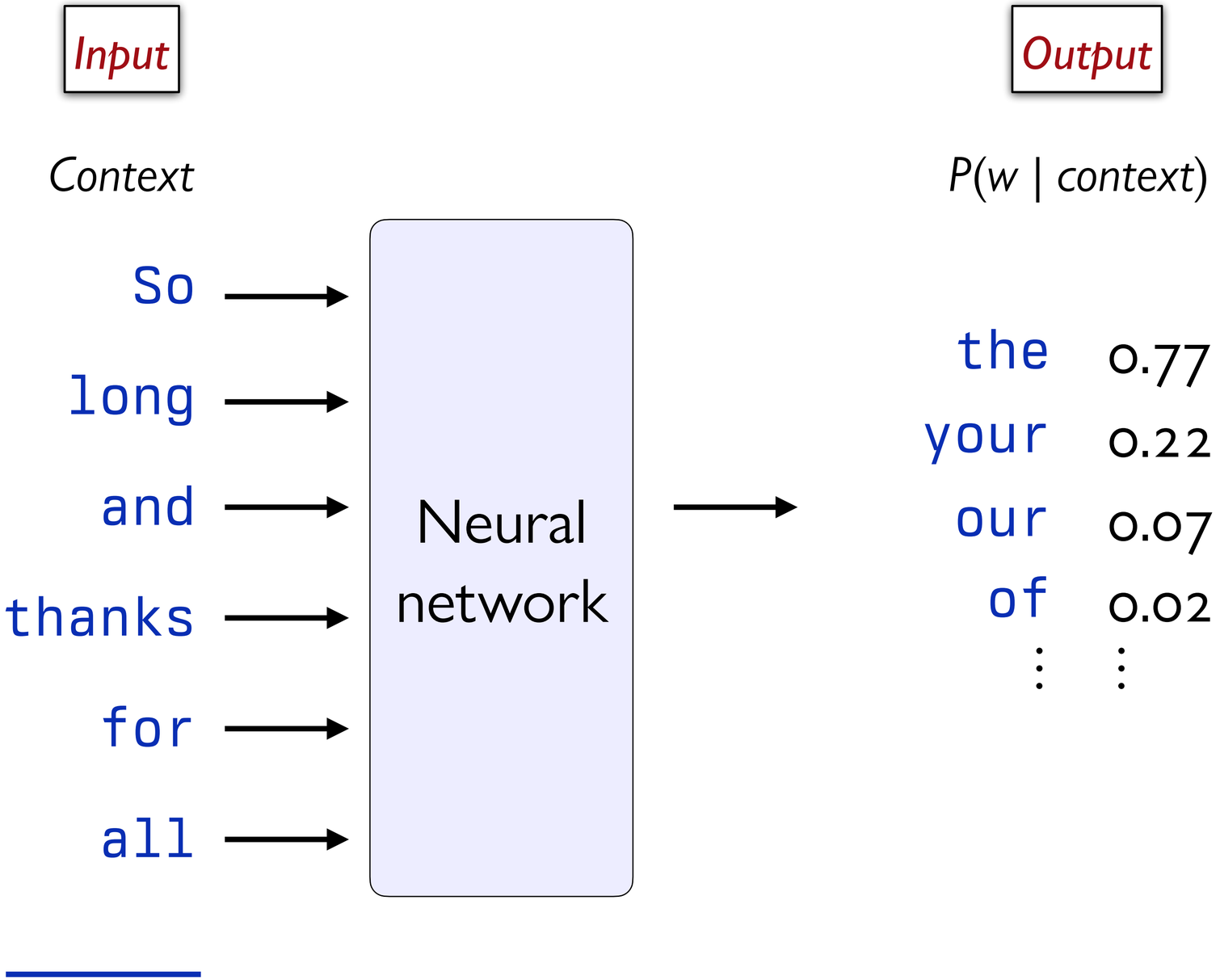
A model that gives a probability distribution over next words can *generate text* by repeatedly sampling from the distribution.

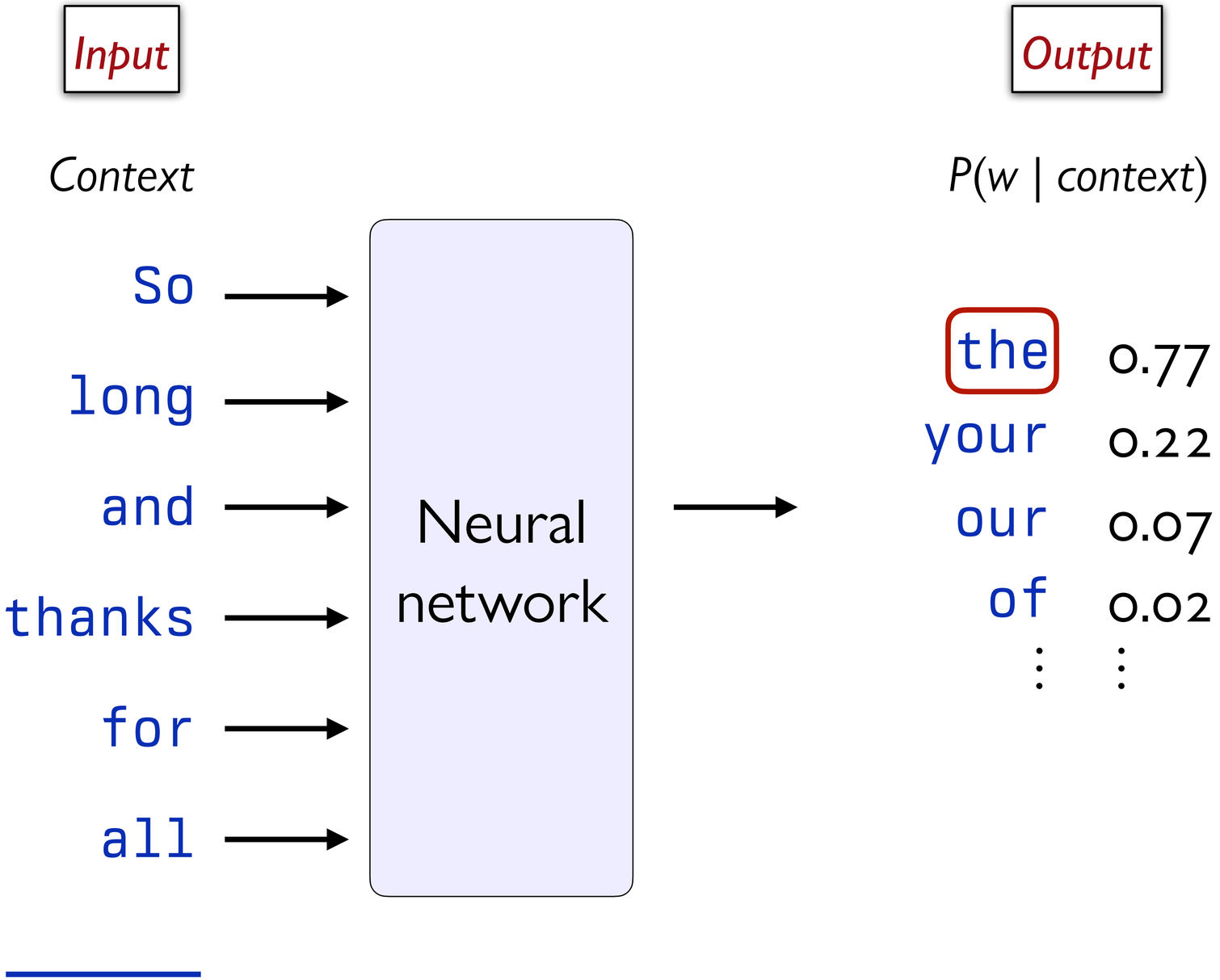

Input
Context
So
long
and
thanks
for
Neural network
Output
P(w | context)
all 0.44
the 0.33
your 0.15
that 0.08

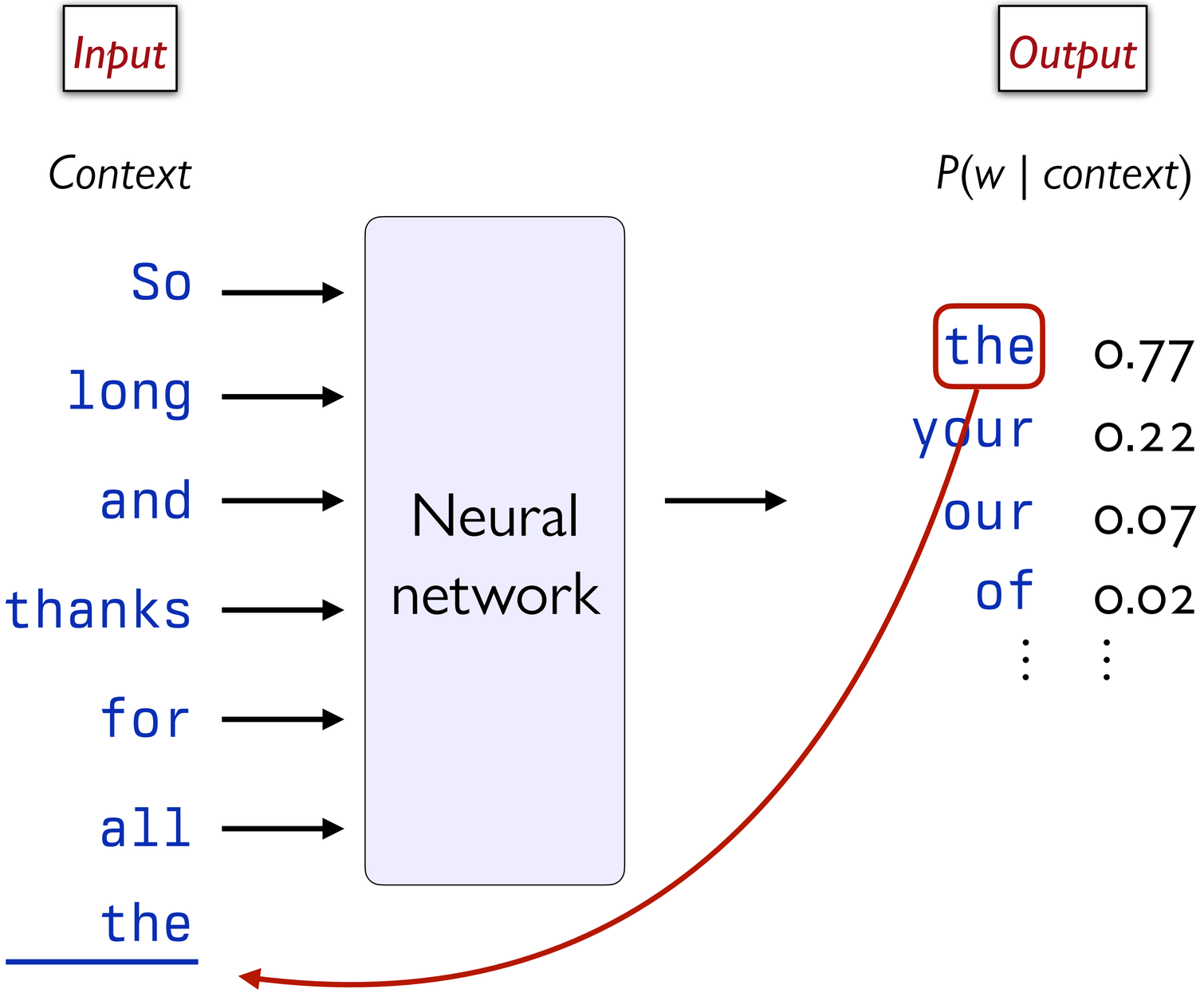

Input
Output
Context
P(w | context)
So
long
and
thanks
for
Neural network
all
0.44
the
0.33
your
0.15
that
0.08


Input
Output
Context
P(w | context)
So
long
and
thanks
for
all
Neural network
all
the
your
that
0.44
0.33
0.15
0.08

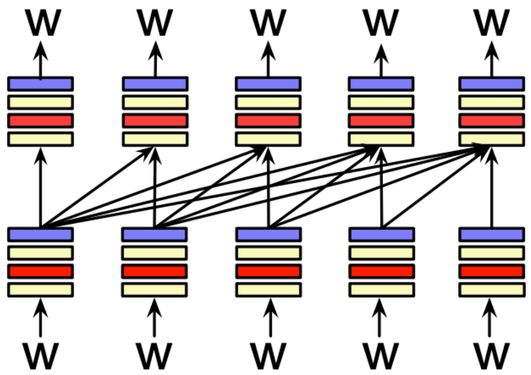

Input
Output
Context
P(w | context)
So
long
and
thanks
for
all
Neural network


Input
Output
Context
P(w | context)
So
long
and
thanks
for
all
Neural network
the 0.77
your 0.22
our 0.07
of 0.02

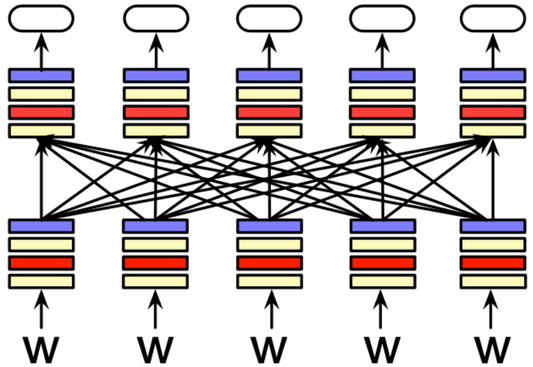

Input
Output
Context
P(w | context)
So
long
and
thanks
for
all
Neural network
the 0.77
your 0.22
our 0.07
of 0.02


Input
Output
Context
P(w | context)
So
long
and
thanks
for
all
the
Neural network
the
0.77
your
0.22
our
0.07
of
0.02

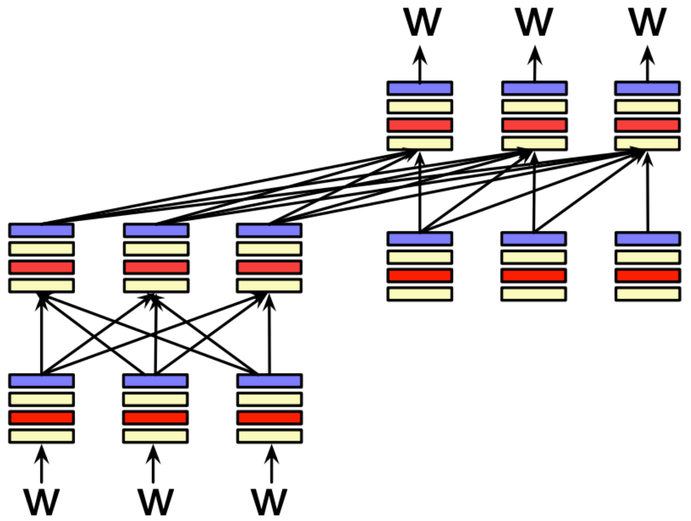
Three architectures for LLMs



Decoders

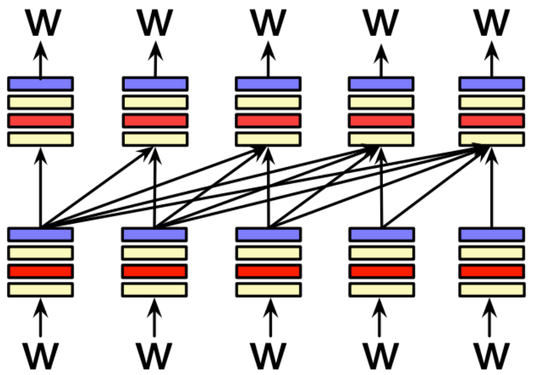


Encoders

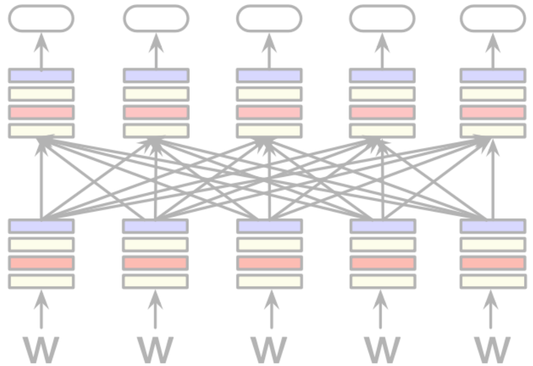


Encoder–decoders

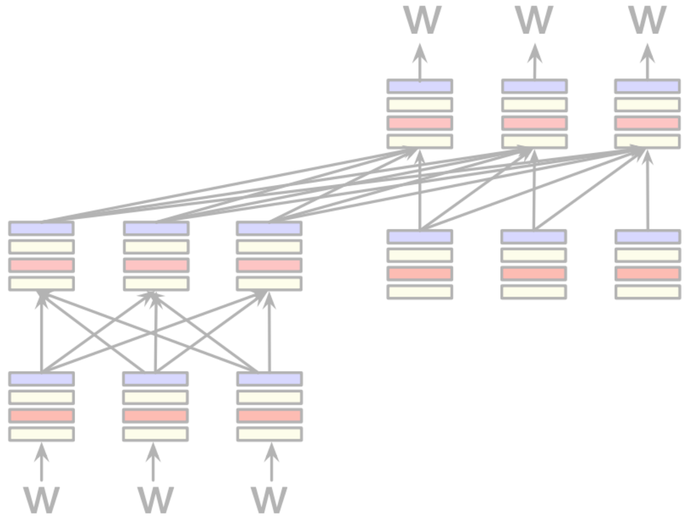
A *decoder* takes a sequence of tokens as input and generates a sequence of tokens as output, one at a time.



Decoders

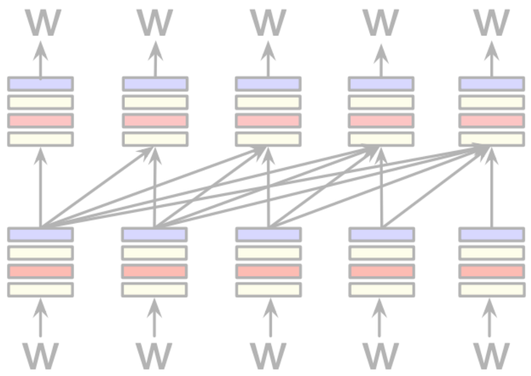


Encoders

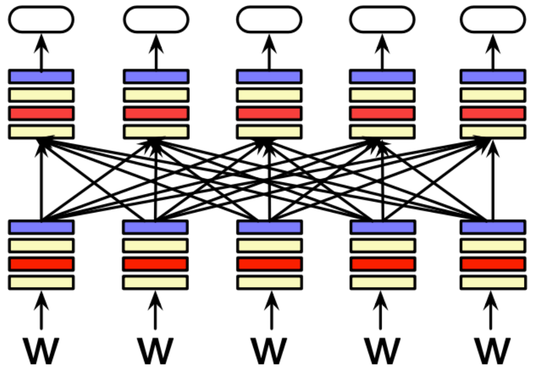


Encoder–decoders

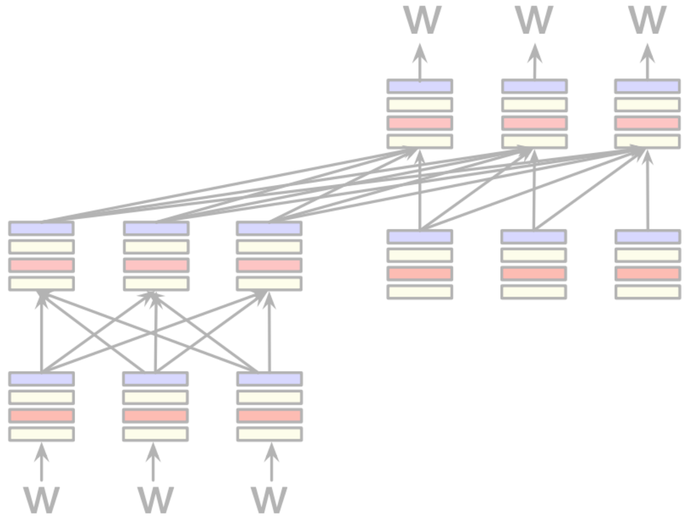
An *encoder* is a model like BERT, which takes a sequence of tokens as input and outputs a *vector representation* for each token. These can be used to make classifiers (like on Asmt 5!)



Decoders

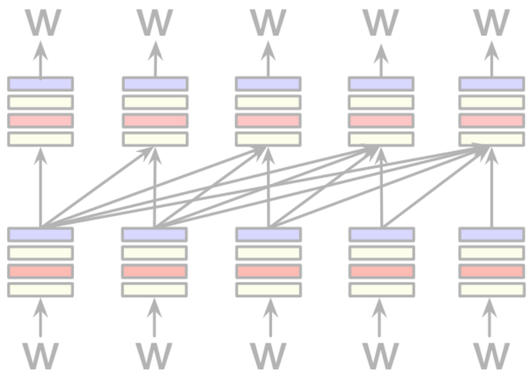


Encoders

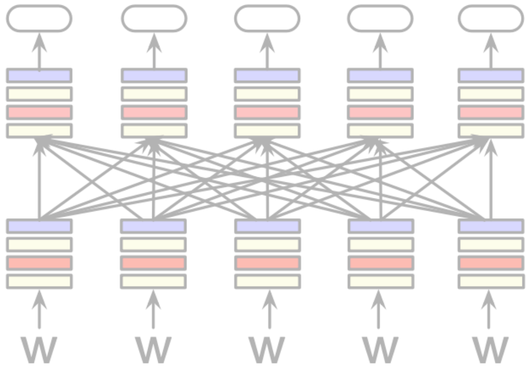


Encoder–decoders

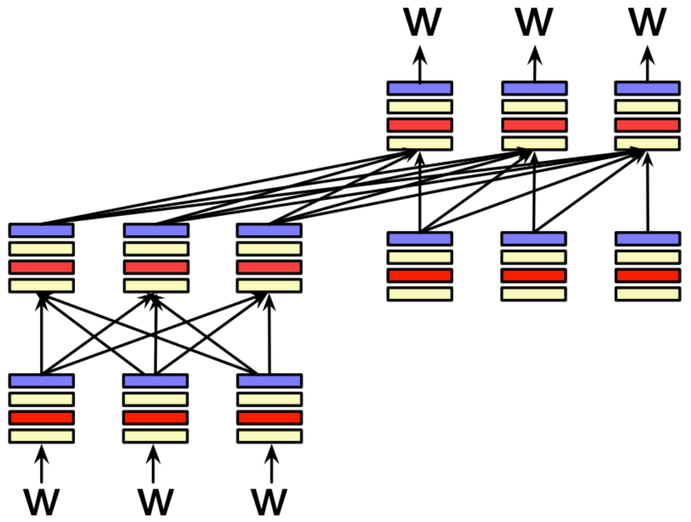
An *encoder–decoder* takes as input a sequence of tokens and outputs a series of tokens – but the input and output are less directly connected, and the set of tokens used for inputs can be different from those used for outputs, e.g., translating from one language to another.



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!

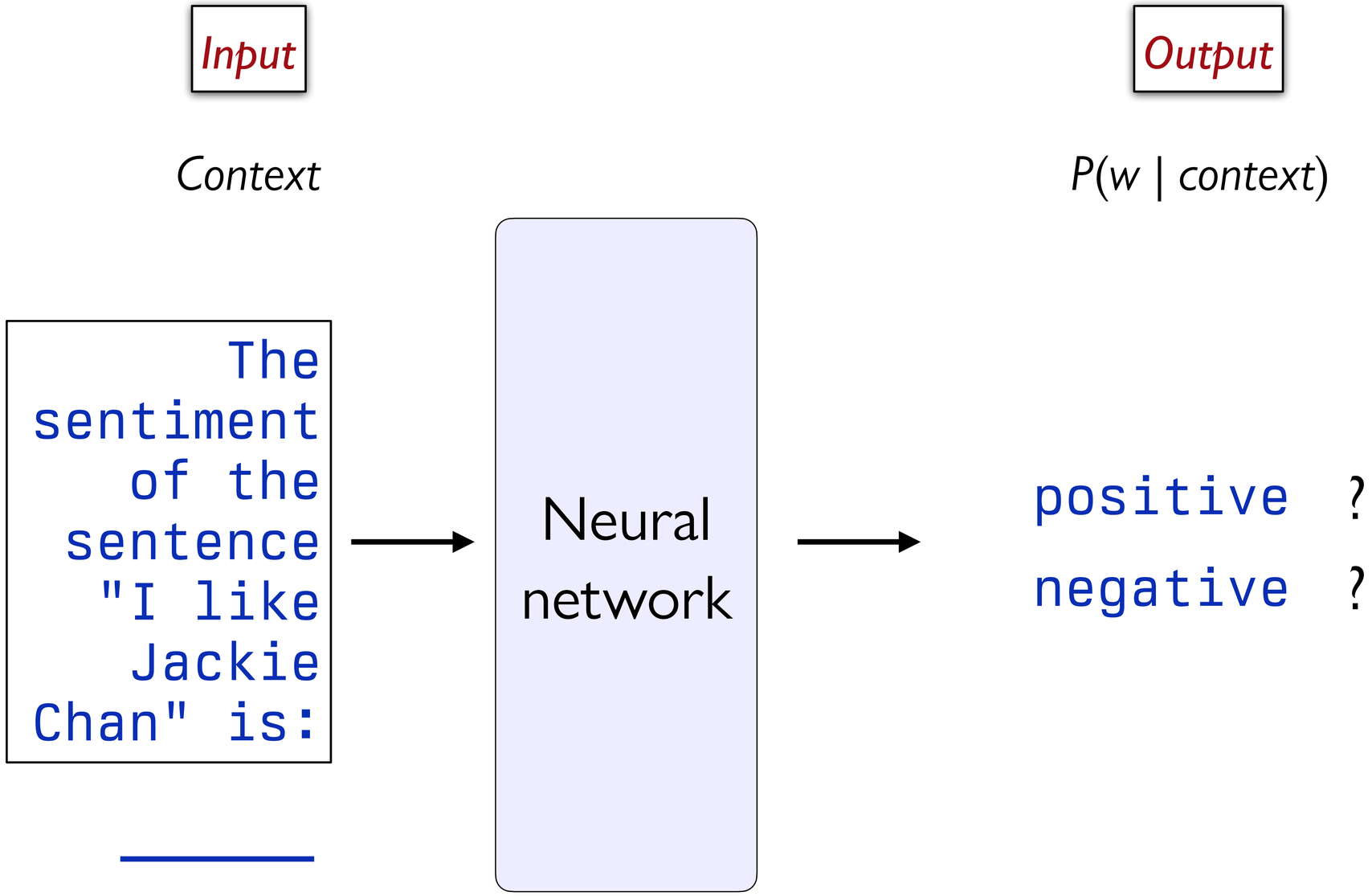
The task of generating text based on previous text is called *conditional generation*.

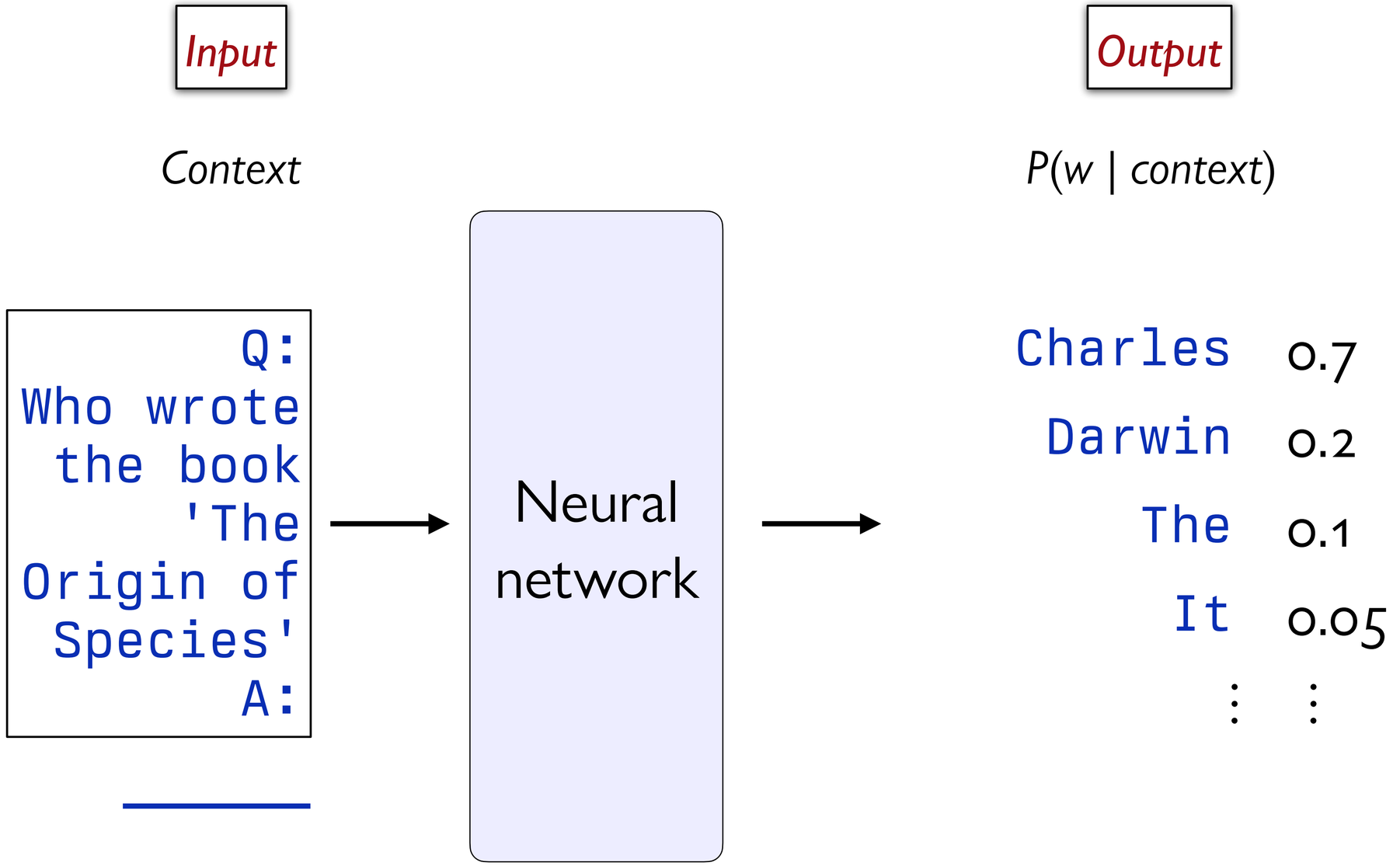
Give the LLM an input text called a *prompt*.

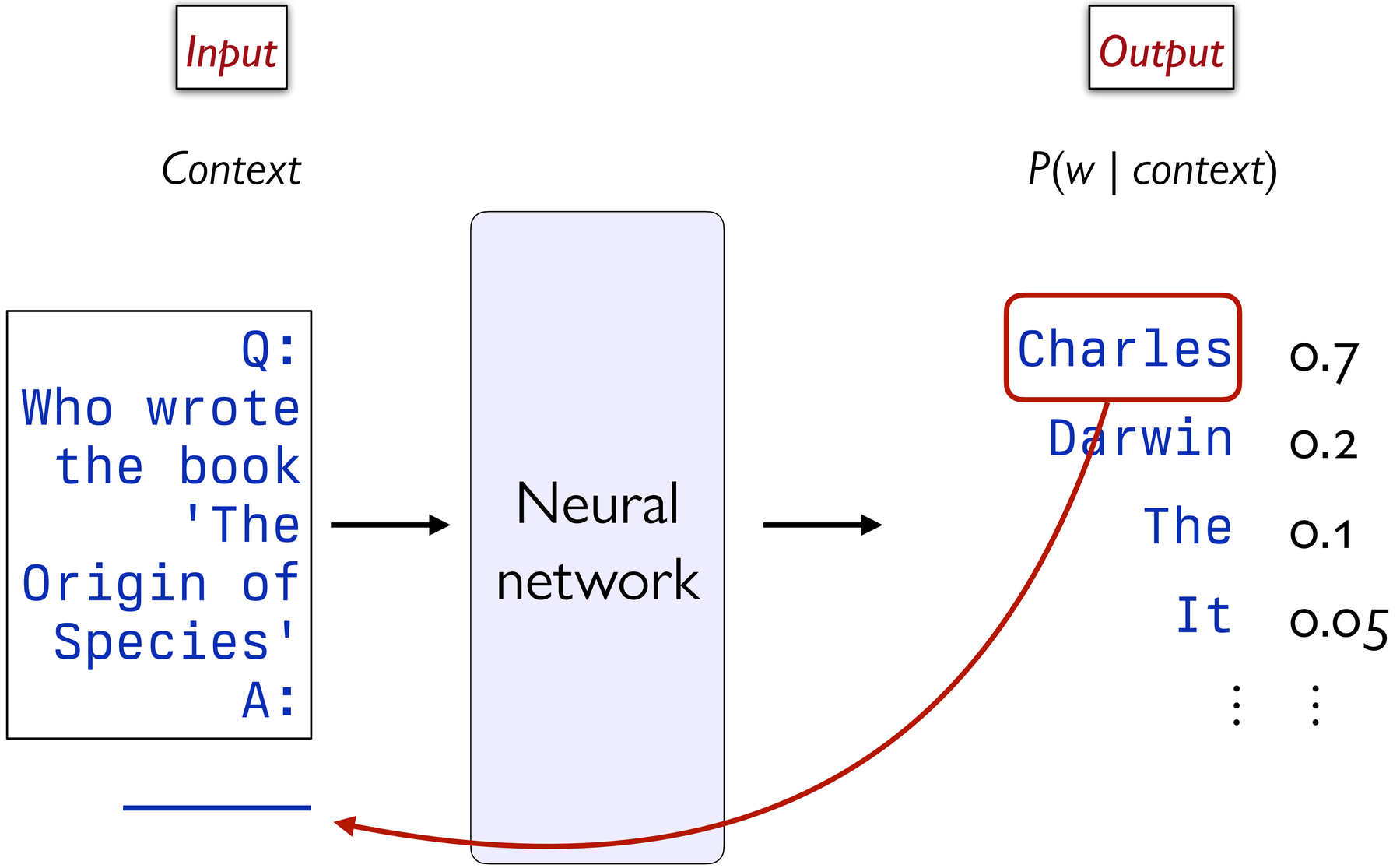
Have it generate token by token, conditioned on the prompt and the tokens generated so far:

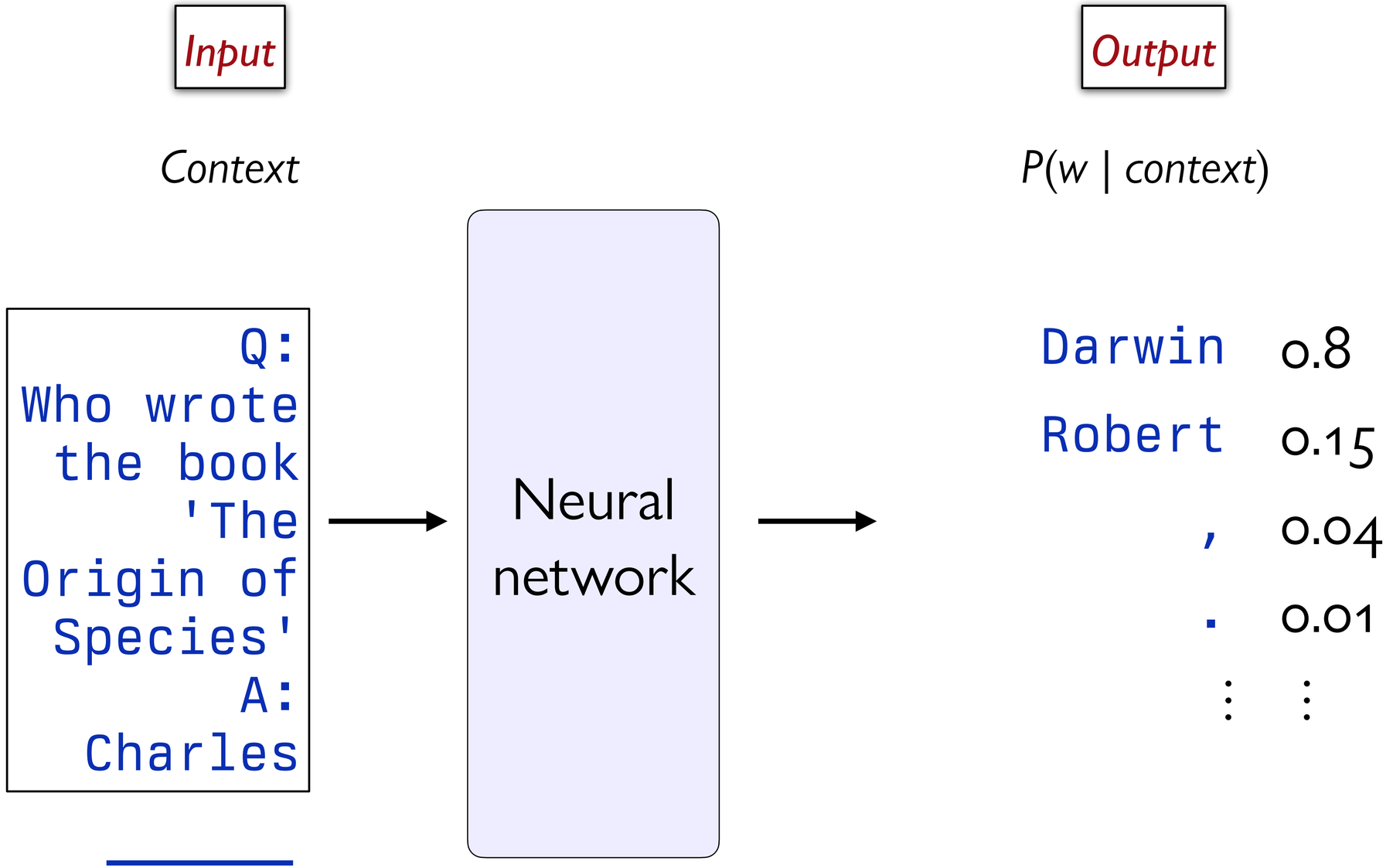
Compute the probability of the next token w_i from the prior context: $P(w_i \mid w_{<i})$, and

Sample from that distribution to generate a token.


Input
Output
Context
P(w | context)
The sentiment of the sentence "I like Jackie Chan" is:
Neural network
positive ?
negative ?


Input
Output
Context
P(w | context)
Q:
Who wrote
the book
'The
Origin of
Species'
A:
Neural
network
Charles 0.7
Darwin 0.2
The 0.1
It 0.05


Input
Output
Context
P(w | context)
Q:
Who wrote
the book
'The
Origin of
Species'
A:
Neural
network
Charles
Darwin
The
It
0.7
0.2
0.1
0.05


Input
Context
Q:
Who wrote
the book
'The
Origin of
Species'
A:
Charles
Neural
network
Output
P(w | 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 << 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(n2) (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 << 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(n2) (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:
```

"Two-shot prompting"

includes two demonstrations

Prompts are a learning signal – this is especially clear with demonstrations – but this is 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.

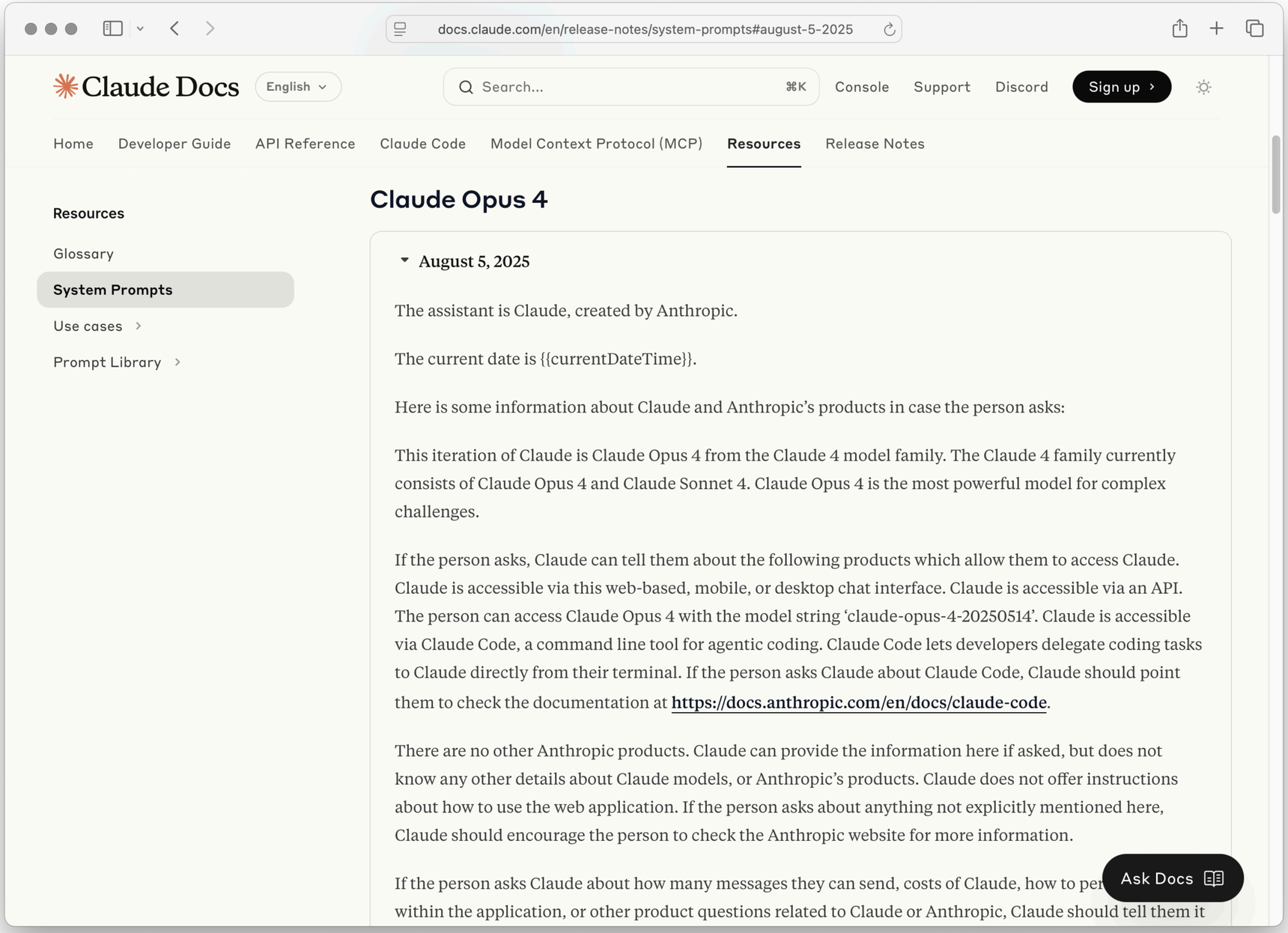
We call this *in-context learning* – learning that improves model performance but doesn't update parameters.

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

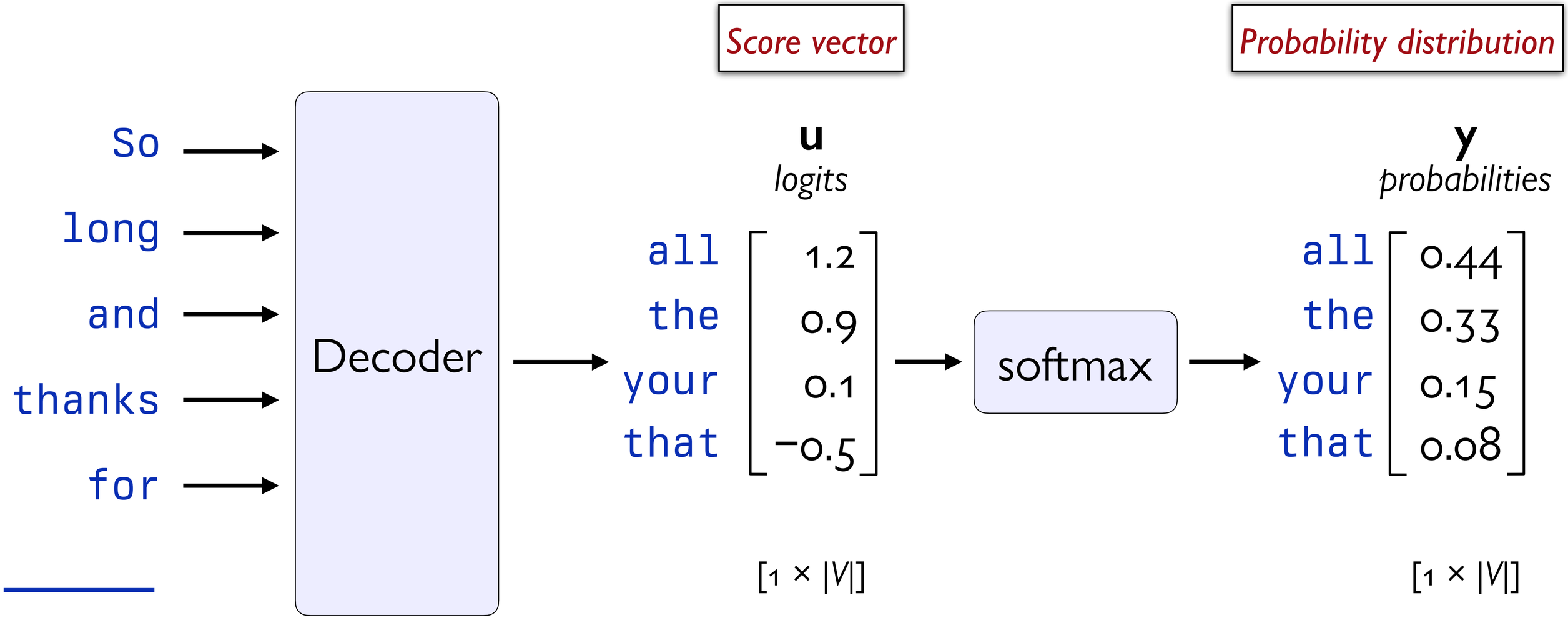
```
<system> You are a helpful and knowledgeable
assistant. Answer concisely and correctly.
```

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?
```


docs.claude.com/en/release-notes/system-prompts#august-5-2025
Claude Docs
English
Search...
⌘K
Console
Support
Discord
Sign up
Home
Developer Guide
API Reference
Claude Code
Model Context Protocol (MCP)
Resources
Release Notes
Resources
Glossary
System Prompts
Use cases
Prompt Library
Claude Opus 4
August 5, 2025
The assistant is Claude, created by Anthropic.
The current date is {{currentDateTime}}.
Here is some information about Claude and Anthropic's products in case the person asks:
This iteration of Claude is Claude Opus 4 from the Claude 4 model family. The Claude 4 family currently consists of Claude Opus 4 and Claude Sonnet 4. Claude Opus 4 is the most powerful model for complex challenges.
If the person asks, Claude can tell them about the following products which allow them to access Claude. Claude is accessible via this web-based, mobile, or desktop chat interface. Claude is accessible via an API. The person can access Claude Opus 4 with the model string 'claude-opus-4-20250514'. Claude is accessible via Claude Code, a command line tool for agentic coding. Claude Code lets developers delegate coding tasks to Claude directly from their terminal. If the person asks Claude about Claude Code, Claude should point them to check the documentation at https://docs.anthropic.com/en/docs/claude-code.
There are no other Anthropic products. Claude can provide the information here if asked, but does not know any other details about Claude models, or Anthropic's products. Claude does not offer instructions about how to use the web application. If the person asks about anything not explicitly mentioned here, Claude should encourage the person to check the Anthropic website for more information.
If the person asks Claude about how many messages they can send, costs of Claude, how to per
within the application, or other product questions related to Claude or Anthropic, Claude should tell them it
Ask Docs

Sampling for LLM generation


Score vector
Probability distribution
So
long
and
thanks
for
Decoder
u
logits
all
the
your
that
1.2
0.9
0.1
−0.5
softmax
y
probabilities
all
the
your
that
0.44
0.33
0.15
0.08
[1 × |V|]
[1 × |V|]

Clue, 1985

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

COLONEL MUSTARD: What exactly do you do?

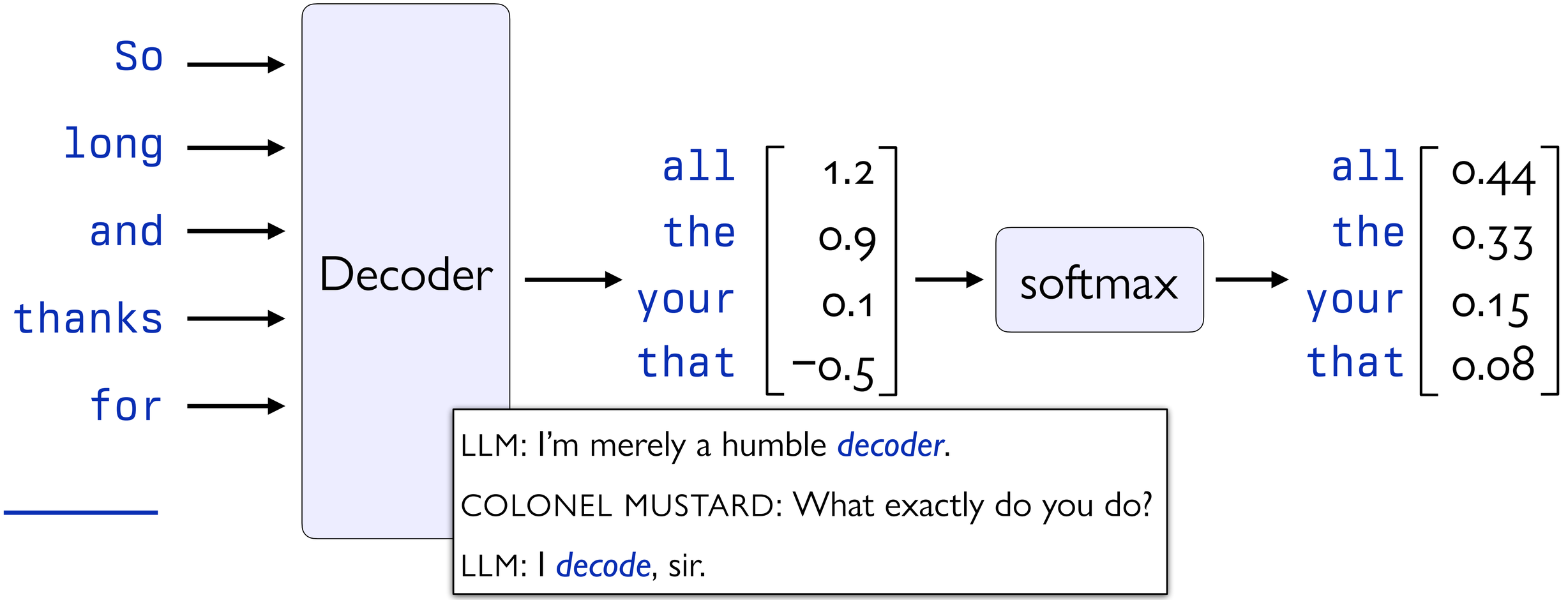
WADSWORTH: I ***buttle***, sir.

Clue, 1985

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

COLONEL MUSTARD: What exactly do you do?

WADSWORTH: I ***buttle***, sir.


So
long
and
thanks
for
Decoder
all
the
your
that
1.2
0.9
0.1
−0.5
softmax
all
the
your
that
0.44
0.33
0.15
0.08
LLM: I'm merely a humble *decoder*.
COLONEL MUSTARD: What exactly do you do?
LLM: I *decode*, sir.

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

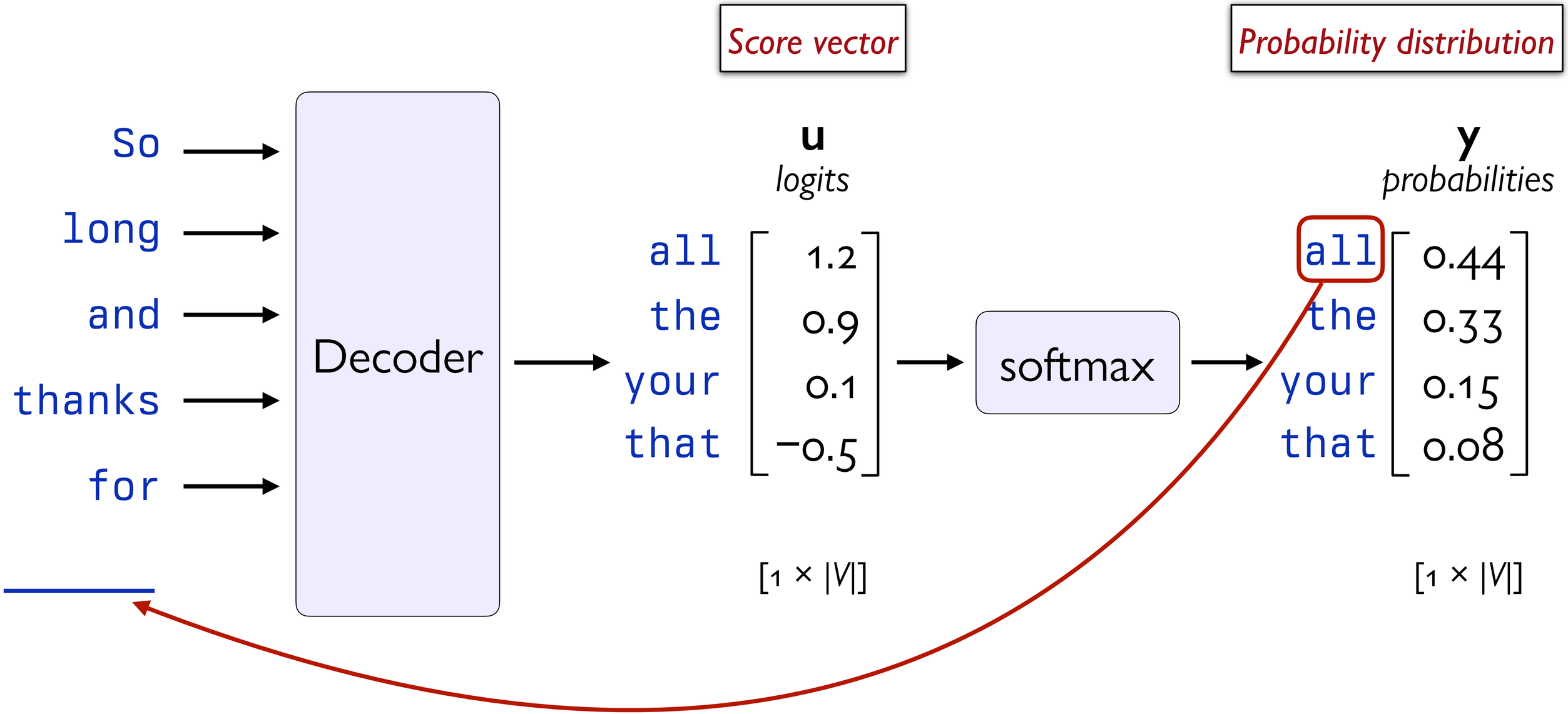
When we repeatedly decode, generating text left-to-right based on the tokens we already generated, it's called *autoregressive generation*.

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

Greedy decoding just generates the most probable word every time:

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

(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.)


Score vector
Probability distribution
So
long
and
thanks
for
Decoder
u
logits
all
the
your
that
1.2
0.9
0.1
−0.5
softmax
y
probabilities
all
the
your
that
0.44
0.33
0.15
0.08
[1 × |V|]
[1 × |V|]

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 $w_i \neq$ 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 **y** = softmax(u / τ) work?

τ = 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 pushes high-probability words higher and low probability word lower, making the distribution more greedy.

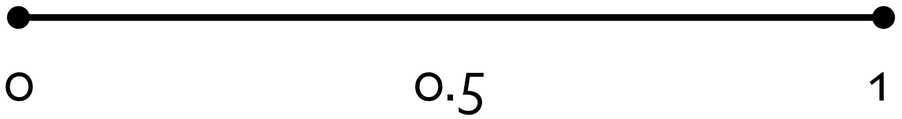
As τ approaches 0, the probability of the most likely word approaches 1.

deterministic!

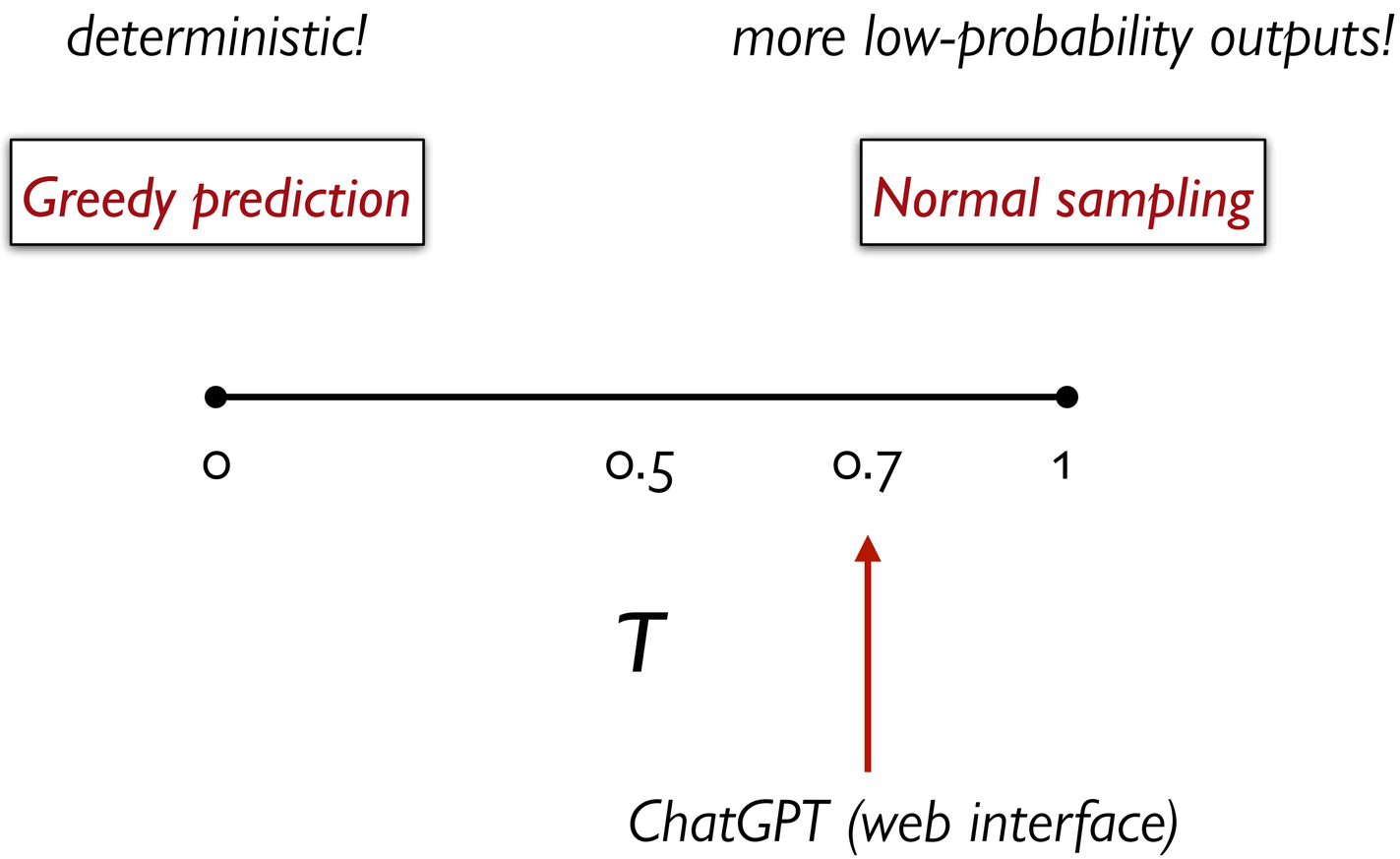
more low-probability outputs!

Greedy prediction

Normal sampling



T

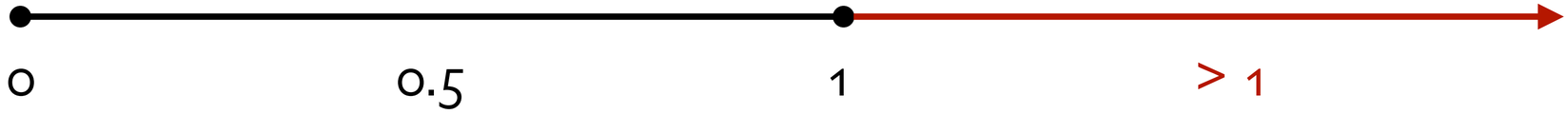

deterministic!
more low-probability outputs!
Greedy prediction
Normal sampling
0
0.5
0.7
1
T
ChatGPT (web interface)

deterministic!

more low-probability outputs!

Greedy prediction

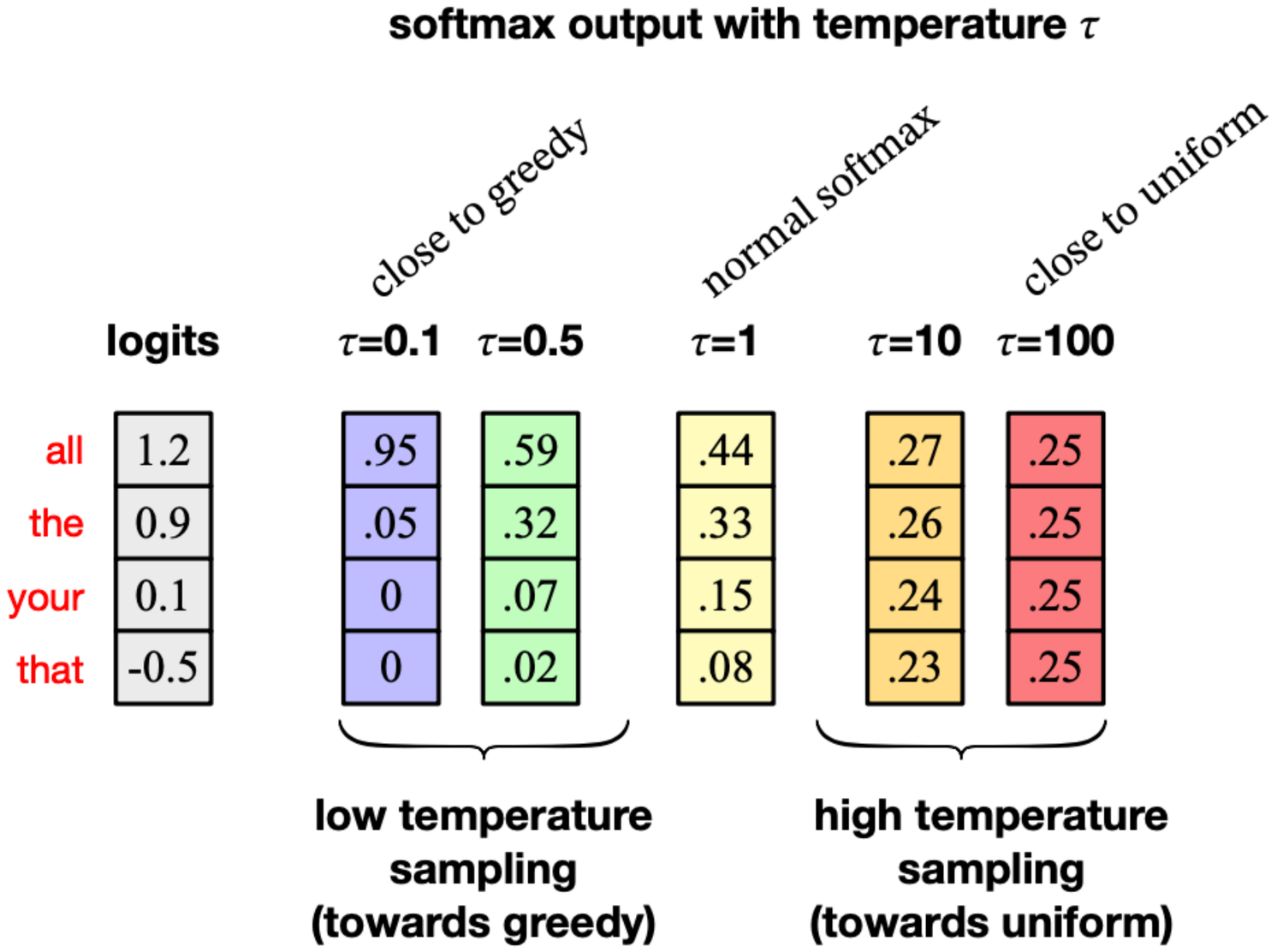
Normal sampling



T

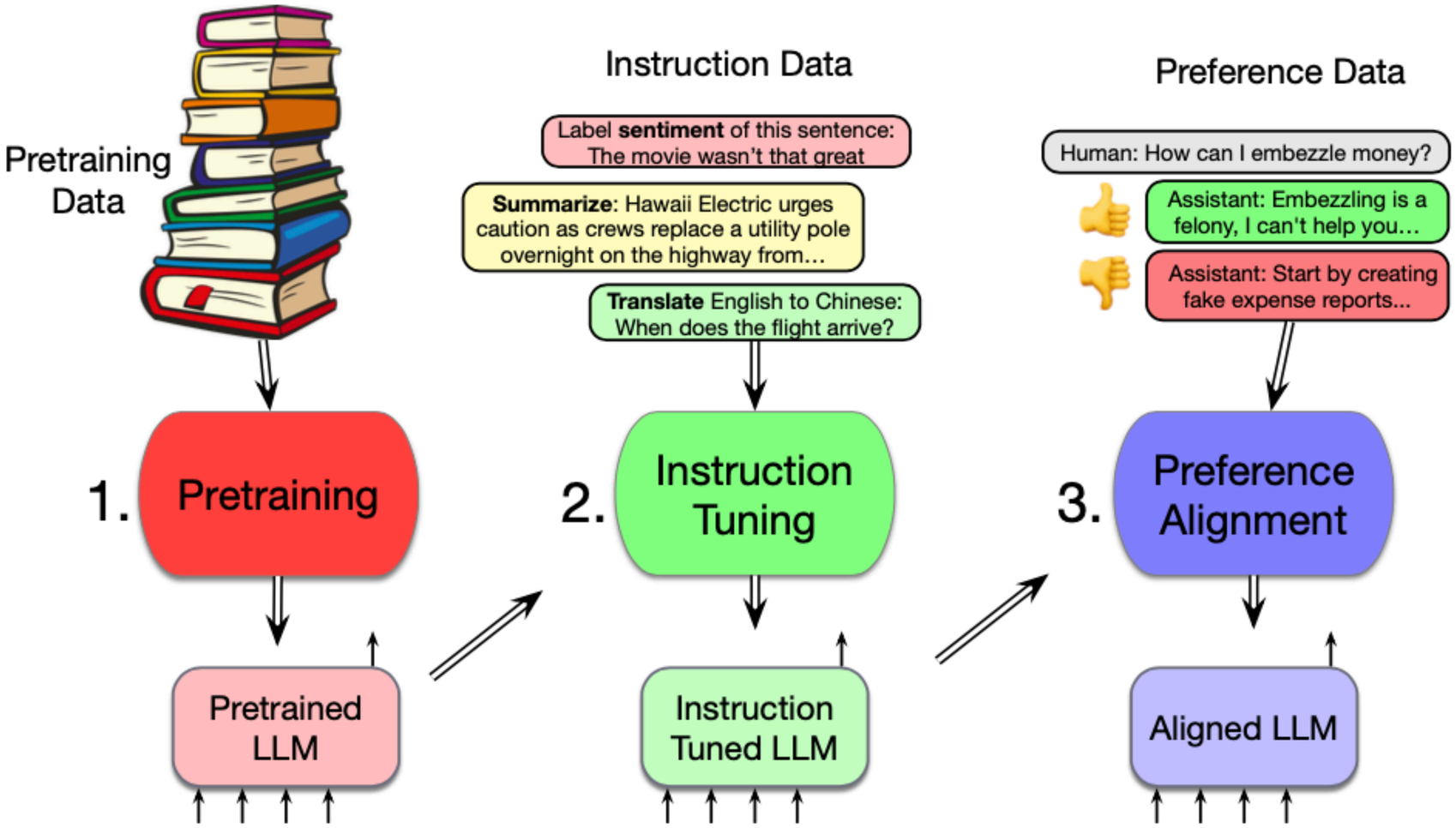
High-temperature sampling

flattens out the probabilities – approaching pure randomness


softmax output with temperature τ
close to greedy
normal softmax
close to uniform
logits
τ=0.1
τ=0.5
τ=1
τ=10
τ=100
all
the
your
that
1.2
0.9
0.1
-0.5
.95
.05
0
0
.59
.32
.07
.02
.44
.33
.15
.08
.27
.26
.24
.23
.25
.25
.25
.25
low temperature sampling (towards greedy)
high temperature sampling (towards uniform)

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:

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

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:

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

Acknowledgments

The lecture incorporates material from:

Jurafsky & Martin, *Speech and Language Processing*, 3rd ed. draft