Modern LLM Recipe
Recap of LM basics

EECS 183/283a: Natural Language Processing

Language Modeling is Hard

$$p(\overline{X}) \in \Delta^{\mathcal{V}^+} \qquad \overline{x} \in \mathcal{V}^+ \qquad p(\overline{x})$$

- How might we go about assigning a probability to **any possible sequence,** even ones we've never seen before?

- One intuition: sentences have internal consistencies!

você fala inglês?
do you speak English?

2nd person singular pronoun, formal

3rd person singular present indicative

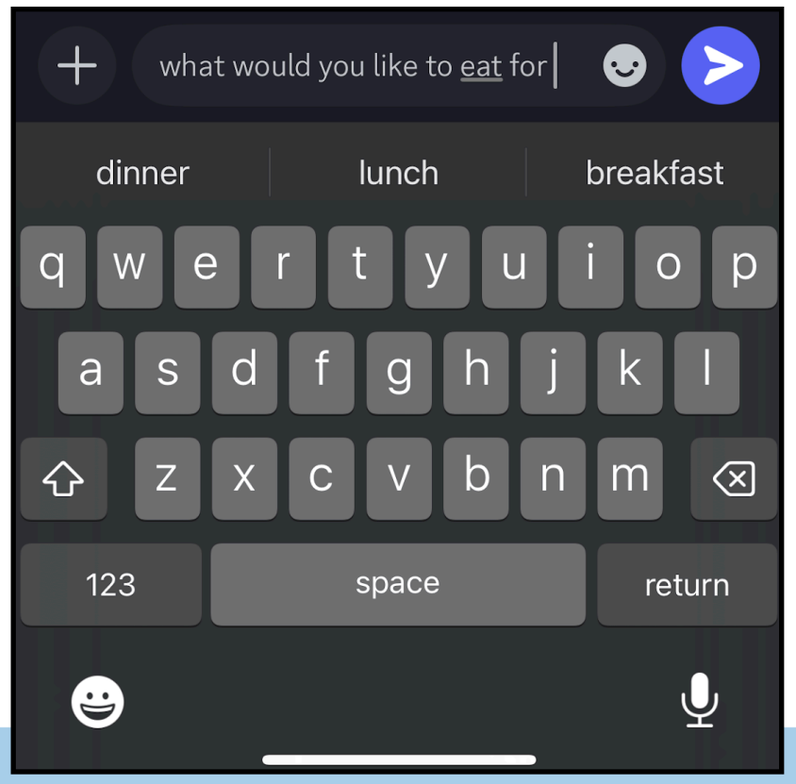
3rd person singular pronoun, masculine	accusative case of "dog"	3rd person singular present, masculine
అతను	కుక్కను	చూస్తాడు
atanu	kukkanu	cūstāḍu
he	the dog	sees

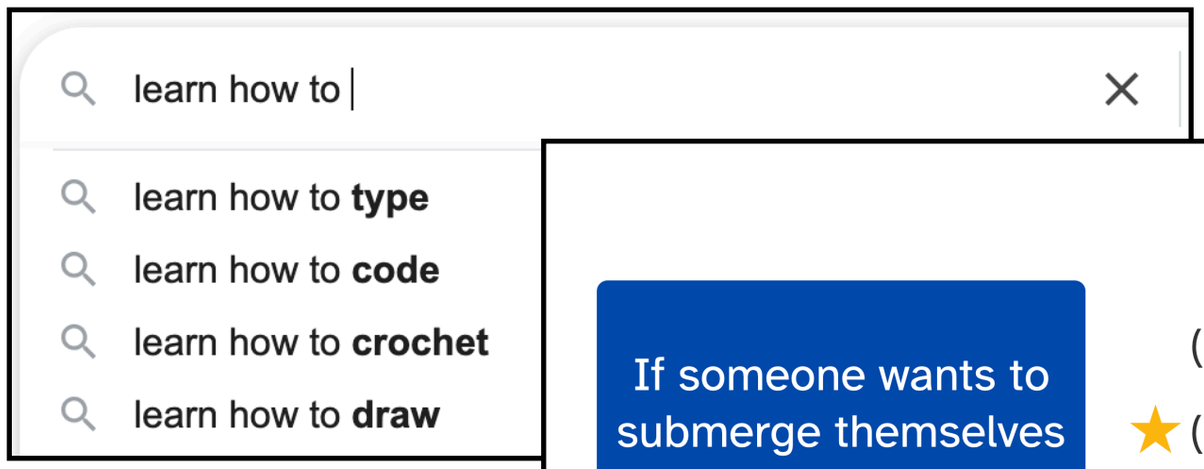
One Approximation

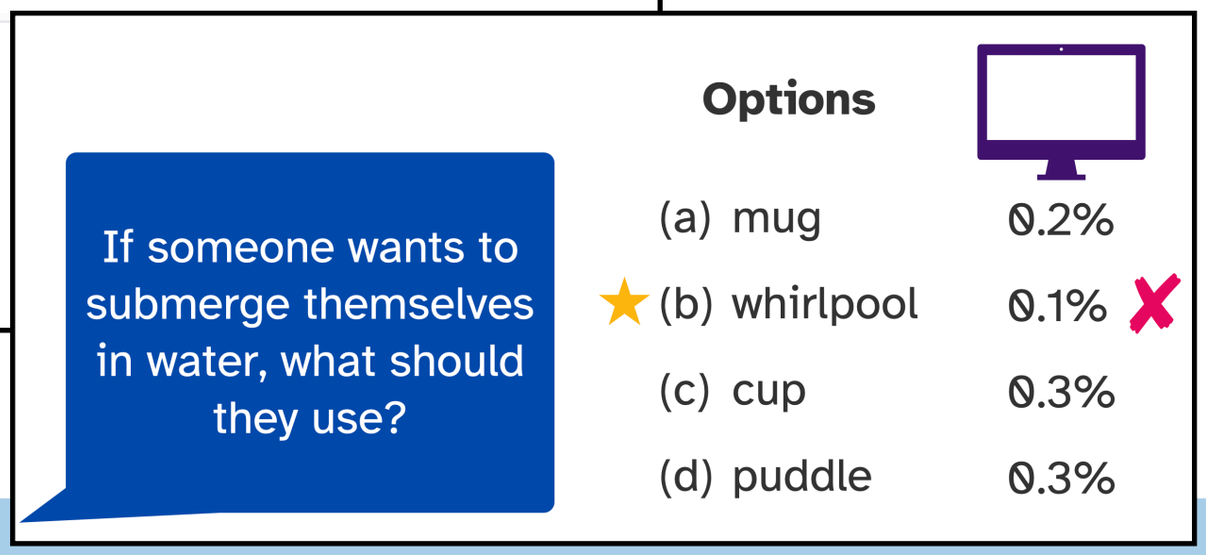
Autoregressive language modeling:

- The probability of a sequence is a product of local token probabilities
- The probability of a token depends on the ones that came before it

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$







Another Approximation

Masked language modeling:

- The probability of a sequence is a product of local token probabilities
- The probability of a token depends on the ones that came before *and after* it

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1} \ldots x_n)$$

The name *anteater* refers to the species' [redacted], which consists mainly of ants and termites.

Autoregressive Language Models

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

$$= p(x_1)p(x_2 \mid x_1) \ldots p(x_n \mid x_1, \ldots, x_{n-1})$$

probability that the first word is x_1

probability that the second word is x_2, given that the first word is x_1

probability that the sentence ends after the sequence $\langle x_1, \ldots, x_{n-1} \rangle$

Core modeling challenge:

How do we compute these conditional probabilities?

Sampling from an Autoregressive Language Model

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

Chain rule decomposition

$$= p(x_1)p(x_2 \mid x_1) \ldots p(x_n \mid x_1, \ldots, x_{n-1})$$

Let's sample a sequence from this approximation:

- Sample the first word $x_1 \sim p(X_1) \in \Delta^{\mathcal{V}}$

$$p(X_1)$$

$\mathcal{X}$	$p(x)$
the	0.04
in	0.02
and	0.02
every	0.02
...	

$$\overline{x} = \langle \textit{the}$$

Sampling from an Autoregressive Language Model

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

$$= p(x_1)p(x_2 \mid x_1) \ldots p(x_n \mid x_1, \ldots, x_{n-1})$$

Let's sample a sequence from this approximation:

- Sample the first word $x_1 \sim p(X_1) \in \Delta^{\mathcal{V}}$
- Sample the second word $x_2 \sim p(X_2 \mid x_1) \in \Delta^{\mathcal{V}}$

$p(X_2 \mid x_1 = \text{the})$

$\mathcal{X}$	$p(x)$
-ir	0.03
same	0.02
impact	0.02
line	0.02
...	

$$\overline{x} = \langle \textit{the, same}$$

Sampling from an Autoregressive Language Model

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

$$= p(x_1)p(x_2 \mid x_1) \ldots p(x_n \mid x_1, \ldots, x_{n-1})$$

Let's sample a sequence from this approximation:

- Sample the first word $x_1 \sim p(X_1) \in \Delta^{\mathcal{V}}$
- Sample the second word
- Sample the third word

$p(X_3 \mid \langle \text{the}, \text{same} \rangle)$

$\mathcal{X}$	$p(x)$
as	0.08
way	0.04
thing	0.04
time	0.04
...	

$$\overline{x} = \langle \textit{ the, same, thing}$$

Sampling from an Autoregressive Language Model

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

$$= p(x_1)p(x_2 \mid x_1) \ldots p(x_n \mid x_1, \ldots, x_{n-1})$$

Let's sample a sequence from this approximation:

- Sample the first word $x_1 \sim p(X_1) \in \Delta^{\mathcal{V}}$ $p(X_i \mid \langle x_1, \ldots, x_{i-1} \rangle)$

$\mathcal{X}$	$p(x)$
.	0.11
as	0.08
that	0.07
,	0.06
...	

- Sample the second word
- Sample the third word
- Keep sampling until we hit `EOS`

$$\overline{x} = \langle \textit{the, same, thing, that}$$

Sampling from an Autoregressive Language Model

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

$$= p(x_1)p(x_2 \mid x_1) \ldots p(x_n \mid x_1, \ldots, x_{n-1})$$

Let's sample a sequence from this approximation:

- Sample the first word $x_1 \sim p(X_1) \in \Delta^{\mathcal{V}}$ $p(X_i \mid \langle x_1, \ldots, x_{i-1} \rangle)$

$\mathcal{X}$	$p(x)$
happened	0.11
happens	0.08
makes	0.07
the	0.06
...	

- Sample the second word
- Sample the third word
- Keep sampling until we hit `EOS`

$$\overline{x} = \langle \textit{the, same, thing, that, the}$$

Sampling from an Autoregressive Language Model

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

$$= p(x_1)p(x_2 \mid x_1) \ldots p(x_n \mid x_1, \ldots, x_{n-1})$$

Let's sample a sequence from this approximation:

- Sample the first word $x_1 \sim p(X_1) \in \Delta^{\mathcal{V}}$ $p(X_i \mid \langle x_1, \ldots, x_{i-1} \rangle)$

x	$p(x)$
-y	0.03
-ir	0.02
person	0.02
line	0.02
…	

- Sample the second word
- Sample the third word
- Keep sampling until we hit `EOS`

$$\overline{x} = \langle \textit{the, same, thing, that, the, -y}$$

Sampling from an Autoregressive Language Model

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

$$= p(x_1)p(x_2 \mid x_1) \ldots p(x_n \mid x_1, \ldots, x_{n-1})$$

Let's sample a sequence from this approximation:

- Sample the first word $x_1 \sim p(X_1) \in \Delta^{\mathcal{V}}$ $p(X_i \mid \langle x_1, \ldots, x_{i-1} \rangle)$

x	$p(x)$
are	0.12
were	0.05
have	0.04
had	0.02
...	

- Sample the second word
- Sample the third word
- Keep sampling until we hit `EOS`

$$\overline{x} = \langle \textit{the, same, thing, that, the, -y, had}$$

Sampling from an Autoregressive Language Model

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

$$= p(x_1)p(x_2 \mid x_1) \ldots p(x_n \mid x_1, \ldots, x_{n-1})$$

Let's sample a sequence from this approximation:

- Sample the first word $x_1 \sim p(X_1) \in \Delta^{\mathcal{V}}$ $p(X_i \mid \langle x_1, \ldots, x_{i-1} \rangle)$
- Sample the second word
- Sample the third word
- Keep sampling until we hit `EOS`

$$\overline{x} = \langle \textit{the, same, thing, that, the, -y, had, EOS} \rangle$$

the same thing that they had

$\mathcal{X}$	$p(x)$
done	0.08
said	0.04
EOS	0.04
to	0.02
…	

Word Types and Tokens

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuckering and Garrod 2004, "A mechanistic psychology of dialogue"

- Text data can be viewed as a sequence of words
- First step in building a language technology: building a function that maps from arbitrary text data to that sequence

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills, '.']
```

tokenized text

Word Types and Tokens

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuekeling and Garrod 2004, "A mechanistic psychology of dialogue"

- Type-token distinction:
  - Type: a unique word in a text corpus
  - Token: an *instance* of a word type, appearing in a particular context

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills, '.']
```

tokenized text

Word Types and Tokens

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuekeling and Garrod 2004, "A mechanistic psychology of dialogue"

```
{'.', ',', ':', 'Every', 'The', 'a', 'adults', 'and', 'are', 'basic',
'can', 'children', 'conversation', 'dialogue', 'even', 'far', 'form',
'from', 'hold', 'illiterate', 'including', 'is', 'language', 'listening',
'most', 'natural', 'of', 'preparing', 'reading', 'skills', 'speeches',
'to', 'universal', 'use', 'user', 'whereas', 'writing', 'young'}
```

wordtypes (vocabulary)

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills, '.']
```

tokenized text

Word Types and Tokens

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuckelring and Garrod 2004, "A mechanistic psychology of dialogue"

```
{'.', ',', ':', 'Every', 'The', 'a', 'adults', 'and', 'are', 'basic',
'can', 'children', 'conversation', 'dialogue', 'even', 'far', 'form',
'from', 'hold', 'illiterate', 'including', 'is', 'language', 'listening',
'most', 'natural', 'of', 'preparing', 'reading', 'skills', 'speeches',
'to', 'universal', 'use', 'user', 'whereas', 'writing', 'young'}
```

wordtypes (vocabulary)

instances (tokens) of wordtype `'.'`

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills, '.']
```

tokenized text

Word Types and Tokens

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuekeling and Garrod 2004, "A mechanistic psychology of dialogue"

```
{'.', ',', ':', 'Every', 'The', 'a', 'adults', 'and', 'are', 'basic',
'can', 'children', 'conversation', 'dialogue', 'even', 'far', 'form',
'from', 'hold', 'illiterate', 'including', 'is', 'language', 'listening',
'most', 'natural', 'of', 'preparing', 'reading', 'skills', 'speeches',
'to', 'universal', 'use', 'user', 'whereas', 'writing', 'young'}
```

wordtypes (vocabulary)

instances (tokens) of wordtype `','`

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills, '.']
```

tokenized text

Word Types and Tokens

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuckering and Garrod 2004, "A mechanistic psychology of dialogue"

```
{'.', ',', ':', 'Every', 'The', 'a', 'adults', 'and', 'are', 'basic',
'can', 'children', 'conversation', 'dialogue', 'even', 'far', 'form',
'from', 'hold', 'illiterate', 'including', 'is', 'language', 'listening',
'most', 'natural', 'of', 'preparing', 'reading', 'skills', 'speeches',
'to', 'universal', 'use', 'user', 'whereas', 'writing', 'young'}
```

wordtypes (vocabulary)

instances (tokens) of wordtype 'Every'

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills, '.']
```

tokenized text

Word Types and Tokens

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuckering and Garrod 2004, "A mechanistic psychology of dialogue"

```
{'.', ',', ':', 'Every', 'The', 'a', 'adults', 'and', 'are', 'basic',
'can', 'children', 'conversation', 'dialogue', 'even', 'far', 'form',
'from', 'hold', 'illiterate', 'including', 'is', 'language', 'listening',
'most', 'natural', 'of', 'preparing', 'reading', 'skills', 'speeches',
'to', 'universal', 'use', 'user', 'whereas', 'writing', 'young'}
```

wordtypes (vocabulary)

instances (tokens) of wordtype 'and'

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills, '.']
```

tokenized text

Word Types and Tokens

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuckelring and Garrod 2004, "A mechanistic psychology of dialogue"

```
{'.', ',', ':', 'Every', 'The', 'a', 'adults', 'and', 'are', 'basic',
'can', 'children', 'conversation', 'dialogue', 'even', 'far', 'form',
'from', 'hold', 'illiterate', 'including', 'is', 'language', 'listening',
'most', 'natural', 'of', 'preparing', 'reading', 'skills', 'speeches',
'to', 'universal', 'use', 'user', 'whereas', 'writing', 'young'}
```

wordtypes (vocabulary)

instances (tokens) of wordtype `'language'`

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills, '.']
```

tokenized text

Word Types and Tokens

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuckelring and Garrod 2004, "A mechanistic psychology of dialogue"

```
vocabulary =
['.', ',', ':', 'Every', 'The', 'a', 'adults', 'and', 'are', 'basic',
'can', 'children', 'conversation', 'dialogue', 'even', 'far', 'form',
'from', 'hold', 'illiterate', 'including', 'is', 'language', 'listening',
'most', 'natural', 'of', 'preparing', 'reading', 'skills', 'speeches',
'to', 'universal', 'use', 'user', 'whereas', 'writing', 'young']
```

vocabulary as a look-up table

```
tokenized_text = ['The', 'most', ... , 'skills, '.']
tokenized_indices = [vocabulary.index(token) for token in tokenized_text]

tokenized_indices: [4, 24, 25, 7, 9, 16, 26, 22, 33, 21, 13, 2, 3, 22,
34, 1, 20, 37, 11, 7, 19, 6, 1, 10, 18, 5, 12, 1, 35, 28, 1, 36, 1, 27,
30, 7, 14, 23, 31, 30, 8, 15, 17, 32, 29, 0]
```

text as a sequence of wordtype indices

Tokenize on Spaces

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fickenring and Garrod 2004, "A mechanistic psychology of dialogue"

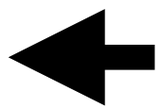
- Simplest tokenizer (for English): splitting on spaces

```
tokenized = s.split(' ')
```

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue:', 'Every', 'language', 'user,', 'including',
'young', 'children', 'and', 'illiterate', 'adults,', 'can', 'hold', 'a',
'conversation,', 'whereas', 'reading,', 'writing,', 'preparing',
'speeches', 'and', 'even', 'listening', 'to', 'speeches', 'are', 'far',
'from', 'universal', 'skills.']
```

- But this gets us some weird wordtypes:

```
'dialogue:'
'user,'
'skills.'
```



Not really words different from *dialogue*, *user*, *skills*

Rule-Based Tokenization

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuckering and Garrod 2004, "A mechanistic psychology of dialogue"

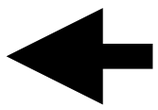
- nltk tokenizers, with special rules for punctuation

```
import nltk
tokenized = nltk.word_tokenize(s)
```

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills, '.']
```

- But this still loses similarity between wordtypes

```
'skills'
'reading'
'speeches'
```



Lexically similar to, but morphologically distinct from *skill, read, speech*

Rule-Based Tokenization

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuckering and Garrod 2004, "A mechanistic psychology of dialogue"

- nltk tokenizers, with special rules for punctuation

```
import nltk
tokenized = nltk.word_tokenize(s)
```

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills, '.']
```

- And doesn't work for all languages

โดยที่การยอมรับนับถือเกียรติศักดิ์ประจำตัว และสิทธิเท่าเทียมกันและโอนมิได้ของบรรดา สมาชิก ทั้ง หลายแห่งครอบครัว มนุษย์เป็นหลักมูลเหตุแห่งอิสรภาพ ความยุติธรรม และสันติภาพในโลก

Universal Declaration of Human Rights in Thai

Rule-Based Tokenization

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuckering and Garrod 2004, "A mechanistic psychology of dialogue"

- nltk tokenizers, with special rules for punctuation

```
import nltk
tokenized = nltk.word_tokenize(s)
```

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills, '.']
```

- Once you've "trained" your tokenizer, you're stuck with it

```
vocab = ['.', ',', ':', 'Every', ... , 'writing', 'young']
vocab.index('ChatGPT') → not found!
```

Rule-Based Tokenization

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuckering and Garrod 2004, "A mechanistic psychology of dialogue"

- nltk tokenizers, with special rules for punctuation

```
import nltk
tokenized = nltk.word_tokenize(s)
```

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills, '.']
```

- Once you've "trained" your tokenizer, you're stuck with it

```
vocab = ['.', ',', ':', 'Every', ... , 'writing', 'young', '<UNK>']
tokenized_indices =
    [vocabulary.index(token) for token in tokenized_text
     if token in vocabulary
     else vocabulary.index('<UNK>')]
```

add a special token for unknown words

Bytes as Wordtypes

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

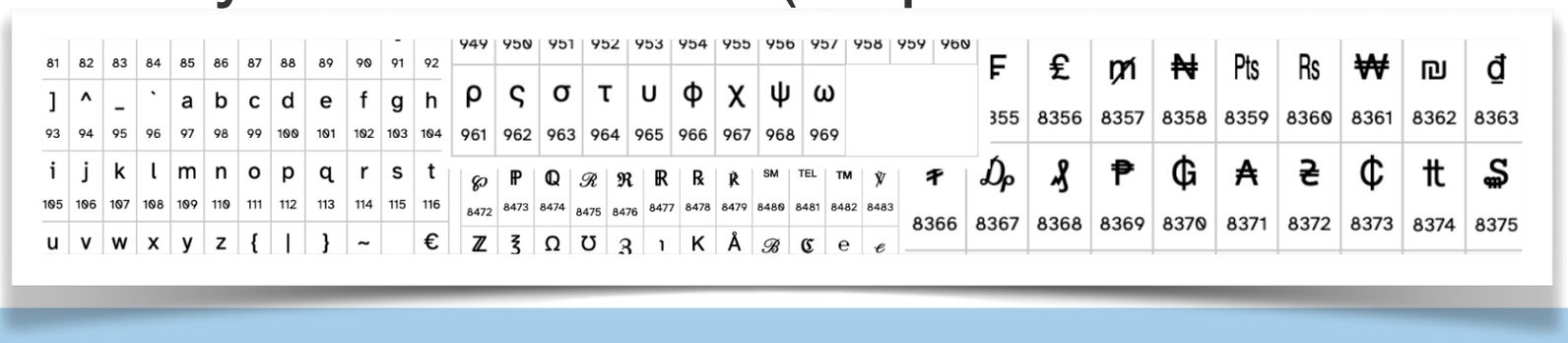
from Fuckering and Garrod 2004, "A mechanistic psychology of dialogue"

- Strings are sequences of characters (bytes)!

```
tokenized = s.encode()
```

```
54 68 65 20 6d 6f 73 74 20 6e 61 74 75 72 61 6c 20 61 6e 64 20 62 61 73 69 63 20 66 6f 72 6d 20 6f 66 20
6c 61 6e 67 75 61 67 65 20 75 73 65 20 69 73 20 64 69 61 6c 6f 67 75 65 3a 20 45 76 65 72 79 20 6c 61 6e
67 75 61 67 65 20 75 73 65 72 2c 20 69 6e 63 6c 75 64 69 6e 67 20 79 6f 75 6e 67 20 63 68 69 6c 64 72 65
6e 20 61 6e 64 20 69 6c 6c 69 74 65 72 61 74 65 20 61 64 75 6c 74 73 2c 20 63 61 6e 20 68 6f 6c 64 20 61
20 63 6f 6e 76 65 72 73 61 74 69 6f 6e 2c 20 77 68 65 72 65 61 73 20 72 65 61 64 69 6e 67 2c 20 77 72 69
74 69 6e 67 2c 20 70 72 65 70 61 72 69 6e 67 20 73 70 65 65 63 68 65 73 20 61 6e 64 20 65 76 65 6e 20 6c
69 73 74 65 6e 69 6e 67 20 74 6f 20 73 70 65 65 63 68 65 73 20 61 72 65 20 66 61 72 20 66 72 6f 6d 20 75
6e 69 76 65 72 73 61 6c 20 73 6b 69 6c 6c 73 2e
```

- Now our vocabulary is a fixed size (all possible Unicode characters)



Bytes as Wordtypes

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

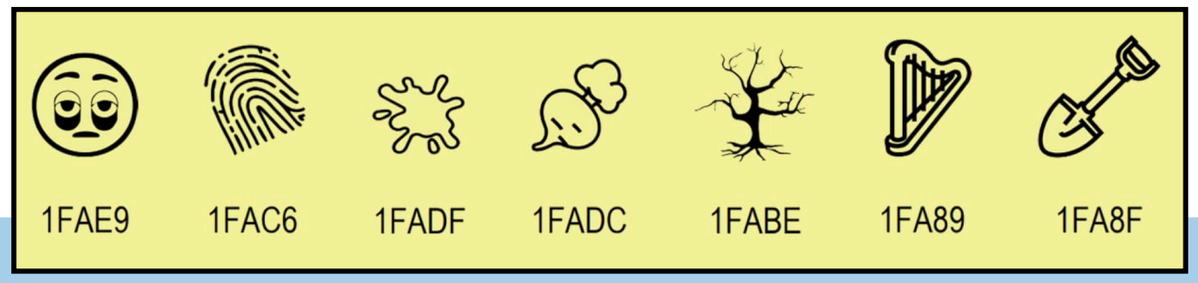
from Fuckelring and Garrod 2004, "A mechanistic psychology of dialogue"

- Strings are sequences of characters (bytes)!

```
tokenized = s.encode()
```

```
54 68 65 20 6d 6f 73 74 20 6e 61 74 75 72 61 6c 20 61 6e 64 20 62 61 73 69 63 20 66 6f 72 6d 20 6f 66 20
6c 61 6e 67 75 61 67 65 20 75 73 65 20 69 73 20 64 69 61 6c 6f 67 75 65 3a 20 45 76 65 72 79 20 6c 61 6e
67 75 61 67 65 20 75 73 65 72 2c 20 69 6e 63 6c 75 64 69 6e 67 20 79 6f 75 6e 67 20 63 68 69 6c 64 72 65
6e 20 61 6e 64 20 69 6c 6c 69 74 65 72 61 74 65 20 61 64 75 6c 74 73 2c 20 63 61 6e 20 68 6f 6c 64 20 61
20 63 6f 6e 76 65 72 73 61 74 69 6f 6e 2c 20 77 68 65 72 65 61 73 20 72 65 61 64 69 6e 67 2c 20 77 72 69
74 69 6e 67 2c 20 70 72 65 70 61 72 69 6e 67 20 73 70 65 65 63 68 65 73 20 61 6e 64 20 65 76 65 6e 20 6c
69 73 74 65 6e 69 6e 67 20 74 6f 20 73 70 65 65 63 68 65 73 20 61 72 65 20 66 61 72 20 66 72 6f 6d 20 75
6e 69 76 65 72 73 61 6c 20 73 6b 69 6c 6c 73 2e
```

- And we don't have any unknown wordtypes (until the Unicode Consortium adds new emoji)



Bytes as Wordtypes

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuckelring and Garrod 2004, "A mechanistic psychology of dialogue"

- Strings are sequences of characters (bytes)!

```
tokenized = s.encode()
```

```
54 68 65 20 6d 6f 73 74 20 6e 61 74 75 72 61 6c 20 61 6e 64 20 62 61 73 69 63 20 66 6f 72 6d 20 6f 66 20
6c 61 6e 67 75 61 67 65 20 75 73 65 20 69 73 20 64 69 61 6c 6f 67 75 65 3a 20 45 76 65 72 79 20 6c 61 6e
67 75 61 67 65 20 75 73 65 72 2c 20 69 6e 63 6c 75 64 69 6e 67 20 79 6f 75 6e 67 20 63 68 69 6c 64 72 65
6e 20 61 6e 64 20 69 6c 6c 69 74 65 72 61 74 65 20 61 64 75 6c 74 73 2c 20 63 61 6e 20 68 6f 6c 64 20 61
20 63 6f 6e 76 65 72 73 61 74 69 6f 6e 2c 20 77 68 65 72 65 61 73 20 72 65 61 64 69 6e 67 2c 20 77 72 69
74 69 6e 67 2c 20 70 72 65 70 61 72 69 6e 67 20 73 70 65 65 63 68 65 73 20 61 6e 64 20 65 76 65 6e 20 6c
69 73 74 65 6e 69 6e 67 20 74 6f 20 73 70 65 65 63 68 65 73 20 61 72 65 20 66 61 72 20 66 72 6f 6d 20 75
6e 69 76 65 72 73 61 6c 20 73 6b 69 6c 6c 73 2e
```

- But: individual characters are not meaningful

```
['.', ',', ':', 'Every', ... , 'user', 'whereas', 'writing', 'young']
```

vs.

```
['.', ',', ':', 'A', 'B', 'C', ... , 'a', 'b', 'c', ... , '7', '8', '9']
```

Bytes as Wordtypes

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuckelring and Garrod 2004, "A mechanistic psychology of dialogue"

- Strings are sequences of characters (bytes)!

```
tokenized = s.encode()
```

```
54 68 65 20 6d 6f 73 74 20 6e 61 74 75 72 61 6c 20 61 6e 64 20 62 61 73 69 63 20 66 6f 72 6d 20 6f 66 20
6c 61 6e 67 75 61 67 65 20 75 73 65 20 69 73 20 64 69 61 6c 6f 67 75 65 3a 20 45 76 65 72 79 20 6c 61 6e
67 75 61 67 65 20 75 73 65 72 2c 20 69 6e 63 6c 75 64 69 6e 67 20 79 6f 75 6e 67 20 63 68 69 6c 64 72 65
6e 20 61 6e 64 20 69 6c 6c 69 74 65 72 61 74 65 20 61 64 75 6c 74 73 2c 20 63 61 6e 20 68 6f 6c 64 20 61
20 63 6f 6e 76 65 72 73 61 74 69 6f 6e 2c 20 77 68 65 72 65 61 73 20 72 65 61 64 69 6e 67 2c 20 77 72 69
74 69 6e 67 2c 20 70 72 65 70 61 72 69 6e 67 20 73 70 65 65 63 68 65 73 20 61 6e 64 20 65 76 65 6e 20 6c
69 73 74 65 6e 69 6e 67 20 74 6f 20 73 70 65 65 63 68 65 73 20 61 72 65 20 66 61 72 20 66 72 6f 6d 20 75
6e 69 76 65 72 73 61 6c 20 73 6b 69 6c 6c 73 2e
```

- It's now the ML model's job to learn to compose words from scratch

Bytes as Wordtypes

The most natural and basic form of language use is dialogue: Every language user, including young children and illiterate adults, can hold a conversation, whereas reading, writing, preparing speeches and even

from Fuckelring and Garrod 2004, "A mechanistic psychology of dialogue"

- Strings are sequences of characters (bytes)!

```
tokenized = s.encode()
```

```
54 68 65 20 6d 6f 73 74 20 6e 61 74 75 72 61 6c 20 61 6e 64 20 62 61 73 69 63 20 66 6f 72 6d 20 6f 66 20
6c 61 6e 67 75 61 67 65 20 75 73 65 20 69 73 20 64 69 61 6c 6f 67 75 65 3a 20 45 76 65 72 79 20 6c 61 6e
67 75 61 67 65 20 75 73 65 72 2c 20 69 6e 63 6c 75 64 69 6e 67 20 79 6f 75 6e 67 20 63 68 69 6c 64 72 65
6e 20 61 6e 64 20 69 6c 6c 69 74 65 72 61 74 65 20 61 64 75 6c 74 73 2c 20 63 61 6e 20 68 6f 6c 64 20 61
20 63 6f 6e 76 65 72 73 61 74 69 6f 6e 2c 20 77 68 65 72 65 61 73 20 72 65 61 64 69 6e 67 2c 20 77 72 69
74 69 6e 67 2c 20 70 72 65 70 61 72 69 6e 67 20 73 70 65 65 63 68 65 73 20 61 6e 64 20 65 76 65 6e 20 6c
69 73 74 65 6e 69 6e 67 20 74 6f 20 73 70 65 65 63 68 65 73 20 61 72 65 20 66 61 72 20 66 72 6f 6d 20 75
6e 69 76 65 72 73 61 6c 20 73 6b 69 6c 6c 73 2e
```

- But: input sequences are much longer

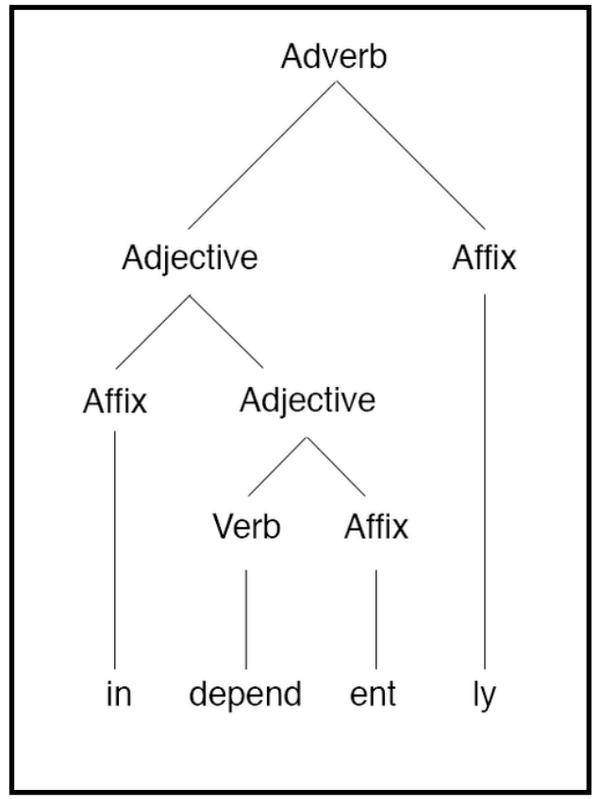
```
‘language’
```

vs.

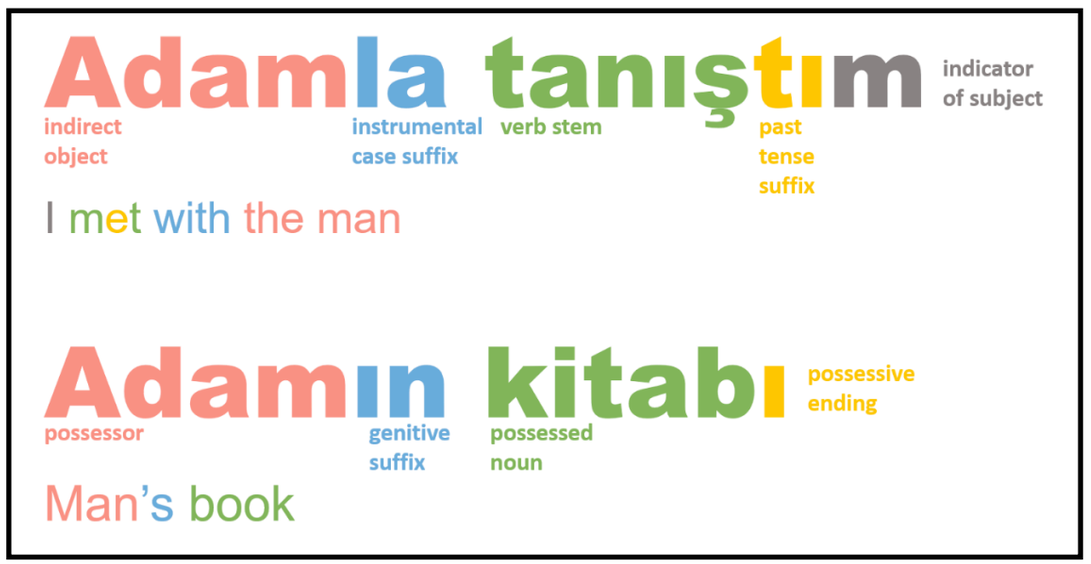
```
‘l’, ‘a’, ’n’, ‘g’, ‘u’, ‘a’, ‘g’, ‘e’
```

A Compromise: Subword Tokenization

- Main principle: words are (often) composed of subparts (morphemes)
- Our vocabulary should have entries for frequent words kept whole, because we have a lot of data about those words
- But it should also have entries for parts of less-frequent words, so our ML models can learn how to compose parts of words into whole words (especially unfamiliar words!)



Wikipedia (Annie Yang)



Byte Pair Encoding

- Modern standard for building a tokenizer
- Inputs: collection of texts and target vocabulary size
- Initial vocabulary is the set of all bytes (characters) across the texts
- Until the target vocabulary size is reached, repeat the following:
  - Tokenize all of the texts using the current vocabulary
  - Find the most common bigram in the tokenized texts, then add it to the vocabulary as a new wordtype

Byte Pair Encoding

Documents + frequencies: `('hug', 10), ('pug', 5), ('pun', 12), ('bun', 4), ('hugs', 5)`

vocabulary

`('h', 'u', 'g', 'p', 'n', 'b', 's')`

Example from HuggingFace

Byte Pair Encoding

Documents + frequencies: `('hug', 10), ('pug', 5), ('pun', 12), ('bun', 4), ('hugs', 5)`

vocabulary

`('h', 'u', 'g', 'p', 'n', 'b', 's')`

tokenized texts

`('h' 'u' 'g', 10), ('p' 'u' 'g', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'u' 'g' 's', 5)`

bigrams + frequencies

```
'h' 'u'    15
```

Example from HuggingFace

Byte Pair Encoding

Documents + frequencies: `('hug', 10), ('pug', 5), ('pun', 12), ('bun', 4), ('hugs', 5)`

vocabulary

`('h', 'u', 'g', 'p', 'n', 'b', 's')` →

tokenized texts

`('h' 'u' 'g', 10), ('p' 'u' 'g', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'u' 'g' 's', 5)`

bigrams + frequencies

```
'h' 'u'    15
'u' 'g'    20
```

Example from HuggingFace

Byte Pair Encoding

Documents + frequencies: `('hug', 10), ('pug', 5), ('pun', 12), ('bun', 4), ('hugs', 5)`

vocabulary

`('h', 'u', 'g', 'p', 'n', 'b', 's')`

tokenized texts

`('h' 'u' 'g', 10), ('p' 'u' 'g', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'u' 'g' 's', 5)`

bigrams + frequencies

```
'h' 'u'    15
'u' 'g'    20
'p' 'u'    17
```

Example from HuggingFace

Byte Pair Encoding

Documents + frequencies: `('hug', 10), ('pug', 5), ('pun', 12), ('bun', 4), ('hugs', 5)`

vocabulary

`('h', 'u', 'g', 'p', 'n', 'b', 's')` →

tokenized texts

`('h' 'u' 'g', 10), ('p' 'u' 'g', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'u' 'g' 's', 5)`

bigrams + frequencies

'h'	'u'	15
'u'	'g'	20
'p'	'u'	17
'u'	'n'	16

Example from HuggingFace

Byte Pair Encoding

Documents + frequencies: `('hug', 10), ('pug', 5), ('pun', 12), ('bun', 4), ('hugs', 5)`

vocabulary

`('h', 'u', 'g', 'p', 'n', 'b', 's')` →

tokenized texts

`('h' 'u' 'g', 10), ('p' 'u' 'g', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'u' 'g' 's', 5)`

bigrams + frequencies

'h'	'u'	15
'u'	'g'	20
'p'	'u'	17
'u'	'n'	16
'b'	'u'	4

Example from HuggingFace

Byte Pair Encoding

Documents + frequencies: `('hug', 10), ('pug', 5), ('pun', 12), ('bun', 4), ('hugs', 5)`

vocabulary

`('h', 'u', 'g', 'p', 'n', 'b', 's')` →

tokenized texts

`('h' 'u' 'g', 10), ('p' 'u' 'g', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'u' 'g' 's', 5)`

bigrams + frequencies

'h'	'u'	15
'u'	'g'	20
'p'	'u'	17
'u'	'n'	16
'b'	'u'	4
'g'	's'	5

Example from HuggingFace

Byte Pair Encoding

Documents + frequencies: `('hug', 10), ('pug', 5), ('pun', 12), ('bun', 4), ('hugs', 5)`

vocabulary

`('h', 'u', 'g', 'p', 'n', 'b', 's')`

tokenized texts

`('h' 'u' 'g', 10), ('p' 'u' 'g', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'u' 'g' 's', 5)`

bigrams + frequencies

'h' 'u'	15
'u' 'g'	20
'p' 'u'	17
'u' 'n'	16
'b' 'u'	4
'g' 's'	5

most frequent bigram; add to vocabulary

Example from HuggingFace

Byte Pair Encoding

Documents + frequencies: `('hug', 10), ('pug', 5), ('pun', 12), ('bun', 4), ('hugs', 5)`

vocabulary *tokenized texts*

`('h', 'u', 'g', 'p', 'n', 'b', 's')` → `('h' 'u' 'g', 10), ('p' 'u' 'g', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'u' 'g' 's', 5)`

`('h', 'u', 'g', 'p', 'n', 'b', 's', 'ug')` → `('h' 'ug', 10), ('p' 'ug', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'ug' 's', 5)`

Example from HuggingFace

Byte Pair Encoding

Documents + frequencies: `('hug', 10), ('pug', 5), ('pun', 12), ('bun', 4), ('hugs', 5)`

vocabulary *tokenized texts*

`('h', 'u', 'g', 'p', 'n', 'b', 's')` → `('h' 'u' 'g', 10), ('p' 'u' 'g', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'u' 'g' 's', 5)`

`('h', 'u', 'g', 'p', 'n', 'b', 's', 'ug')` → `('h' 'ug', 10), ('p' 'ug', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'ug' 's', 5)`

Example from HuggingFace

Byte Pair Encoding

Documents + frequencies: `('hug', 10), ('pug', 5), ('pun', 12), ('bun', 4), ('hugs', 5)`

vocabulary *tokenized texts*

`('h', 'u', 'g', 'p', 'n', 'b', 's')` → `('h' 'u' 'g', 10), ('p' 'u' 'g', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'u' 'g' 's', 5)`

`('h', 'u', 'g', 'p', 'n', 'b', 's', 'ug')` → `('h' 'ug', 10), ('p' 'ug', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'ug' 's', 5)`

`('h', 'u', 'g', 'p', 'n', 'b', 's', 'ug', 'un')` → `('h' 'ug', 10), ('p' 'ug', 5), ('p' 'un', 12), ('b' 'un', 4), ('h' 'ug' 's', 5)`

Example from HuggingFace

Byte Pair Encoding

Documents + frequencies: `('hug', 10), ('pug', 5), ('pun', 12), ('bun', 4), ('hugs', 5)`

vocabulary *tokenized texts*

`('h', 'u', 'g', 'p', 'n', 'b', 's')` → `('h' 'u' 'g', 10), ('p' 'u' 'g', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'u' 'g' 's', 5)`

`('h', 'u', 'g', 'p', 'n', 'b', 's', 'ug')` → `('h' 'ug', 10), ('p' 'ug', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'ug' 's', 5)`

`('h', 'u', 'g', 'p', 'n', 'b', 's', 'ug', 'un')` → `('h' 'ug', 10), ('p' 'ug', 5), ('p' 'un', 12), ('b' 'un', 4), ('h' 'ug' 's', 5)`

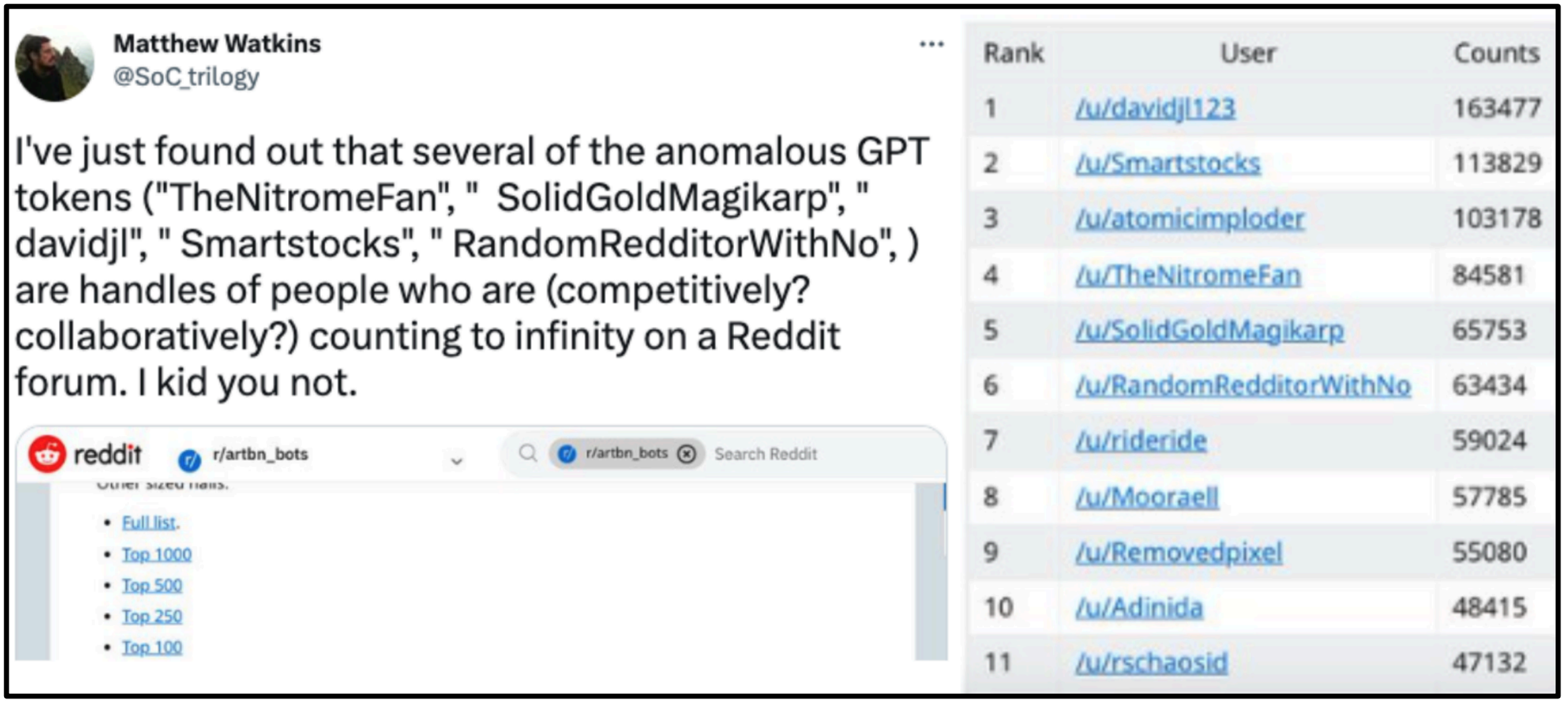
`('h', 'u', 'g', 'p', 'n', 'b', 's', 'ug', 'un', 'hug')` → `('hug', 10), ('p' 'ug', 5), ('p' 'un', 12), ('b' 'un', 4), ('hug' 's', 5)`

New word: “puns”

`'p' 'u' 'n' 's'` → `'p' 'un' 's'` ✓

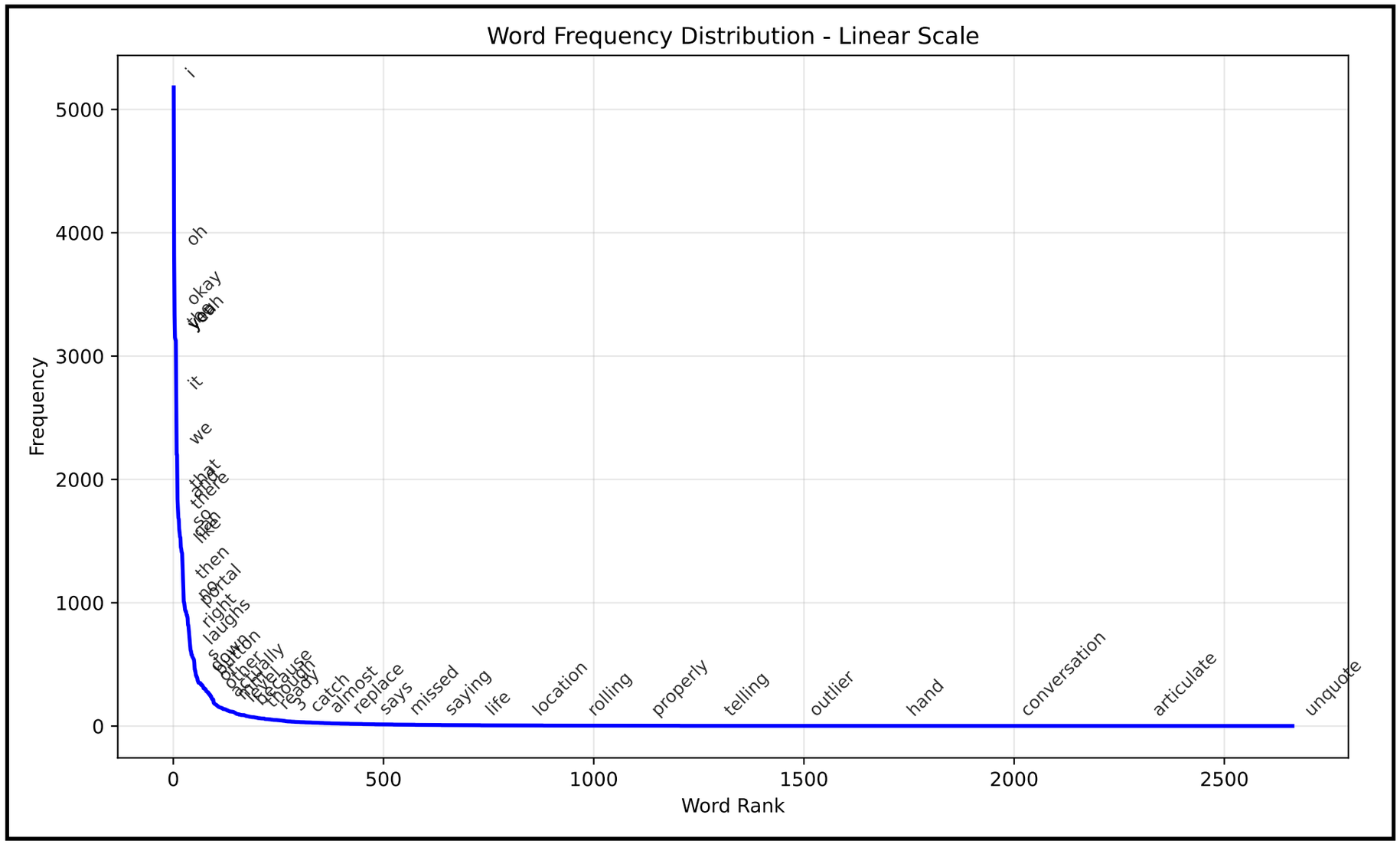
Example from HuggingFace

Vocabularies



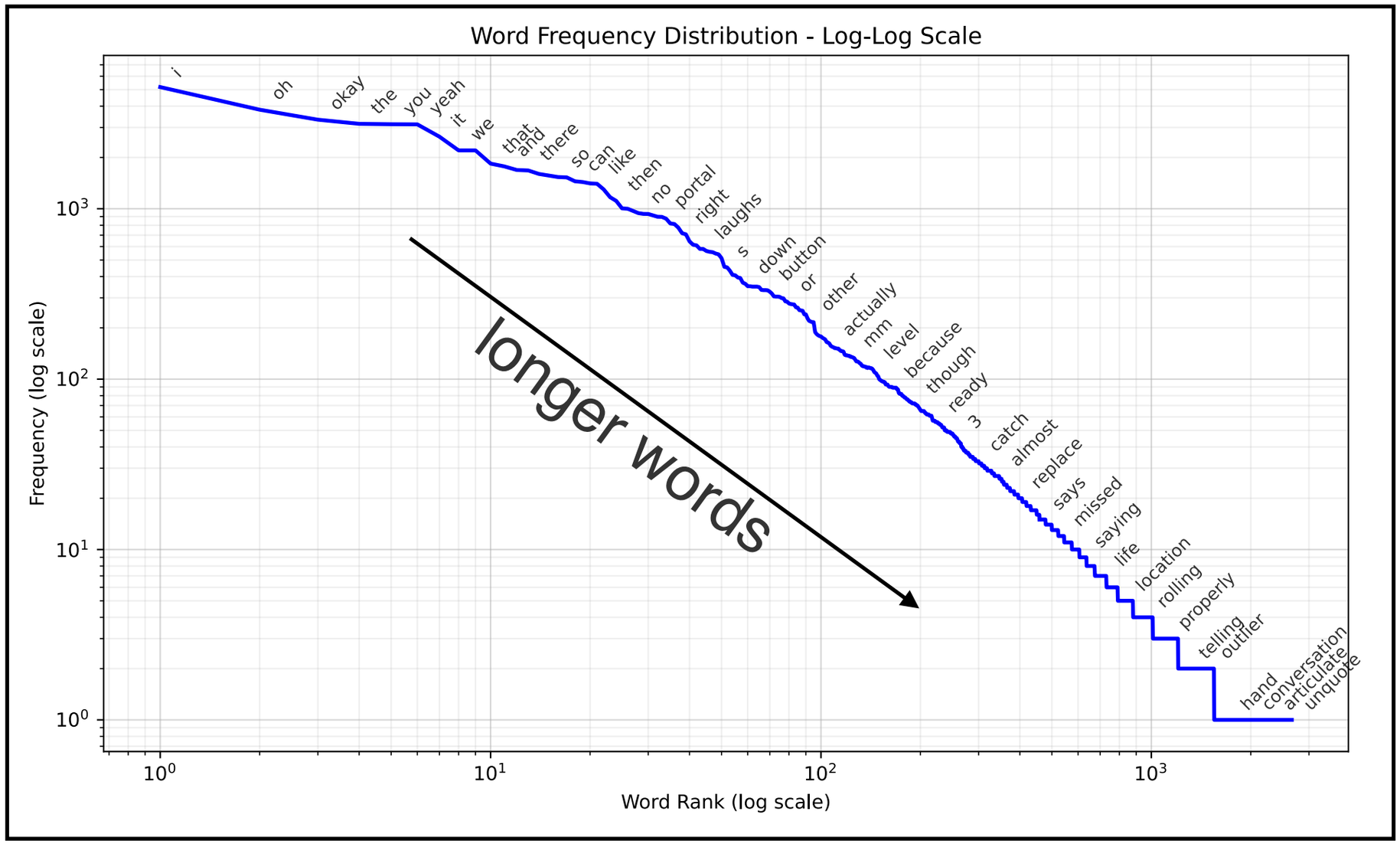
Vocabularies

Long-tail (“Zipfian”) distribution of wordtypes
(from Portal 2 dialogues)



Vocabularies

Long-tail (“Zipfian”) distribution of wordtypes
(from Portal 2 dialogues)



Recap: Sequence-to-Sequence with RNNs

Encode

Decode

BOS

I

like

pizza

EOS

$\overrightarrow{h_0}$ $\overrightarrow{h_1}$ $\overrightarrow{h_2}$ $\overrightarrow{h_3}$ $\overrightarrow{h_4}$

$\overleftarrow{h_4}$ $\overleftarrow{h_3}$ $\overleftarrow{h_2}$ $\overleftarrow{h_1}$ $\overleftarrow{h_0}$

$\text{Enc}(\overline{x})$

Problem 2
Fixed-size representation of entire input sequence.

BOS → mi → pizza → EOS

h_1 h_2 h_5

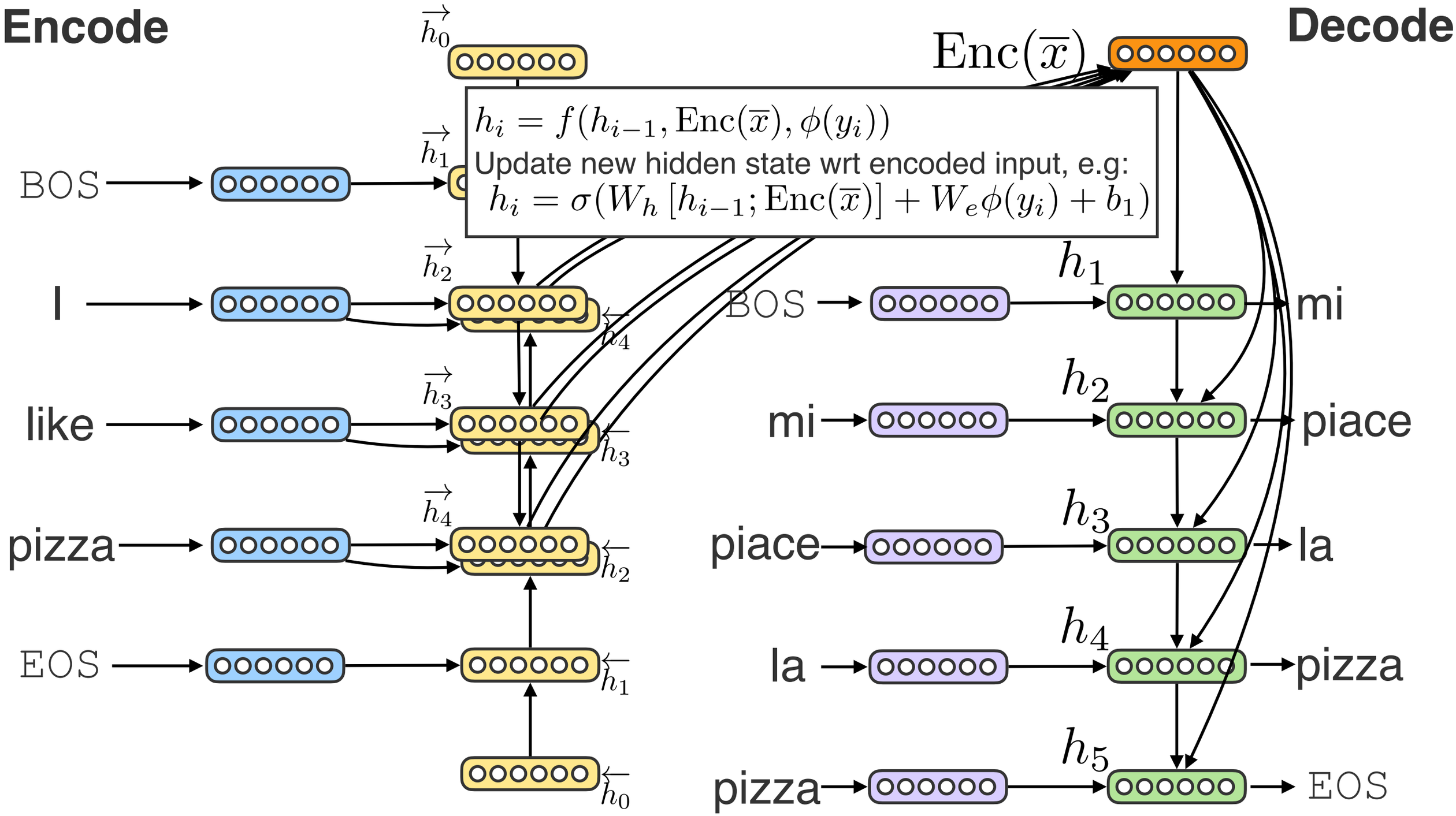
Problem 1
Long-distance dependencies are harder to represent than shorter ones.

*The federal government wants to **do** something about the high prices for gas, electricity, and fuel*

*Die Bundesregierung will etwas gegen die hohen Preise für Gas, Strom und Treibstoff **machen**.*

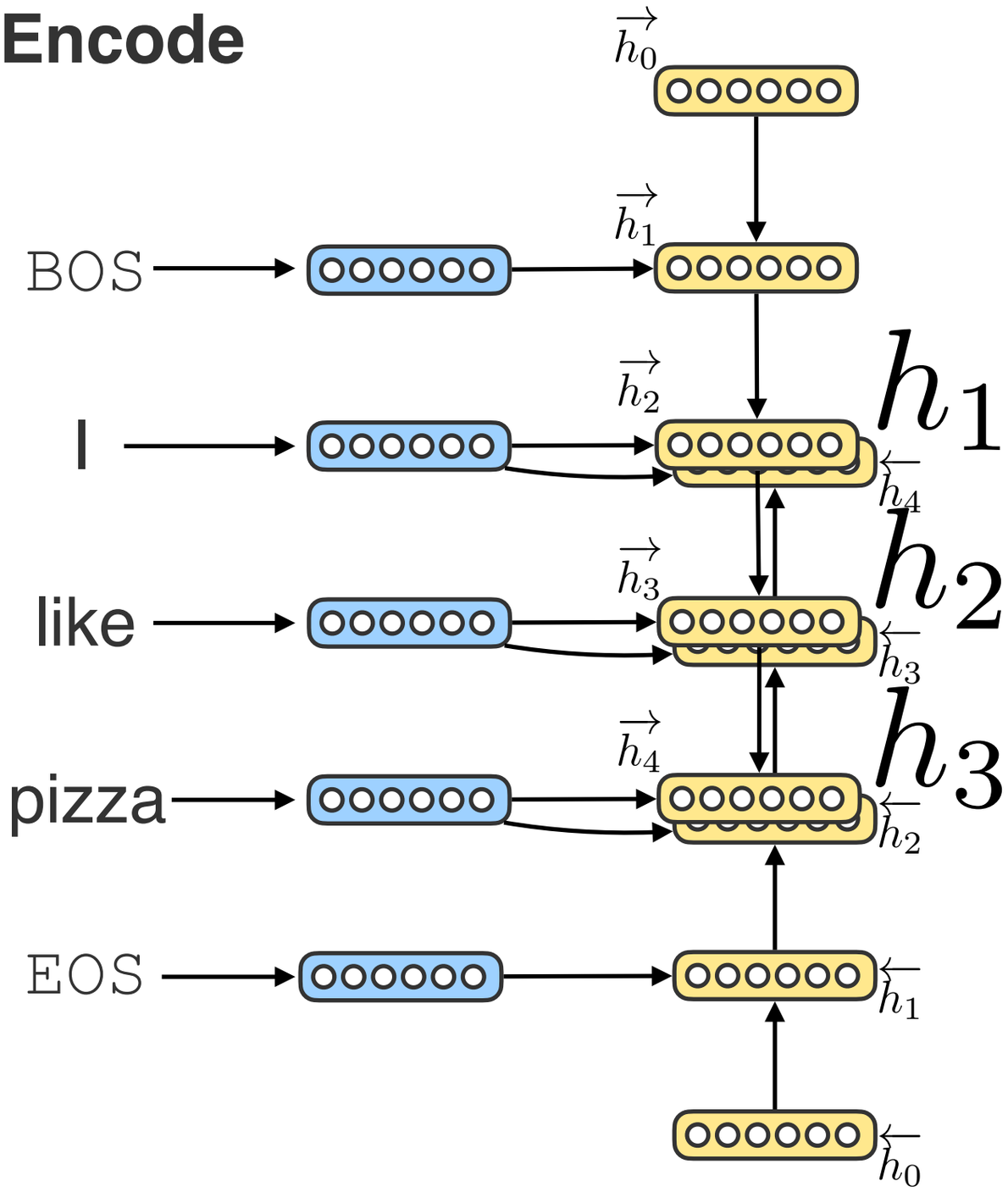
Problem 3
We have to encode the input sequence token-by-token O(n)

Recap:
Problem 1: Long-Distance Dependencies



Recap:
Problem 2: Fixed-Size Sequence Embedding

Encode



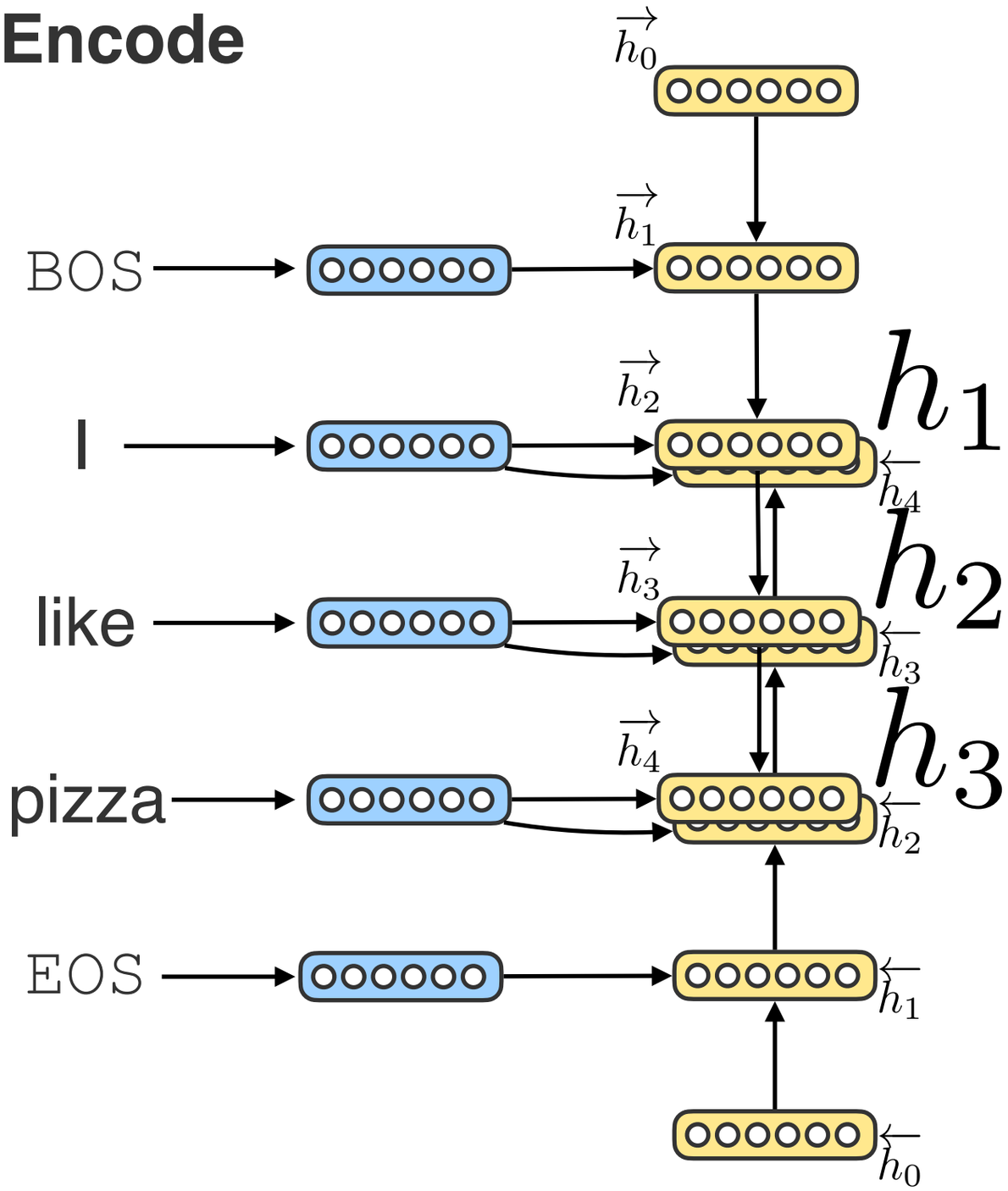
$$\mathrm{Enc}(\overline{x}) = \sum_{i=1}^{|\overline{x}|} p(i) h_i$$

$$p(I) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$$

- Generalization of pooling: assign a **probability distribution** over input indices, and compute **weighted** sum over hidden states

- What if this probability distribution could change during generation?

Recap: Attention

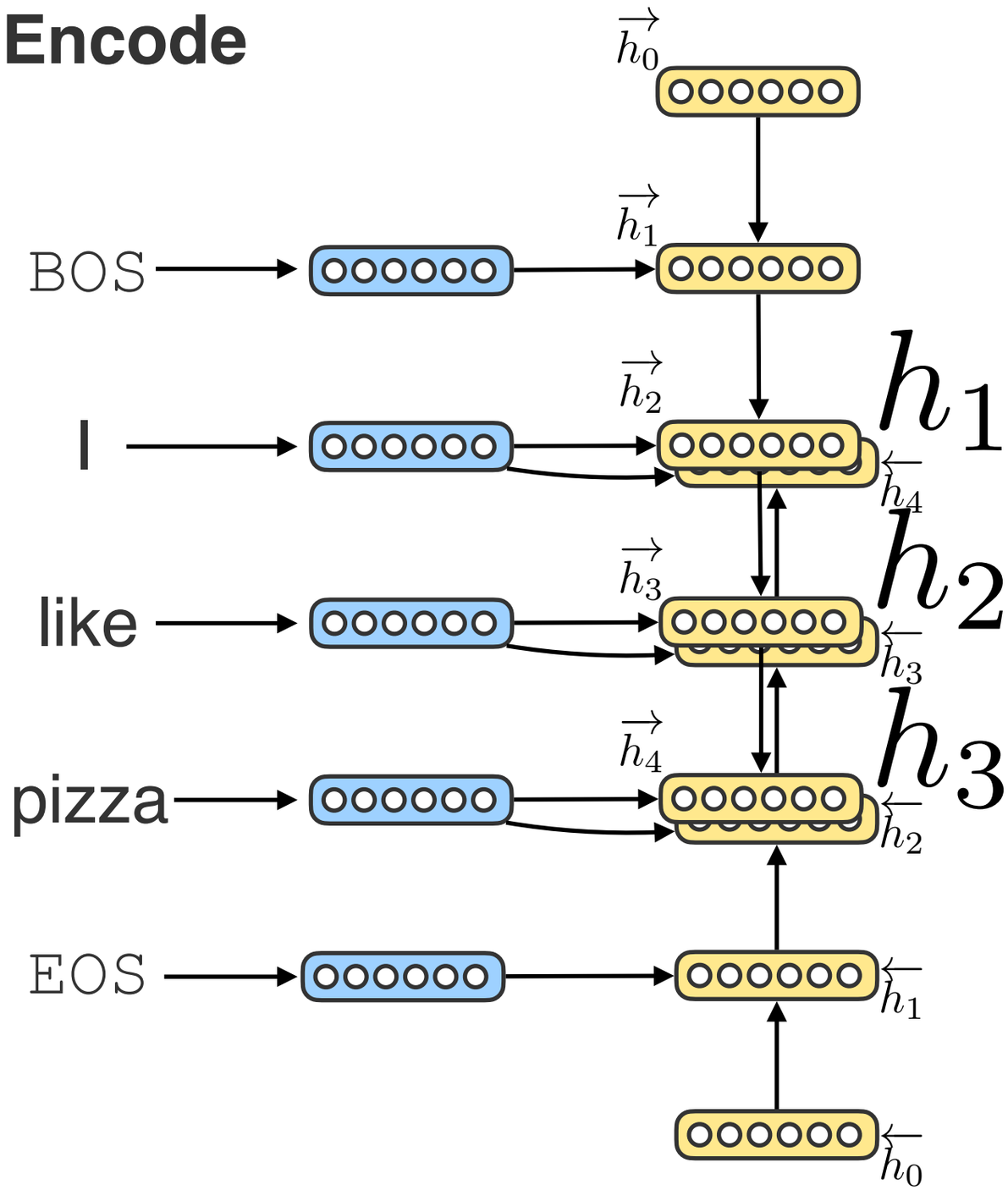


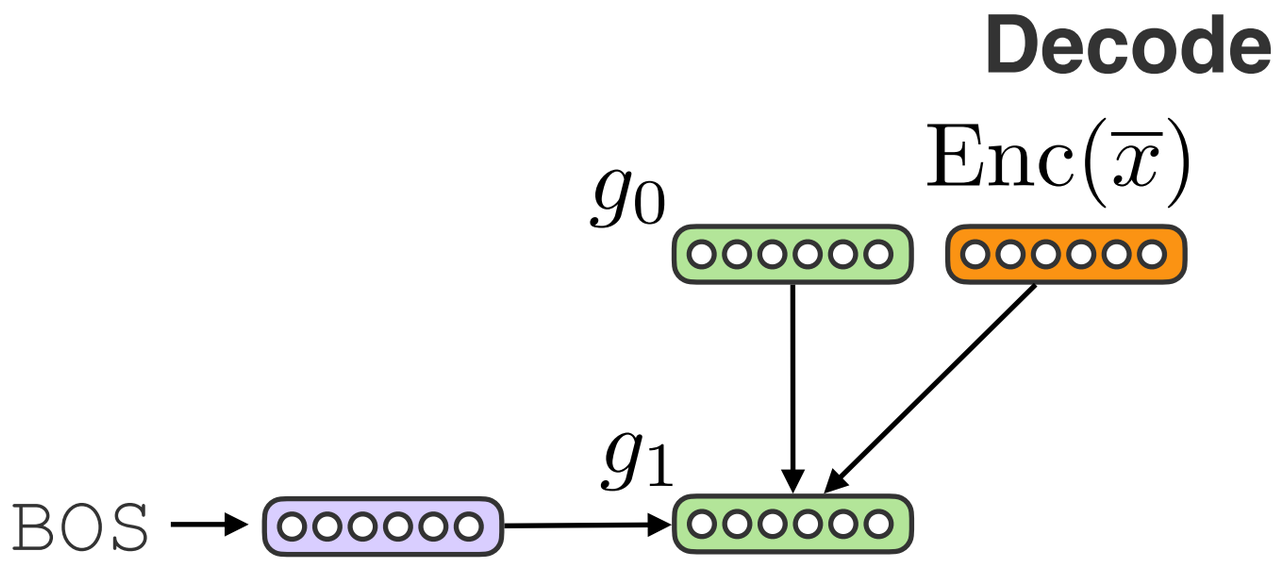
$$\mathrm{Enc}(\overline{x}, g) = \sum_{i=1}^{|\overline{x}|} p(i \mid g) h_i$$

$$g \in \mathbb{R}^d \quad p : \mathbb{R}^d \rightarrow \Delta^{\mathbb{N}_{1:|\overline{x}|}}$$

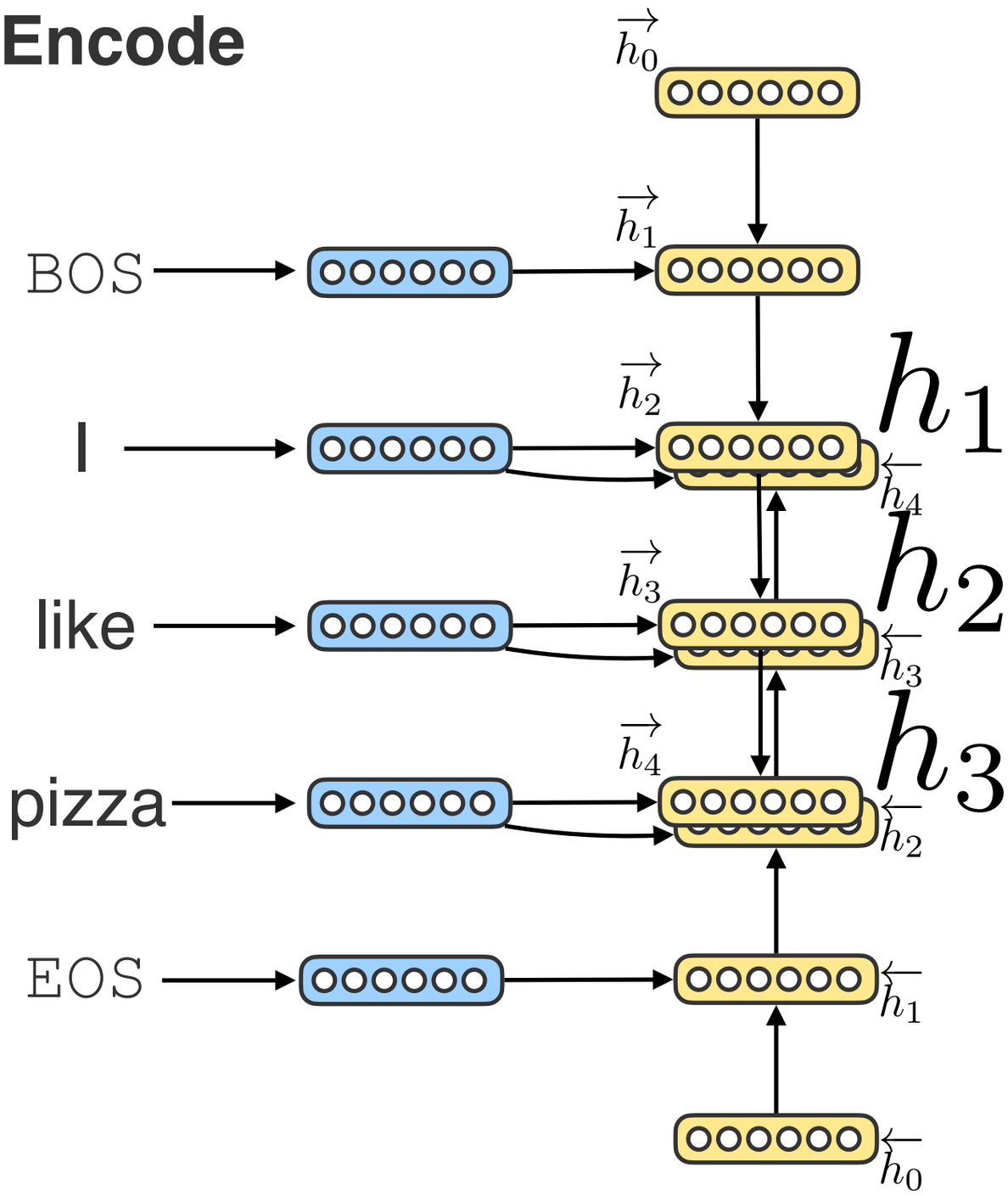
- Compute a new distribution over input indices depending on some other vector *g*
- Two questions:
  - Where does *g* come from?
  - How to implement *p*?

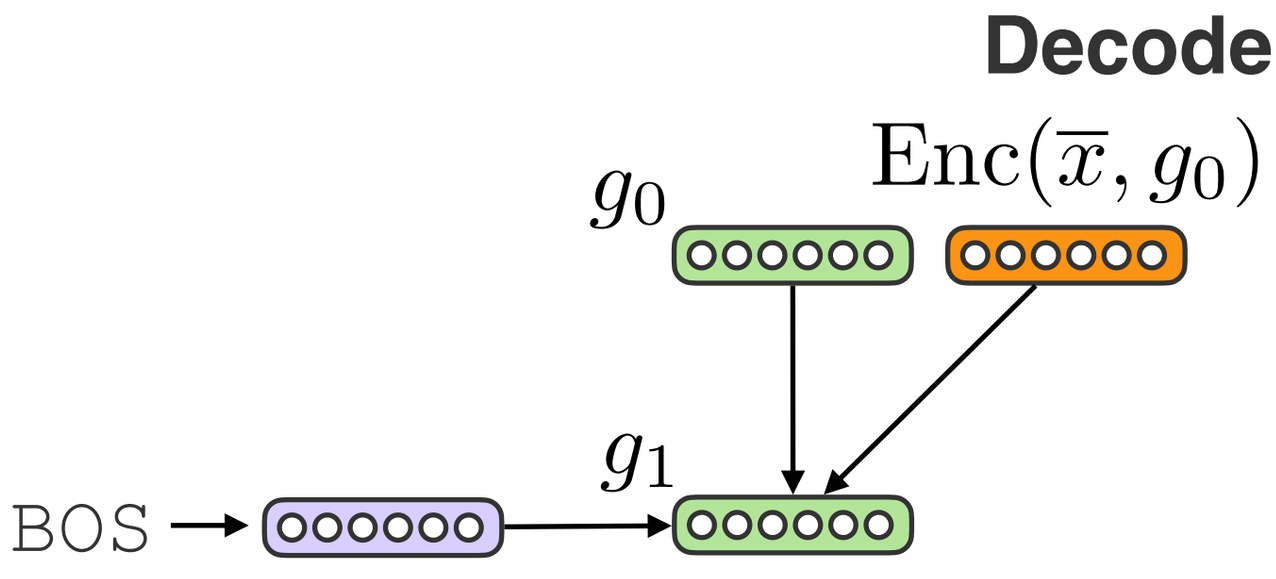
Recap: Attention





Recap: Attention





$$g_i = f(g_{i-1}, \mathrm{Enc}(\overline{x}, g_{i-1}), \phi(y_i))$$

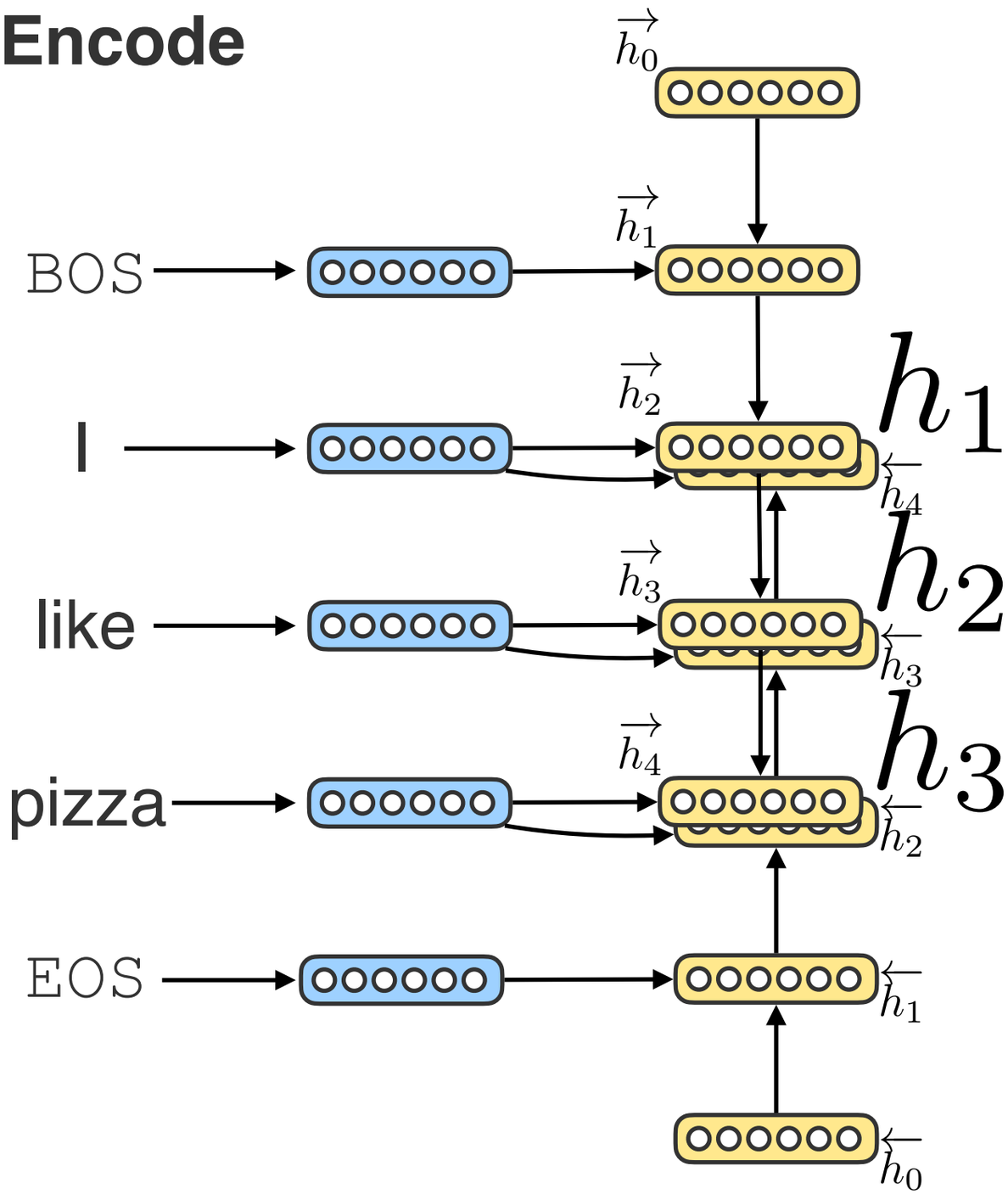
text encoding also depends on the previous state!

Recall:

$$\mathrm{Enc}(\overline{x}, g) = \sum_{i=1}^{|\overline{x}|} p(i \mid g) h_i$$

Recap: Attention

Encode **Decode**



h_1 h_2 h_3

$p(I \mid g_0)$

60%

35%

5

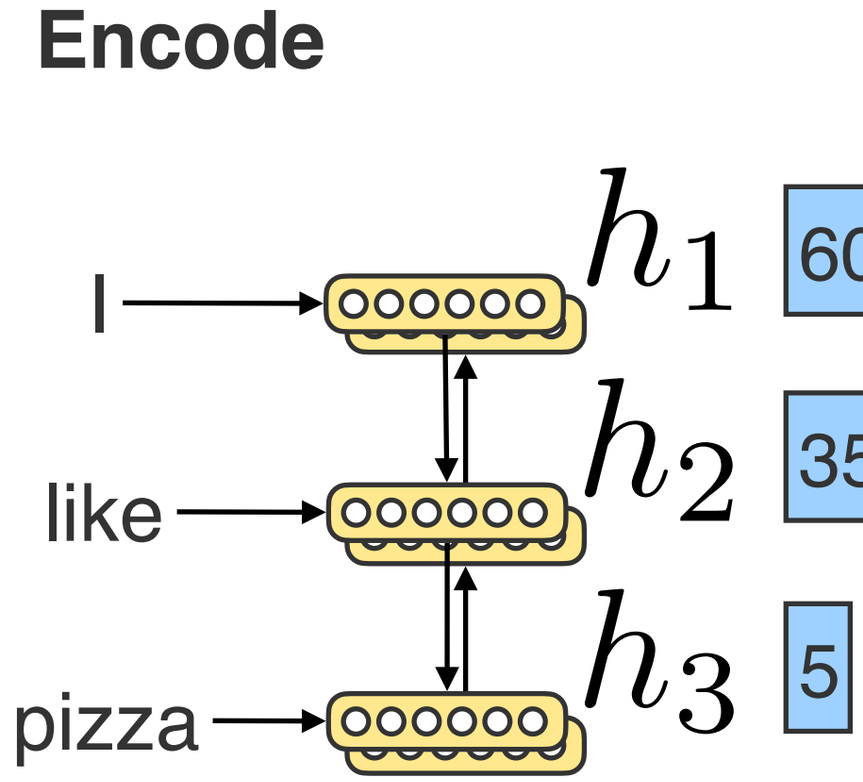
Weighted sum:

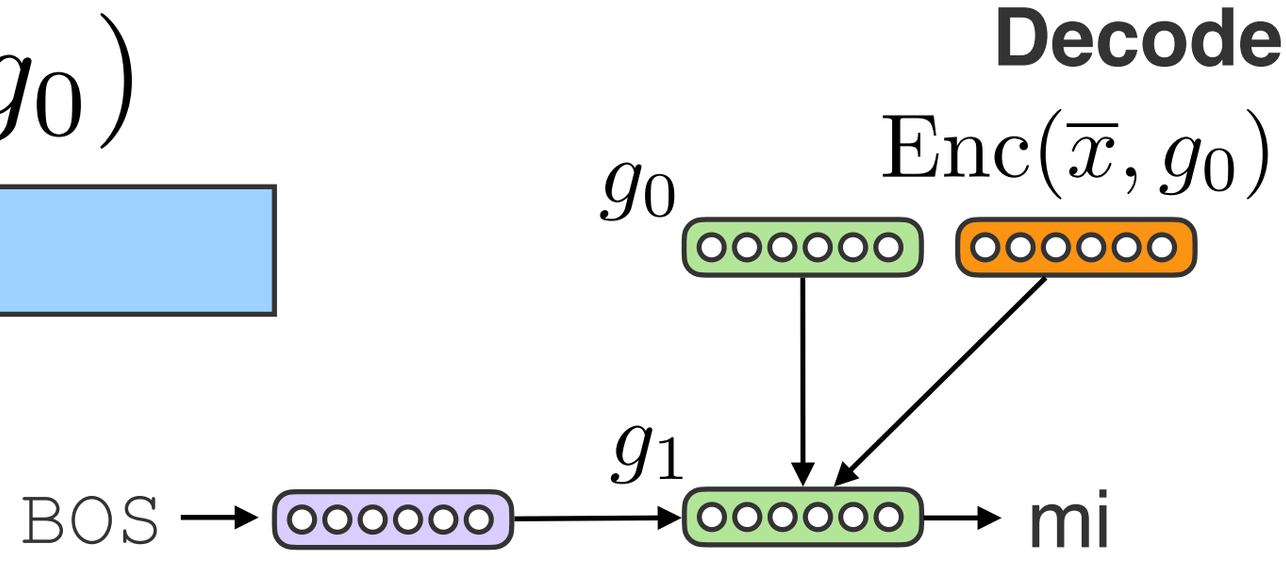
$$\mathrm{Enc}(\overline{x}, g_0) = \sum_{i=1}^{|\overline{x}|} p(i \mid g_0) h_i$$

Recall:

$$\mathrm{Enc}(\overline{x}, g) = \sum_{i=1}^{|\overline{x}|} p(i \mid g) h_i$$

Recap: Attention



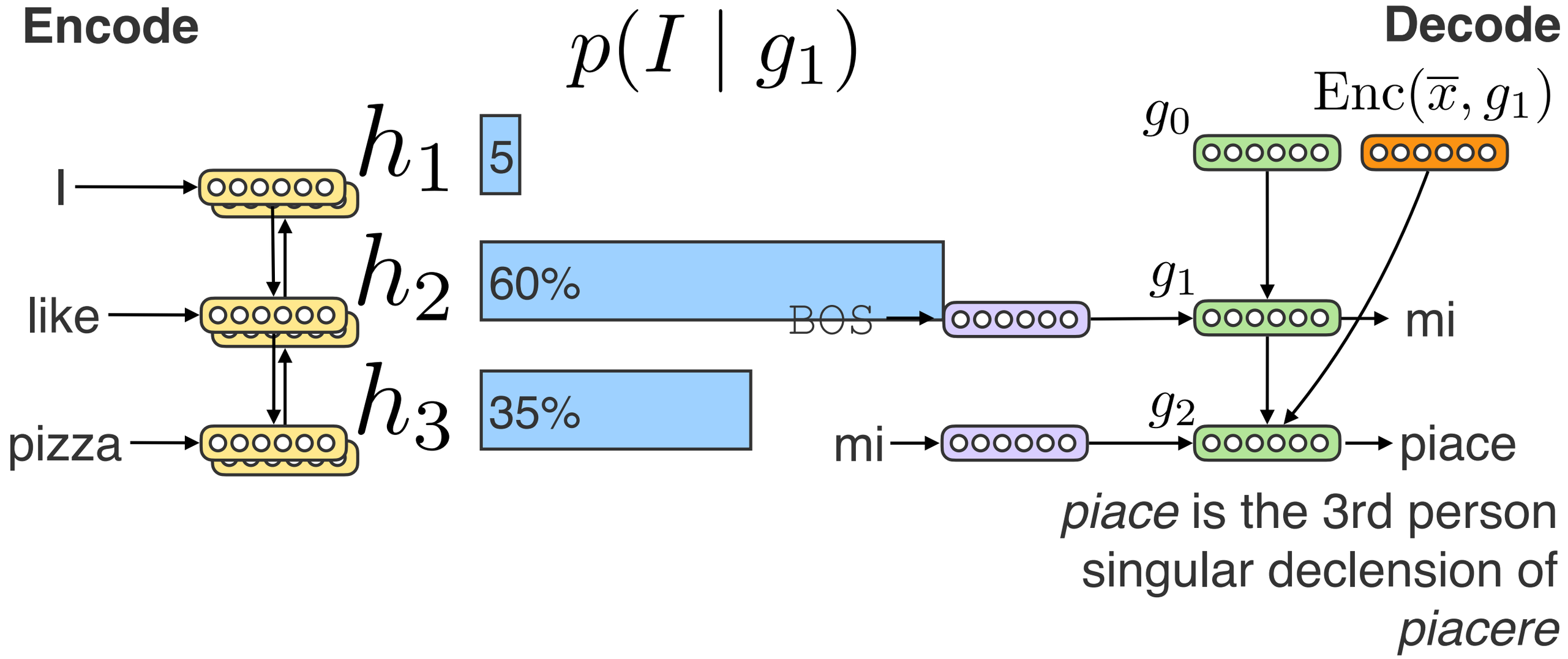


mi is the 1st person, objective case pronoun

Recall:

$$\mathrm{Enc}(\overline{x}, g) = \sum_{i=1}^{|\overline{x}|} p(i \mid g) h_i$$

Recap: Attention



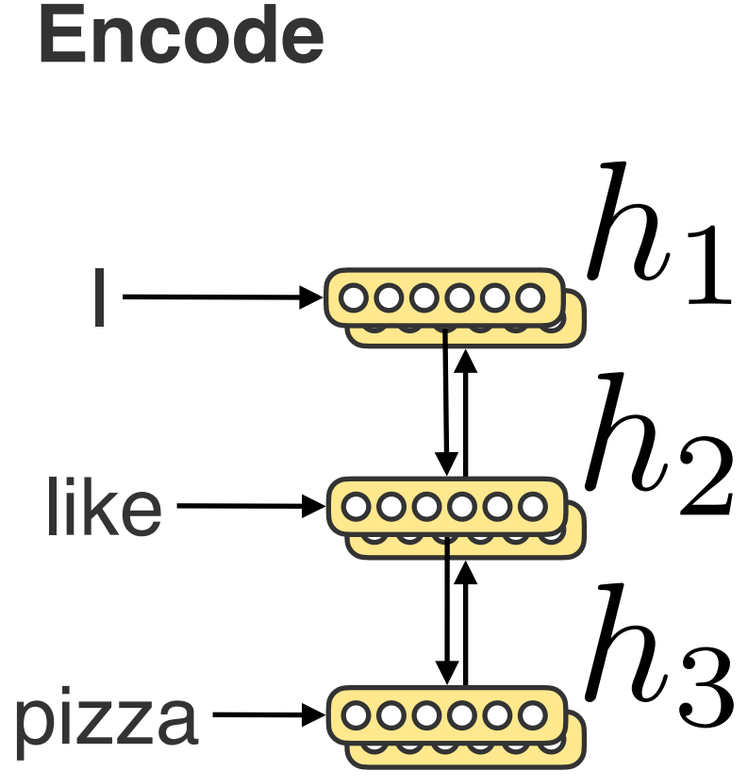
piace is the 3rd person singular declension of *piacere*

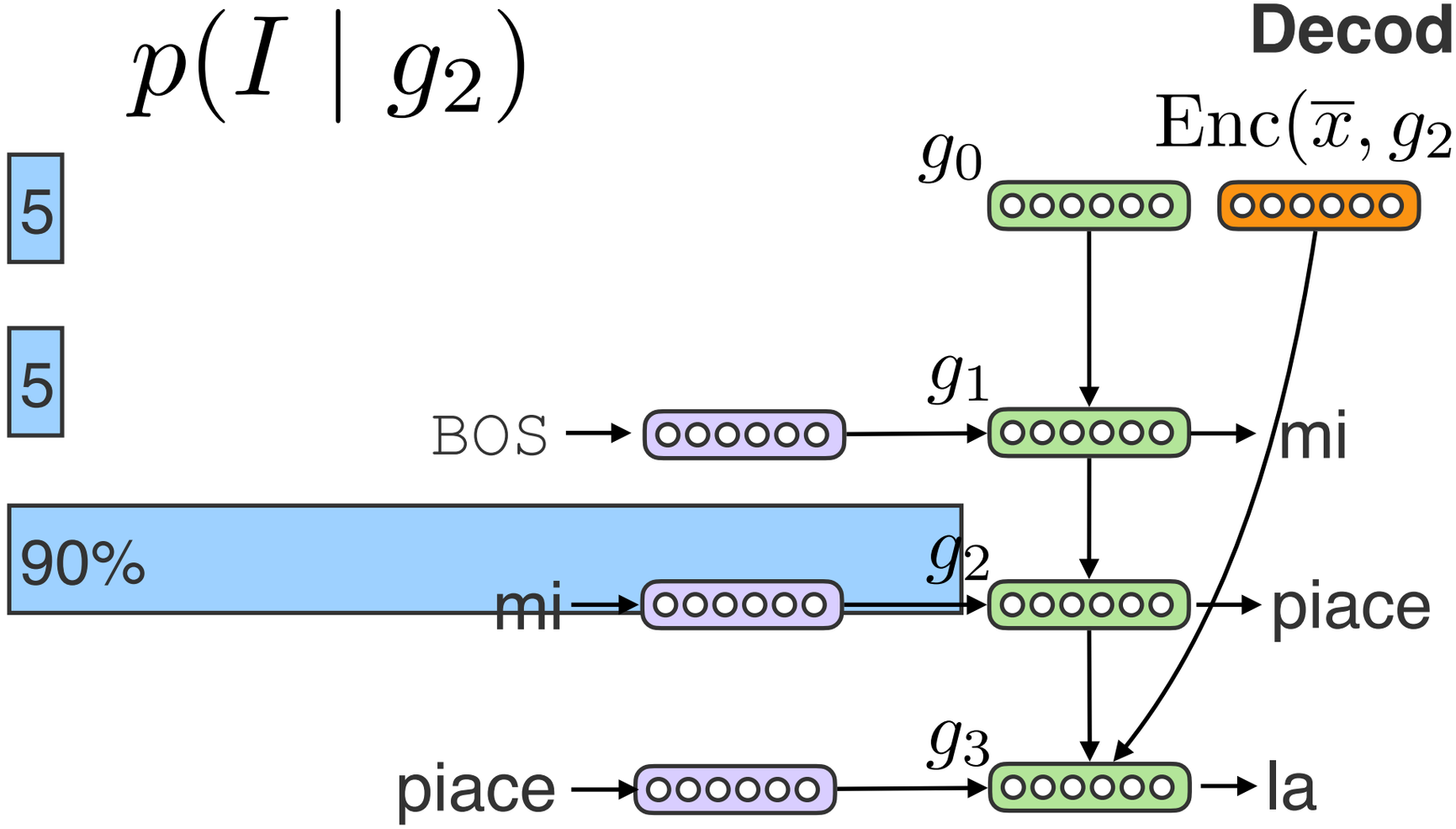
Recall:

$$\mathrm{Enc}(\overline{x}, g) = \sum_{i=1}^{|\overline{x}|} p(i \mid g) h_i$$

Recap: Attention

$$p(I \mid g_2)$$





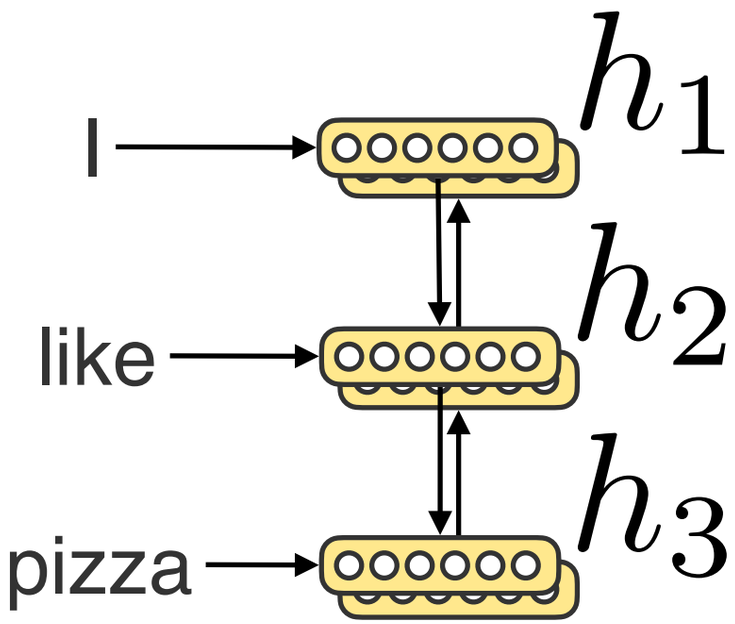
Recall:

$$\text{Enc}(\overline{x}, g) = \sum_{i=1}^{|\overline{x}|} p(i \mid g) h_i$$

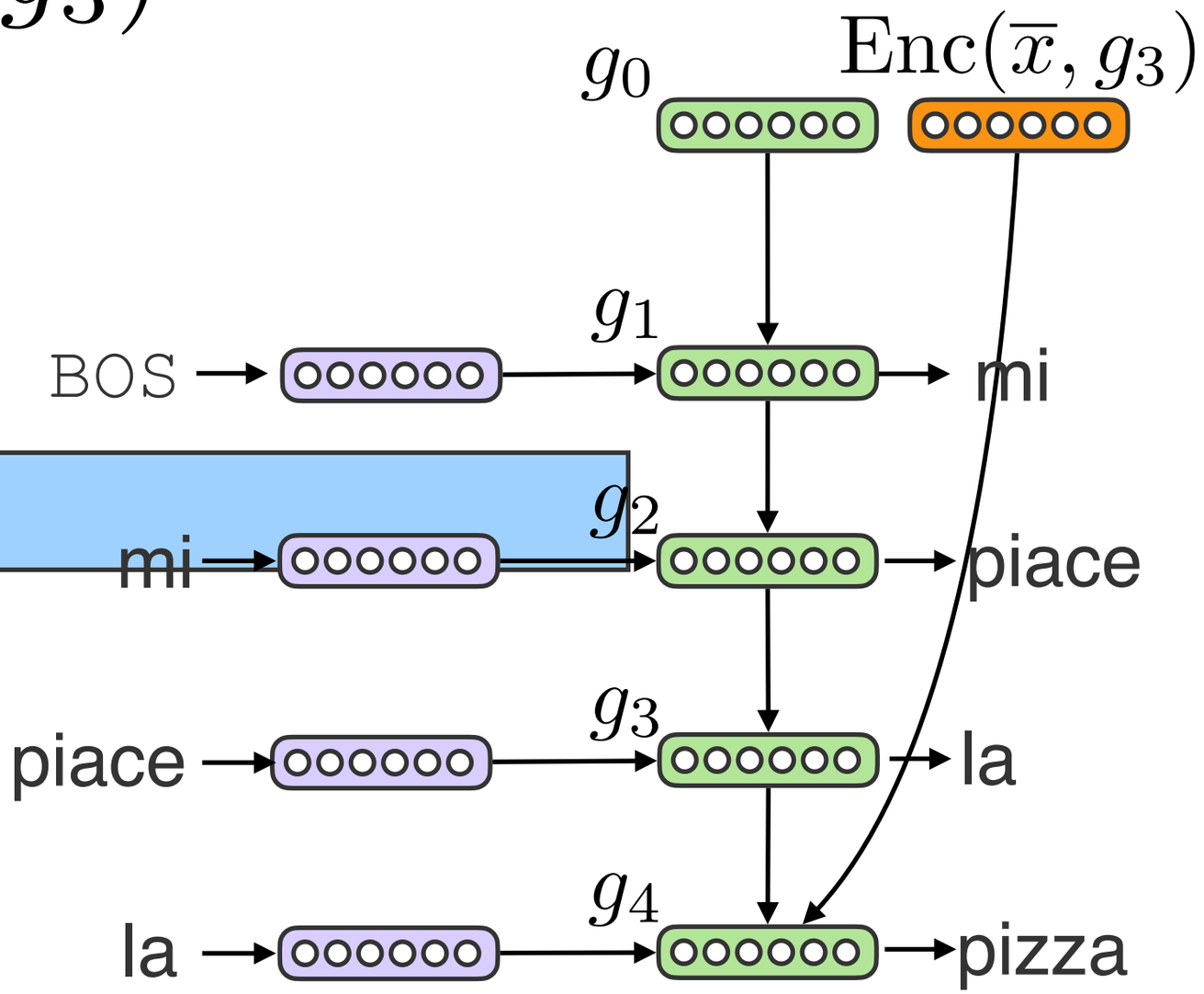
Recap: Attention

$$p(I \mid g_3)$$

Encode



Decode



Recall:

$$\mathrm{Enc}(\overline{x}, g) = \sum_{i=1}^{|\overline{x}|} p(i \mid g) h_i$$

Recap: Attention

- Update decoding hidden state based on an updated version of the input sequence encoding

$$g_i = f(g_{i-1}, \mathrm{Enc}(\overline{x}, g_{i-1}), \phi(y_i))$$

- The updated sequence is determined via a weighted sum over input hidden states

$$\mathrm{Enc}(\overline{x}, g) = \sum_{i=1}^{|\overline{x}|} p(i \mid g) h_i \qquad p : \mathbb{R}^d \to \Delta^{\mathbb{N}_{1:|\overline{x}|}}$$

- Next time: how do we compute $p(I \mid g)$?

Recap: Attention

- We encoded our input sequence into hidden states: $h_1, \ldots, h_{|\overline{x}|} = \mathbf{h} \in \mathbb{R}^{d \times |\overline{x}|}$

Recap: Attention

- We encoded our input sequence into hidden states: $h_1, \ldots, h_{|\overline{x}|} = \mathbf{h} \in \mathbb{R}^{d \times |\overline{x}|}$
- Now we want to predict the next word y_{i+1}

Recap: Attention

- We encoded our input sequence into hidden states: $h_1, \ldots, h_{|\overline{x}|} = \mathbf{h} \in \mathbb{R}^{d \times |\overline{x}|}$
- Now we want to predict the next word y_{i+1}
- We have access to the previous decoder hidden state g_i

Recap: Attention

- We encoded our input sequence into hidden states: $h_1, \ldots, h_{|\overline{x}|} = \mathbf{h} \in \mathbb{R}^{d \times |\overline{x}|}$
- Now we want to predict the next word y_{i+1}
- We have access to the previous decoder hidden state g_i
- First, compute attention scores for each input hidden state using similarity between query (g_i) and keys ($\mathbf{h}$): $s_i = a(g_i, \mathbf{h}) \in \mathbb{R}^{|\overline{x}|}$

Recap: Attention

- We encoded our input sequence into hidden states: $h_1, \ldots, h_{|\overline{x}|} = \mathbf{h} \in \mathbb{R}^{d \times |\overline{x}|}$
- Now we want to predict the next word y_{i+1}
- We have access to the previous decoder hidden state g_i
- First, compute attention scores for each input hidden state using similarity between query (g_i) and keys ($\mathbf{h}$): $s_i = a(g_i, \mathbf{h}) \in \mathbb{R}^{|\overline{x}|}$
- Then, take softmax of attention scores to get a distribution over keys:

$$\alpha_i = \text{softmax}(s_i) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$$

Recap: Attention

- We encoded our input sequence into hidden states: $h_1, \ldots, h_{|\overline{x}|} = \mathbf{h} \in \mathbb{R}^{d \times |\overline{x}|}$
- Now we want to predict the next word y_{i+1}
- We have access to the previous decoder hidden state g_i
- First, compute attention scores for each input hidden state using similarity between query (g_i) and keys ($\mathbf{h}$): $s_i = a(g_i, \mathbf{h}) \in \mathbb{R}^{|\overline{x}|}$
- Then, take softmax of attention scores to get a distribution over keys:

$$\alpha_i = \text{softmax}(s_i) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$$

- Finally, compute a weighted sum of values ($\mathbf{h}$) using this distribution

$$c_i = \sum_{j=1}^{|\overline{x}|} \alpha_{i,j} h_j \in \mathbb{R}^d$$

Recap: Attention

- We encoded our input sequence into hidden states: $h_1, \ldots, h_{|\overline{x}|} = \mathbf{h} \in \mathbb{R}^{d \times |\overline{x}|}$
- Now we want to predict the next word y_{i+1}
- We have access to the previous decoder hidden state g_i
- First, compute attention scores for each input hidden state using similarity between query (g_i) and keys ($\mathbf{h}$): $s_i = a(g_i, \mathbf{h}) \in \mathbb{R}^{|\overline{x}|}$
- Then, take softmax of attention scores to get a distribution over keys:

$$\alpha_i = \text{softmax}(s_i) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$$

- Finally, compute a weighted sum of values ($\mathbf{h}$) using this distribution

$$c_i = \sum_{j=1}^{|\overline{x}|} \alpha_{i,j} h_j \in \mathbb{R}^d$$

- Use the weighted sum to predict the next word:

$$p(Y_{i+1} \mid \overline{x}, y_1, \ldots, y_i) = \text{softmax}(f(c_i, g_i))$$

Recap: Attention

- We encoded our input sequence into hidden states: $h_1, \ldots, h_{|\overline{x}|} = \mathbf{h} \in \mathbb{R}^{d \times |\overline{x}|}$
- Now we want to predict the next word y_{i+1}
- We have access to the previous decoder hidden state g_i
- First, compute attention scores for each input hidden state using similarity between query (g_i) and keys ($\mathbf{h}$): $s_i = a(g_i, \mathbf{h}) \in \mathbb{R}^{|\overline{x}|}$
- Then, take softmax of attention scores to get a distribution over keys:

$$\alpha_i = \text{softmax}(s_i) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$$

- Finally, compute a weighted sum of values ($\mathbf{h}$) using this distribution

$$c_i = \sum_{j=1}^{|\overline{x}|} \alpha_{i,j} h_j \in \mathbb{R}^d$$

- Use the weighted sum to predict the next word:

$$p(Y_{i+1} \mid \overline{x}, y_1, \ldots, y_i) = \text{softmax}(f(c_i, g_i))$$

- Use the weighted sum to update the decoder hidden state: $g_i = g(g_{i-1}, c_{i-1}, y_i)$

Recap: Self-Attention

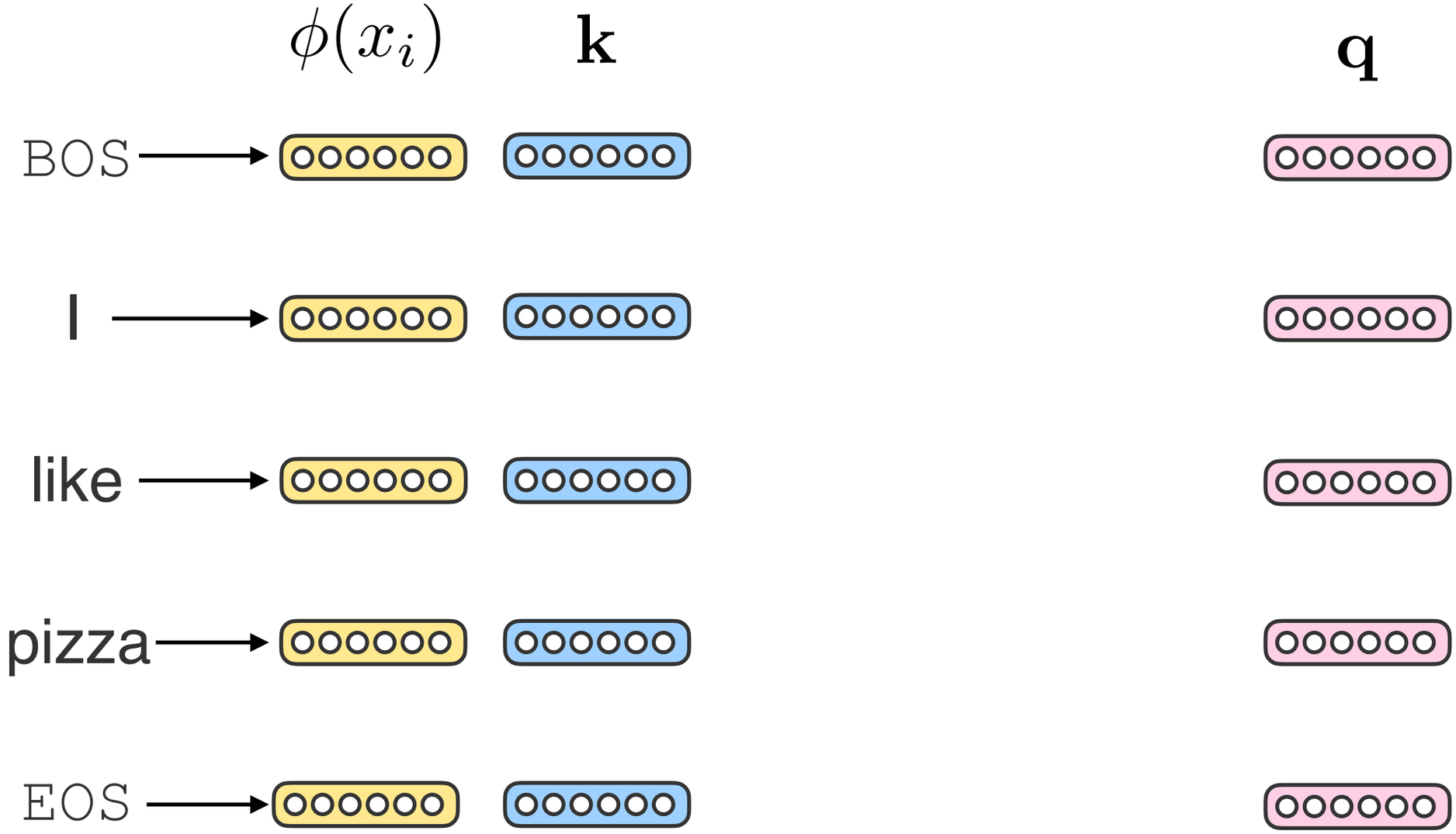
- For each token generated at index *i*, we have:
  - A key: $k_i = K\phi(x_i) \in \mathbb{R}^d \qquad K \in \mathbb{R}^{d \times d}$
  - A value: $v_i = V\phi(x_i) \in \mathbb{R}^d \quad V \in \mathbb{R}^{d \times d}$

- When trying to generate our next token at index *j*, we need a query: $q_j = Q\phi(x_j) \in \mathbb{R}^d \quad Q \in \mathbb{R}^{d \times d}$
  - Scores over keys: $s_{ji} = q_j^\intercal k_i$
  - Attention distribution over keys: $\alpha_{ji} = \frac{\exp(s_{ji})}{\sum_{i'=1}^{j} \exp(s_{ji'})}$
  - Weighted sum of values: $c_j = \sum_{i=1}^{j} \alpha_{ji} v_i$

$$x_{j+1} \sim p(X_{j+1} \mid x_1, \ldots, x_j) = \text{softmax}(f(c_j, \phi(x_j))$$

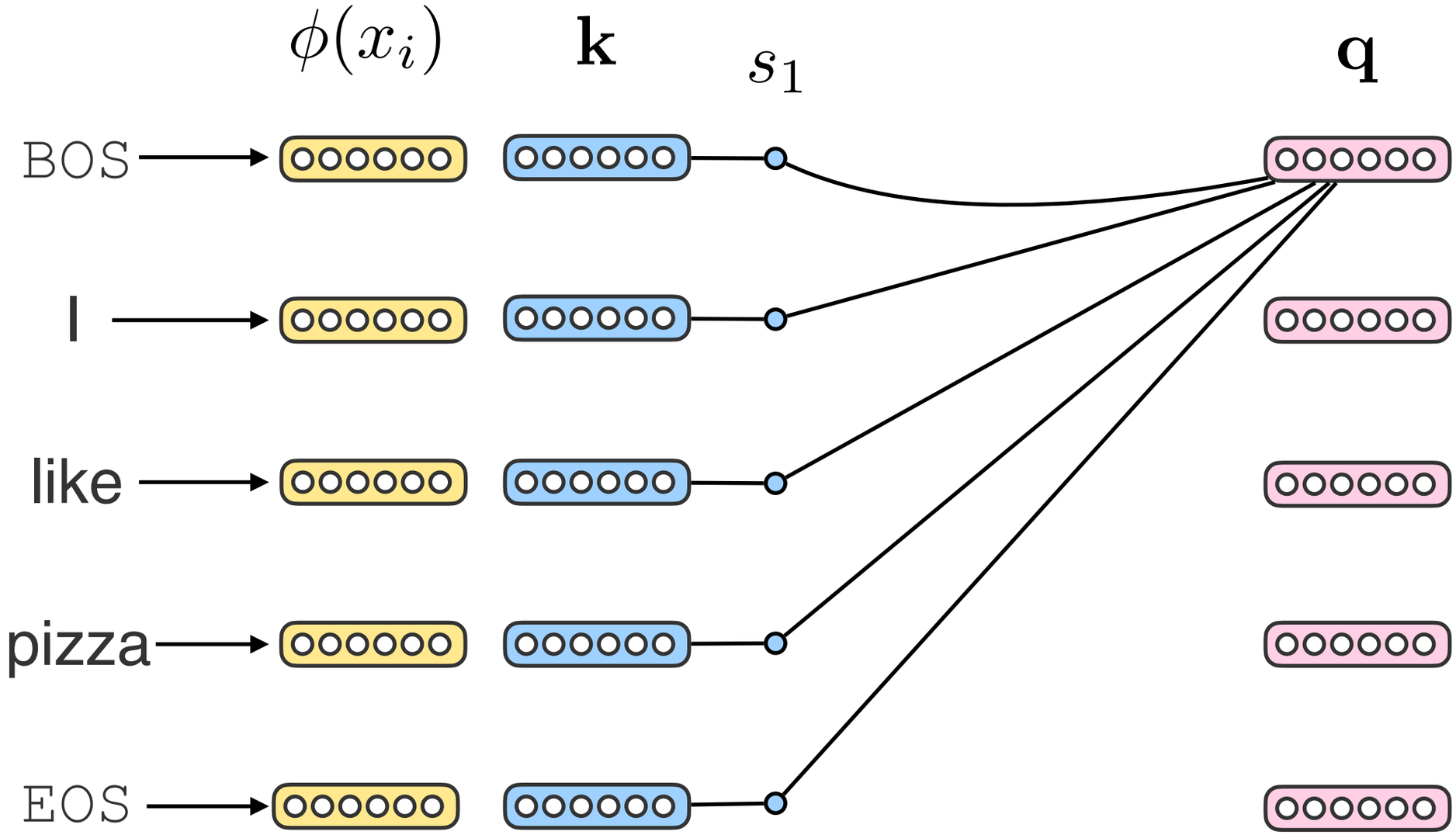
Self-Attention in Encoders

Encode



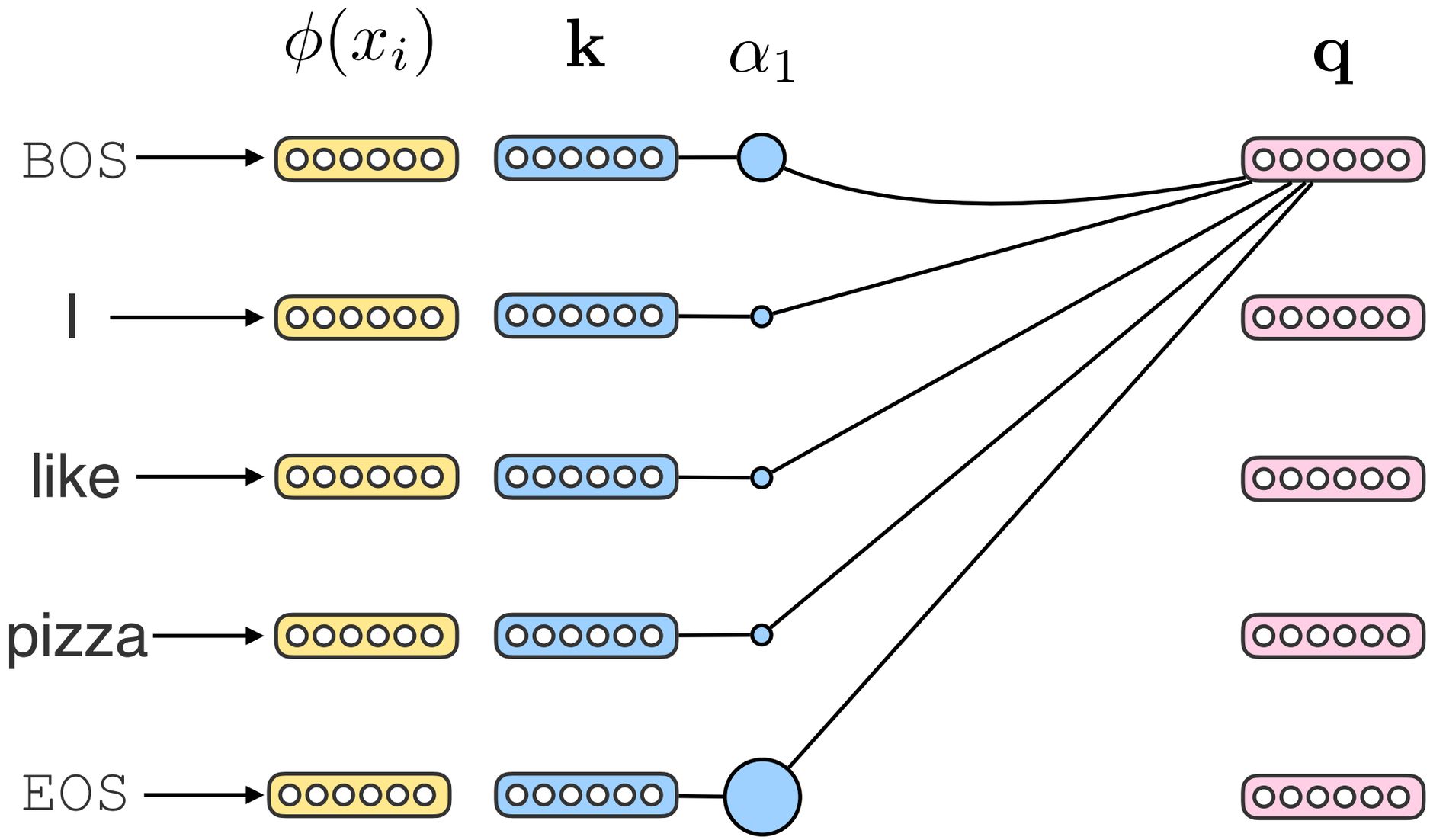
Self-Attention in Encoders

Encode



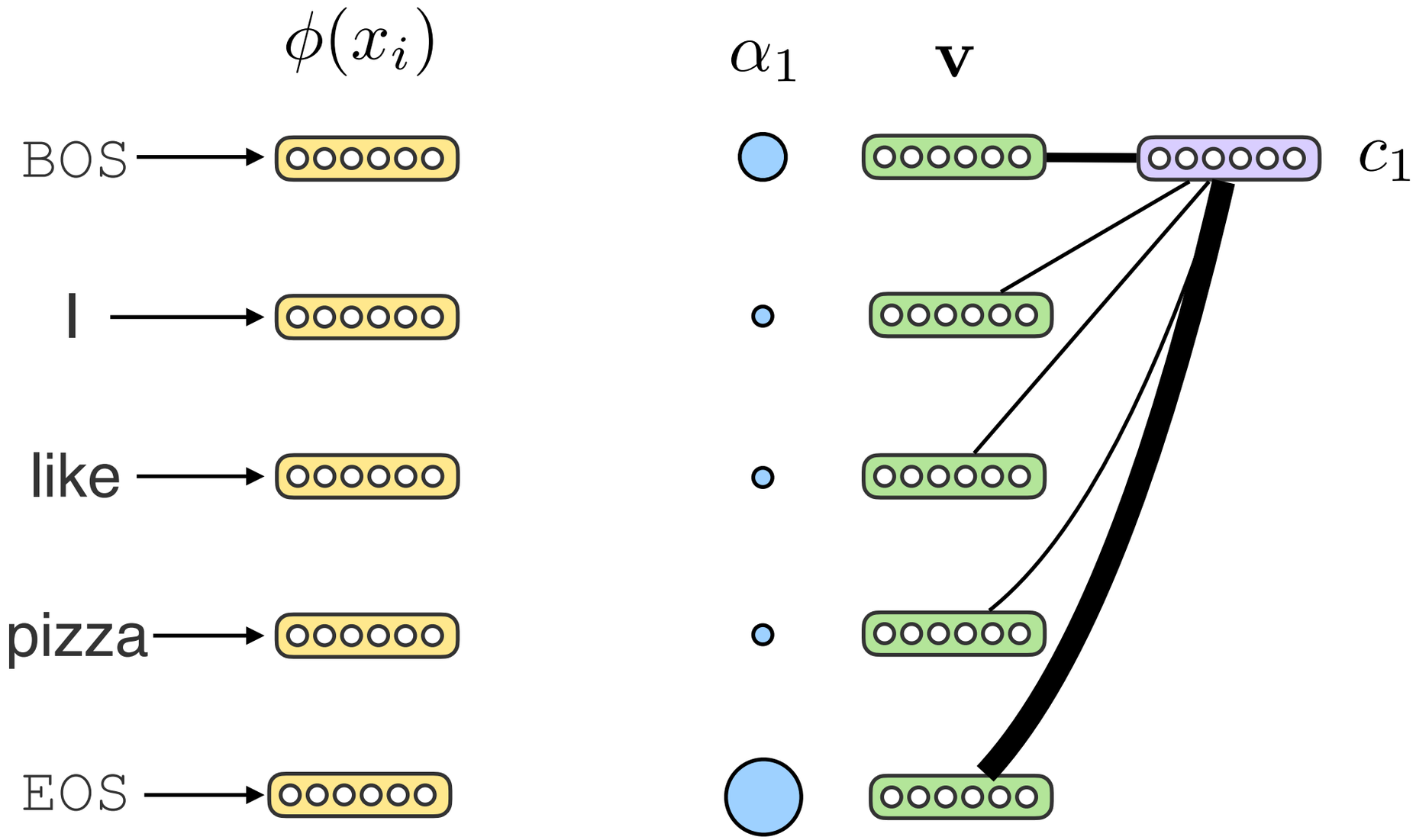
Self-Attention in Encoders

Encode



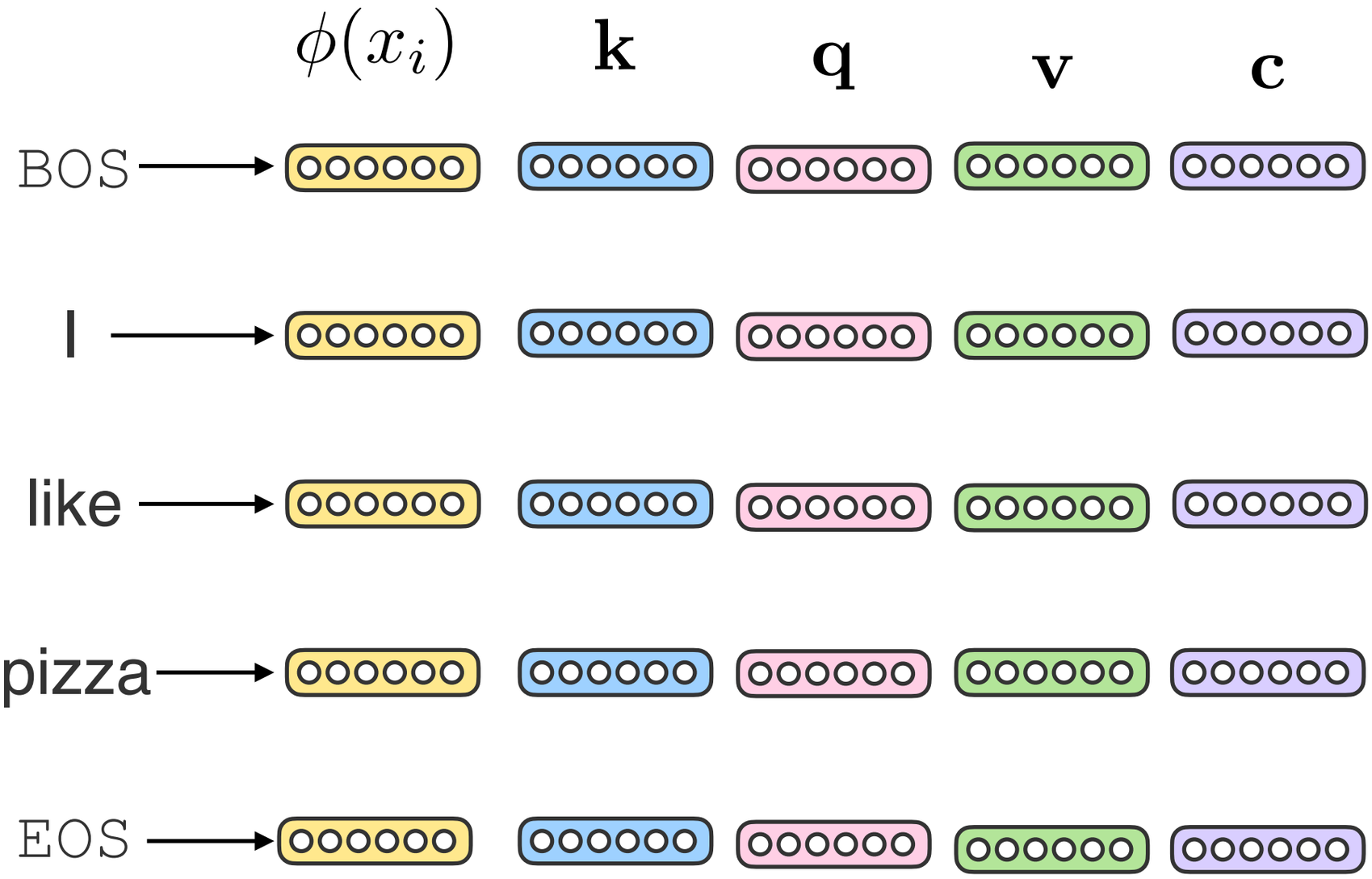
Self-Attention in Encoders

Encode



Self-Attention in Encoders

Encode



$$\mathbf{s} = \mathbf{q}^\intercal \mathbf{k} \in \mathbb{R}^{|\overline{x}| \times |\overline{x}|}$$

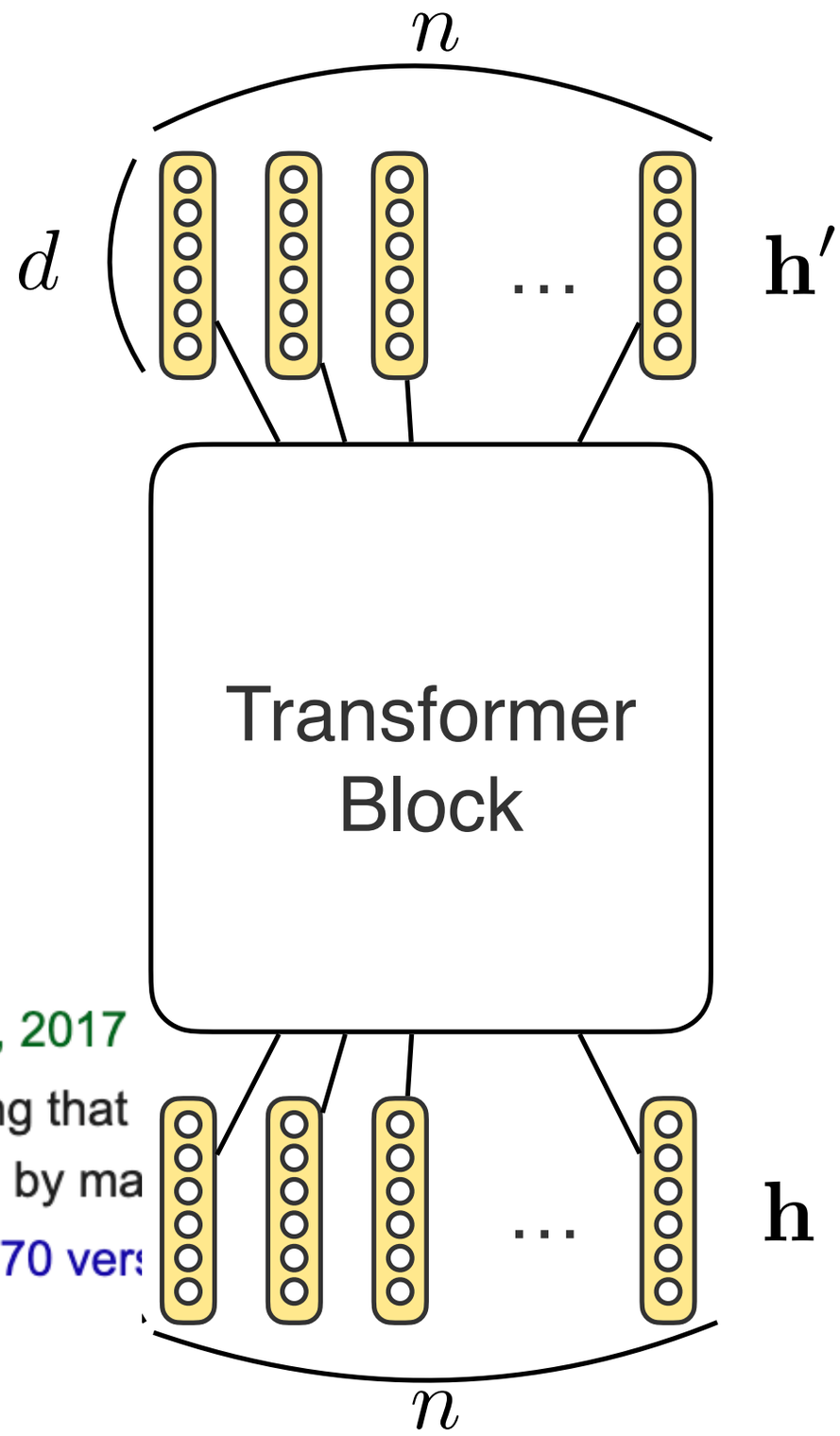
$$\alpha = \mathrm{softmax}_{\mathrm{dim}=1}(\mathbf{s}) \in \{\Delta^{\mathbb{N}_{1:|\overline{x}|}}\}^{|\overline{x}|}$$

$$\mathbf{c} = \alpha \mathbf{v} \in \mathbb{R}^{d \times |\overline{x}|}$$

We can get context-sensitive representations of each input token **in parallel**!

Building a Transformer Block

- A transformer block takes as input a sequence of input vectors $\mathbf{h} \in \mathbb{R}^{d \times n}$
- It generates a sequence of output vectors $\mathbf{h}' \in \mathbb{R}^{d \times n}$



Attention is all you need

A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017

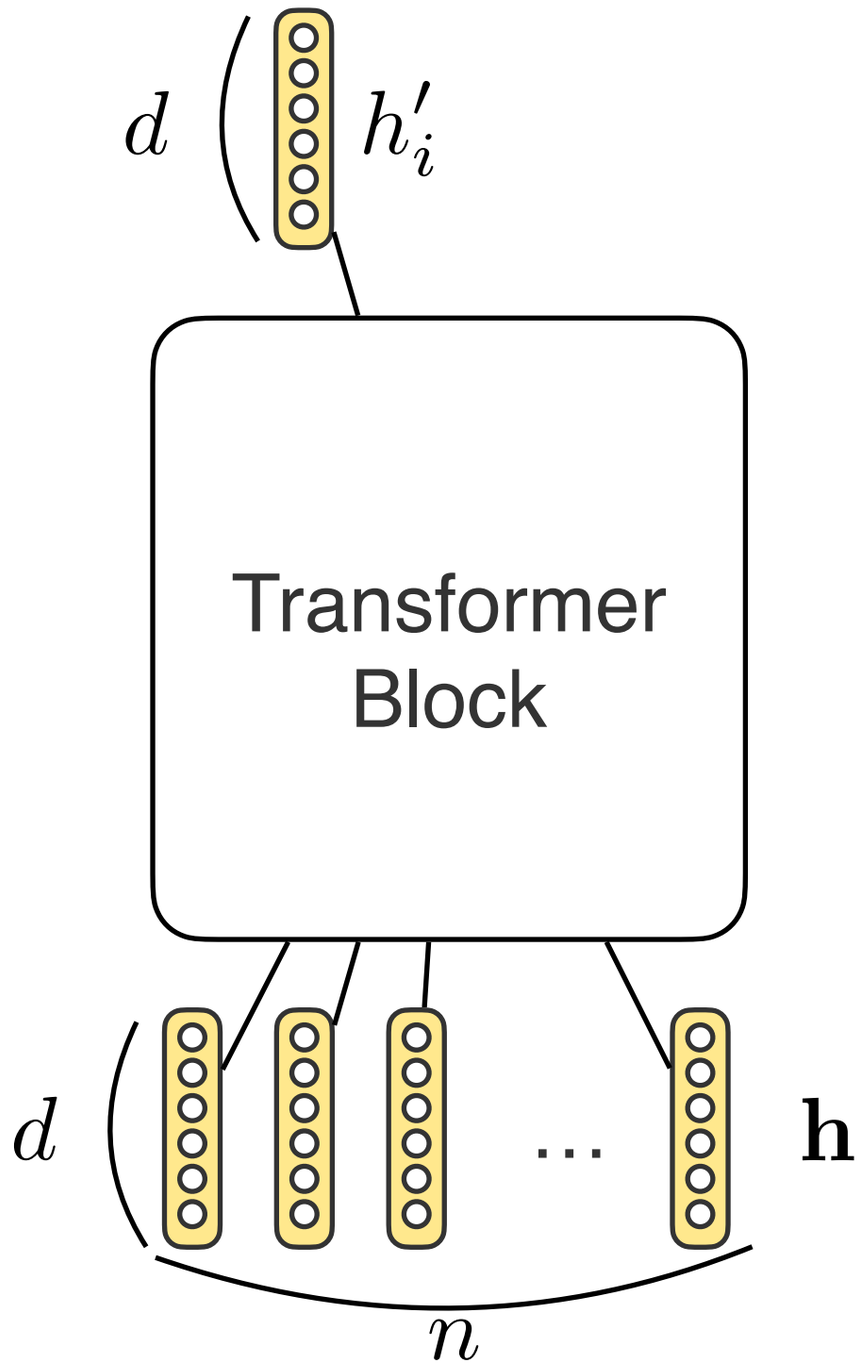
… to attend to **all** positions in the decoder up to and including that

… **We** implement this inside of scaled dot-product **attention** by ma

☆ Save 99 Cite Cited by 196986 Related articles All 70 vers

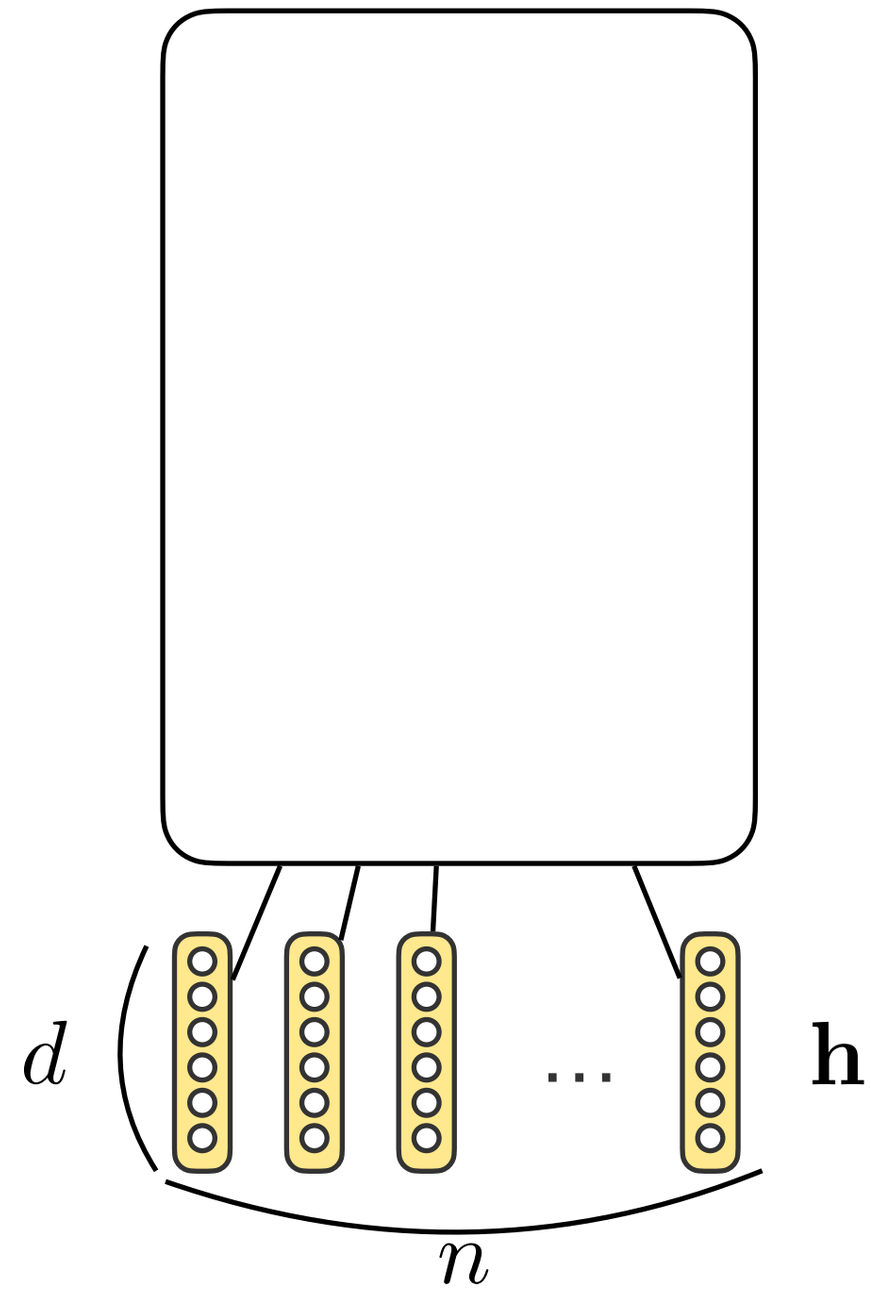
Building a Transformer Block

- A transformer block takes as input a sequence of input vectors $\mathbf{h} \in \mathbb{R}^{d \times n}$
- It generates a sequence of output vectors $\mathbf{h}' \in \mathbb{R}^{d \times n}$
- For now, let's focus on how it computes the output vector corresponding to the input at index *i*, $h'_i \in \mathbb{R}^d$



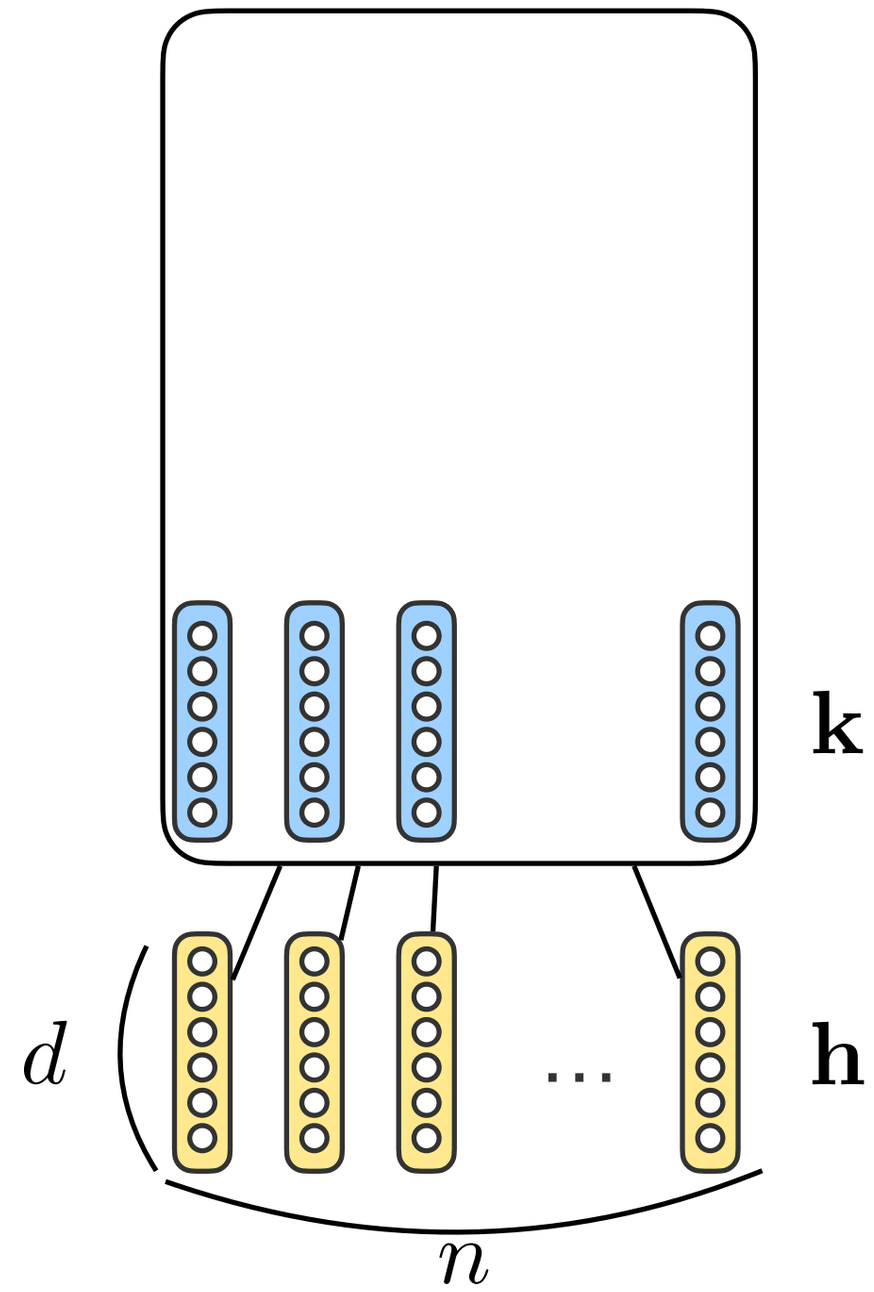
Self-Attention

1. Compute attention over input vectors



Self-Attention

1. Compute attention over input vectors
$\mathbf{k} = K\mathbf{h} \in \mathbb{R}^{d_k \times n}$

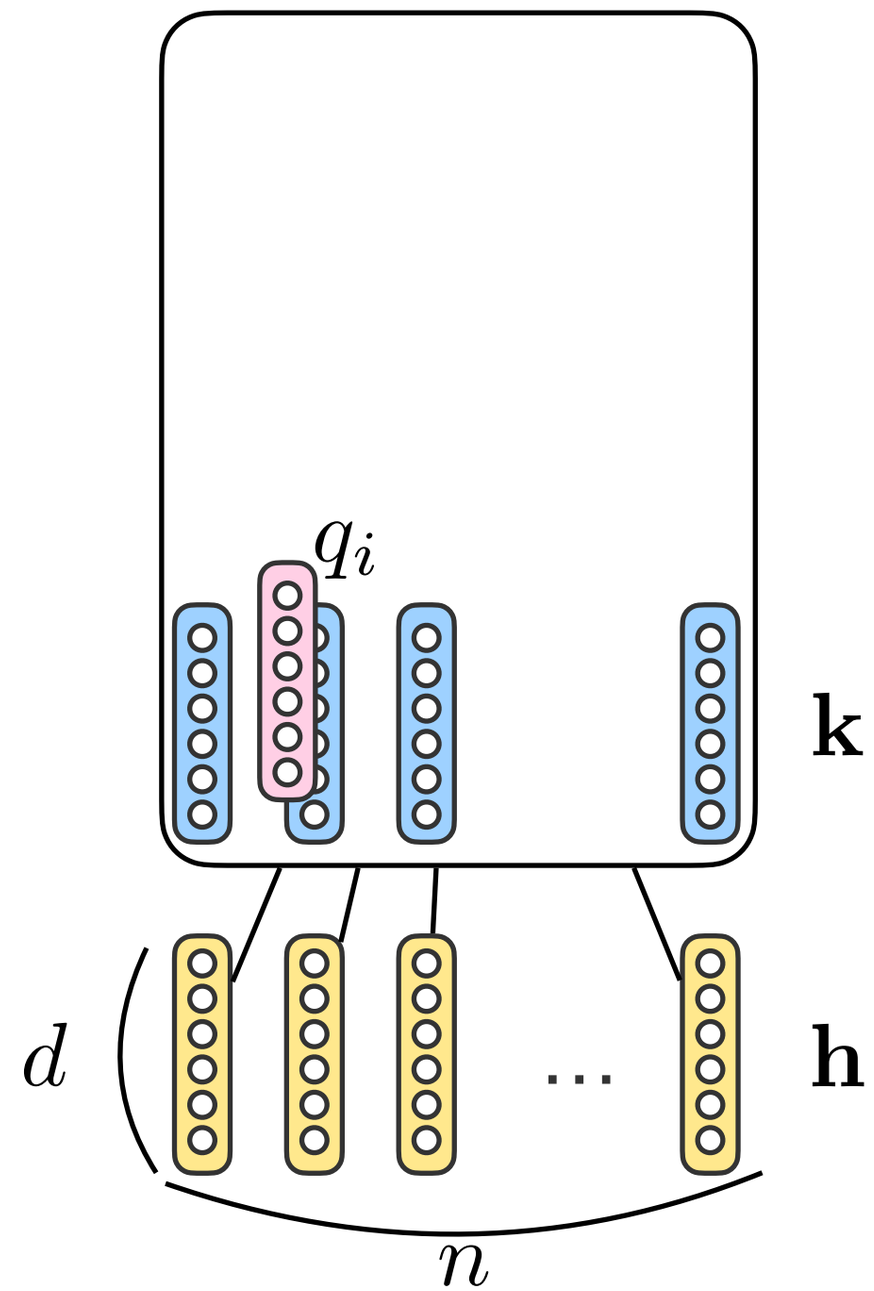


Self-Attention

1. Compute attention over input vectors

$$\mathbf{k} = K\mathbf{h} \in \mathbb{R}^{d_k \times n}$$

$$q_i = Qh_i \in \mathbb{R}^{d_k}$$



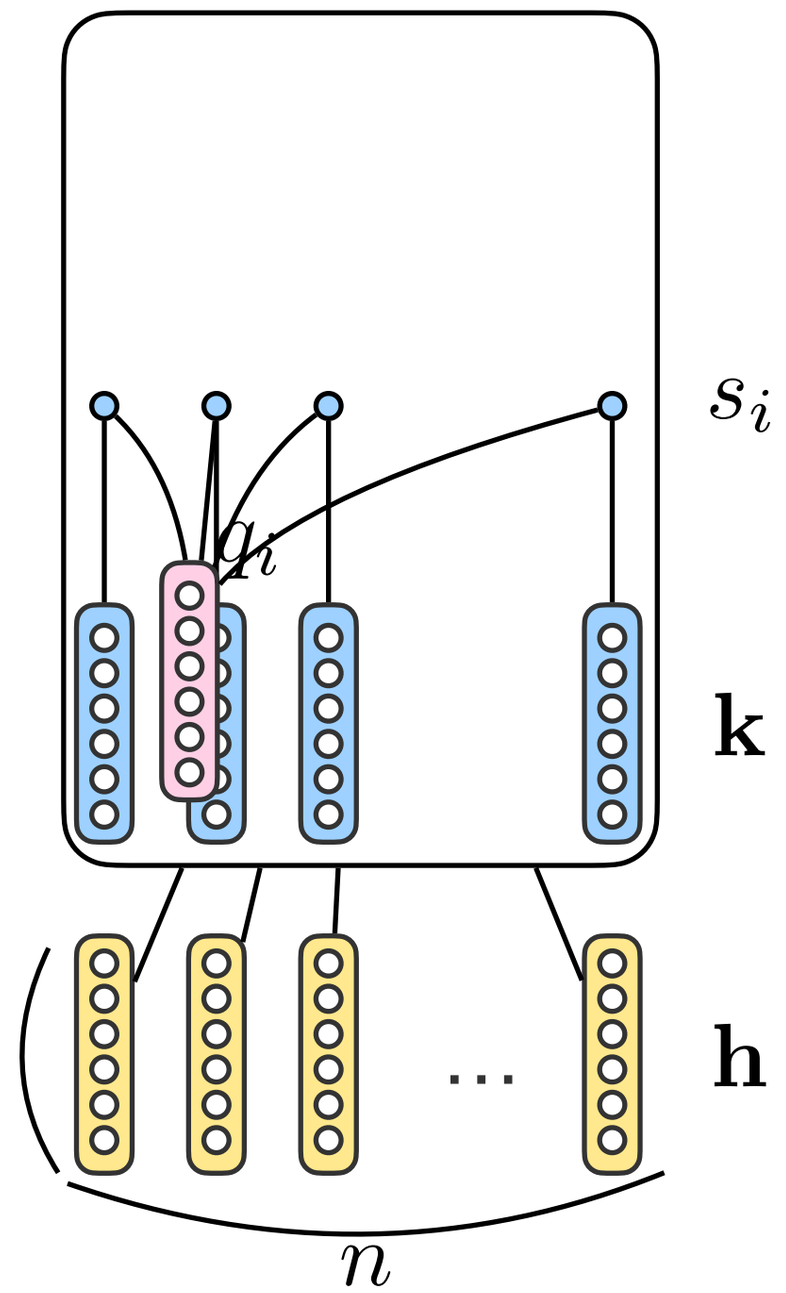
Self-Attention

1. Compute attention over input vectors

$$\mathbf{k} = K\mathbf{h} \in \mathbb{R}^{d_k \times n}$$

$$q_i = Qh_i \in \mathbb{R}^{d_k}$$

$$s_i = \frac{q_i^\intercal \mathbf{k}}{\sqrt{d_k}} \in \mathbb{R}^n$$



Scaling by vector dimension ensures the values don't become way too large.

Self-Attention

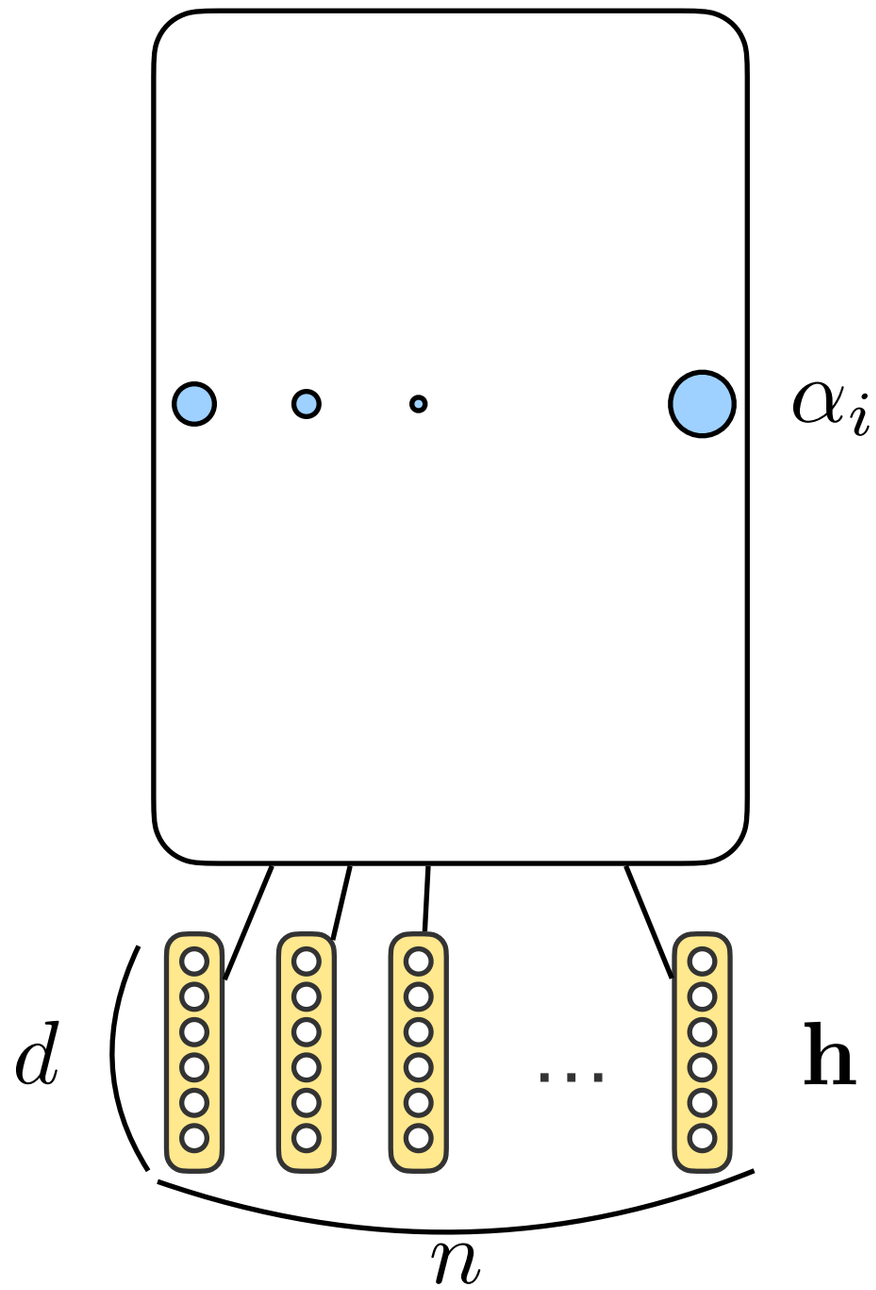
1. Compute attention over input vectors

$$\mathbf{k} = K\mathbf{h} \in \mathbb{R}^{d_k \times n}$$

$$q_i = Qh_i \in \mathbb{R}^{d_k}$$

$$s_i = \frac{q_i^\intercal \mathbf{k}}{\sqrt{d_k}} \in \mathbb{R}^n$$

$$\alpha_i = \text{softmax}\left(s_i\right) \in \Delta^{\mathbb{N}_{1:n}}$$



Self-Attention

1. Compute attention over input vectors

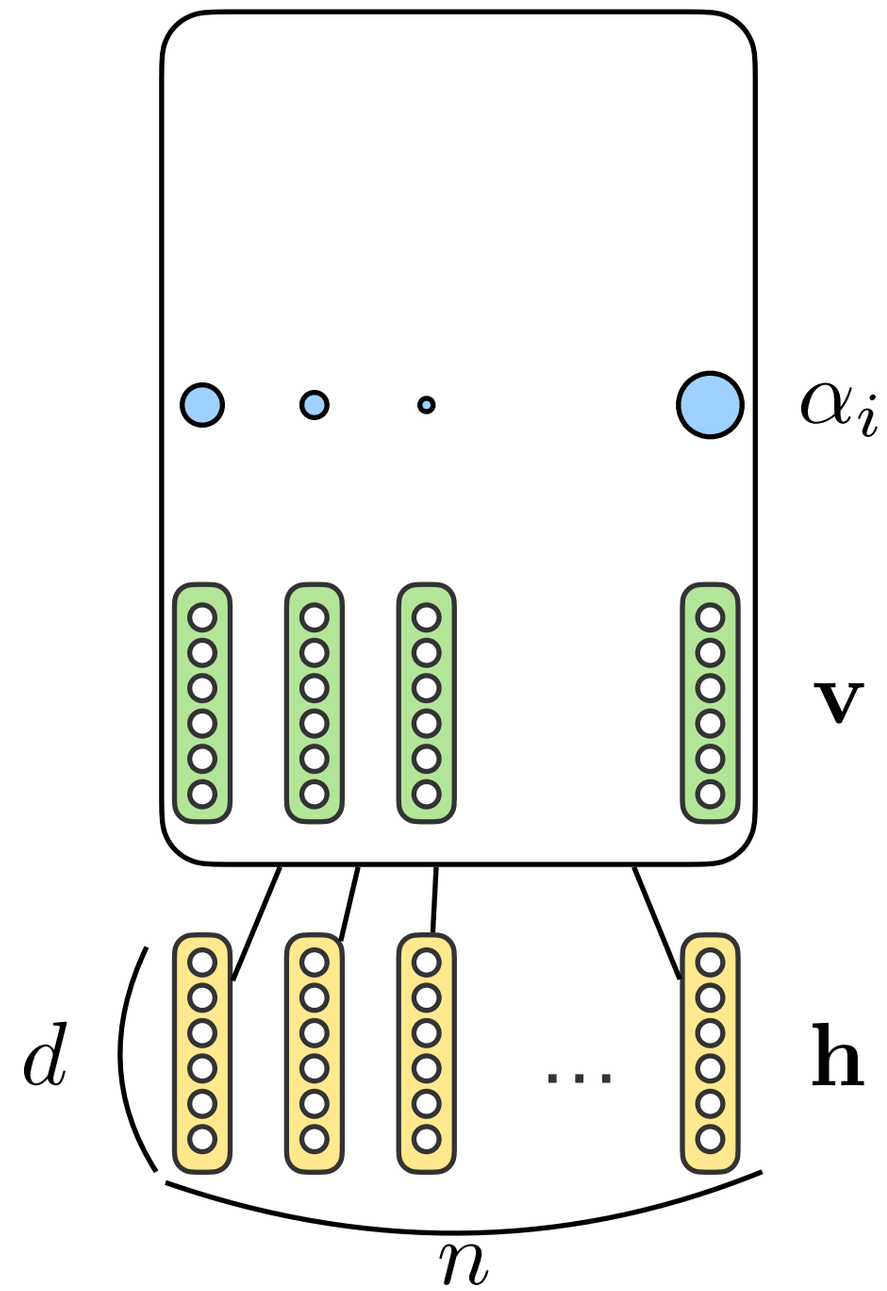
$$\mathbf{k} = K\mathbf{h} \in \mathbb{R}^{d_k \times n}$$

$$q_i = Qh_i \in \mathbb{R}^{d_k}$$

$$s_i = \frac{q_i^\intercal \mathbf{k}}{\sqrt{d_k}} \in \mathbb{R}^n$$

$$\alpha_i = \text{softmax}\left(s_i\right) \in \Delta^{\mathbb{N}_{1:n}}$$

$$\mathbf{v} = V\mathbf{h} \in \mathbb{R}^{d_k \times n}$$



Self-Attention

1. Compute attention over input vectors

$$\mathbf{k} = K\mathbf{h} \in \mathbb{R}^{d_k \times n}$$

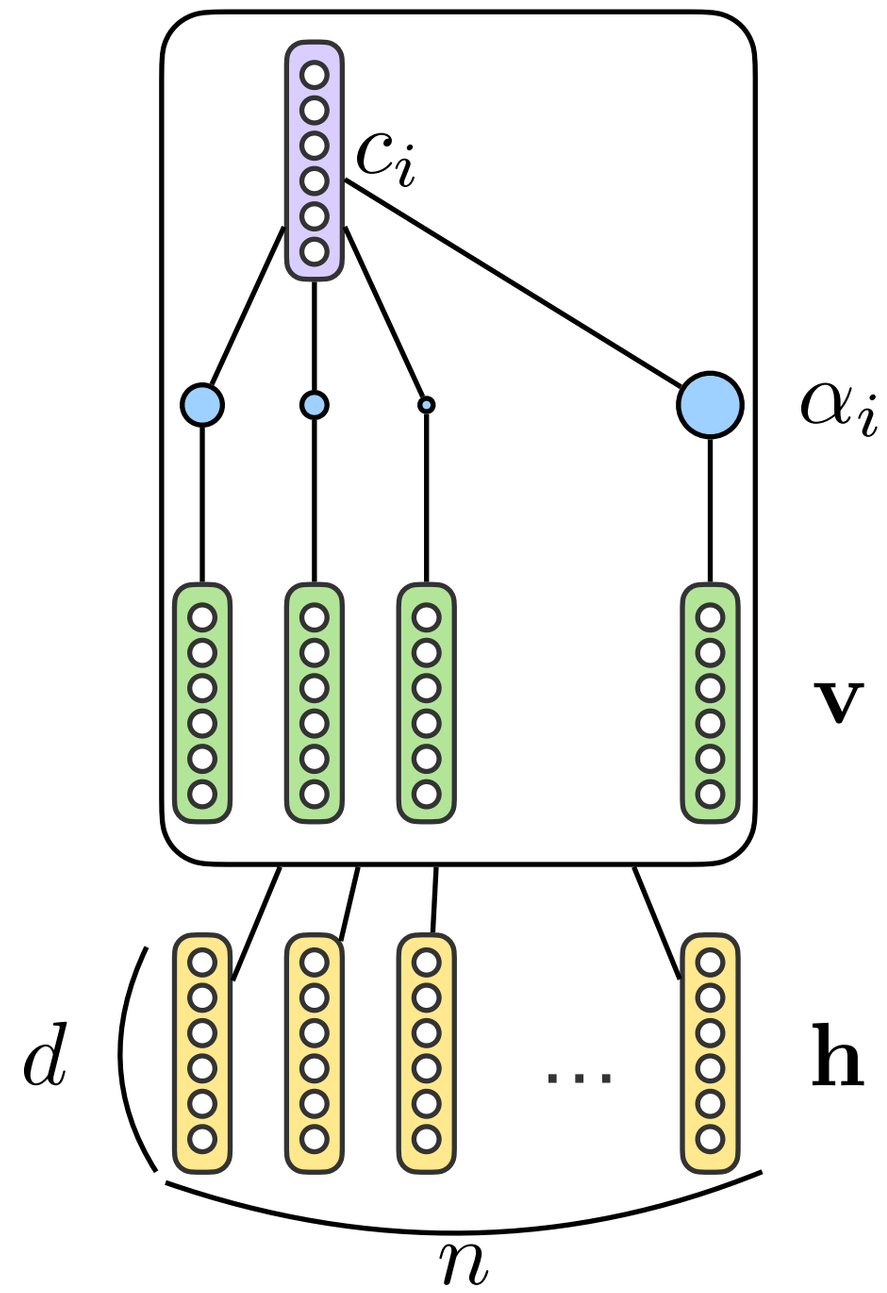
$$q_i = Qh_i \in \mathbb{R}^{d_k}$$

$$s_i = \frac{q_i^\intercal \mathbf{k}}{\sqrt{d_k}} \in \mathbb{R}^n$$

$$\alpha_i = \text{softmax}\left(s_i\right) \in \Delta^{\mathbb{N}_{1:n}}$$

$$\mathbf{v} = V\underset{n}{\mathbf{h}} \in \mathbb{R}^{d_k \times n}$$

$$c_i = \sum_{i'=1} \alpha_i v_{i'}$$



Self-Attention

1. Compute attention over input vectors

$$\mathbf{k} = K\mathbf{h} \in \mathbb{R}^{d_k \times n}$$

$$q_i = Qh_i \in \mathbb{R}^{d_k}$$

$$s_i = \frac{q_i^\intercal \mathbf{k}}{\sqrt{d_k}} \in \mathbb{R}^n$$

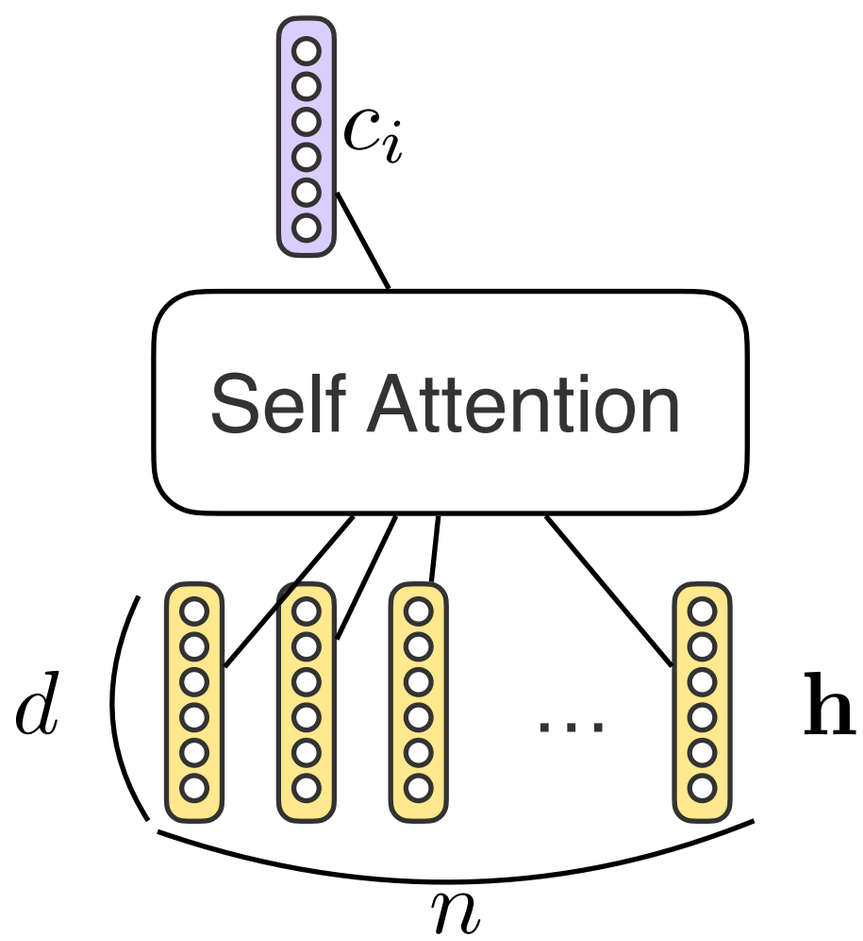
$$\alpha_i = \text{softmax}\left(s_i\right) \in \Delta^{\mathbb{N}_{1:n}}$$

$$\mathbf{v} = V\underset{n}{\mathbf{h}} \in \mathbb{R}^{d_k \times n}$$

$$c_i = \sum_{i'=1} \alpha_i v_{i'}$$

$$c_i = \text{SelfAttention}(\mathbf{h})_i$$

$$\theta_{\text{SelfAttention}} = \{K, Q, V\}$$



Multi-Head Attention

1. Compute attention over input vectors

$$\mathbf{k}^{(j)} = K_j \mathbf{h} \in \mathbb{R}^{d_k \times n}$$

$$q_i^{(j)} = Q_j h_i \in \mathbb{R}^{d_k}$$

$$s_i^{(j)} = \frac{q_i^{(j)\intercal} \mathbf{k}^{(j)}}{\sqrt{d_k}} \in \mathbb{R}^n$$

$$\alpha_i^{(j)} = \text{softmax}\left(s_i^{(j)}\right) \in \Delta^{\mathbb{N}_{1:n}}$$

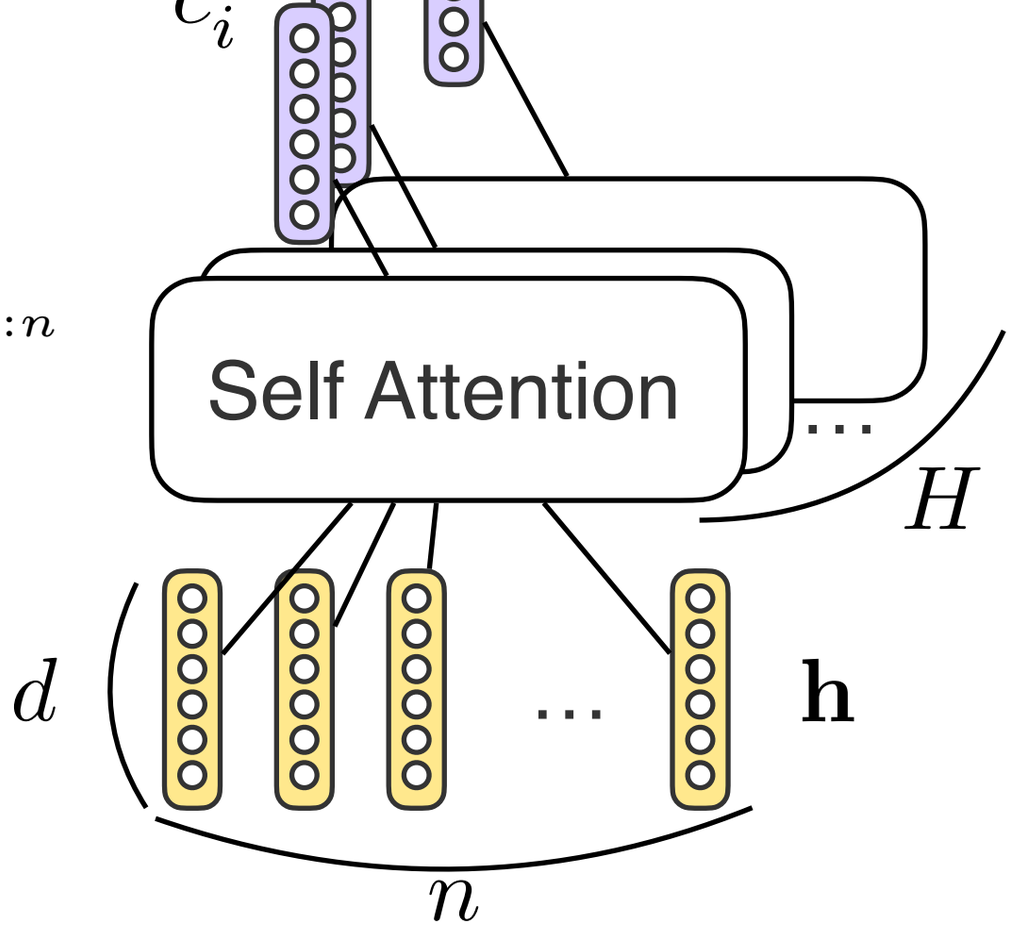
$$\mathbf{v}^{(j)} = V_j \mathbf{h} \in \mathbb{R}^{d_k \times n}$$

$$c_i^{(j)} = \sum_{i'=1} \alpha_i^{(j)} v_{i'}^{(j)}$$

$$c_i^{(j)} = \text{SelfAttention}_j(\mathbf{h})_i$$

Multi-Head Attention

$$\theta_{\text{SelfAttention}_j} = \{K_j, Q_j, V_j\}$$



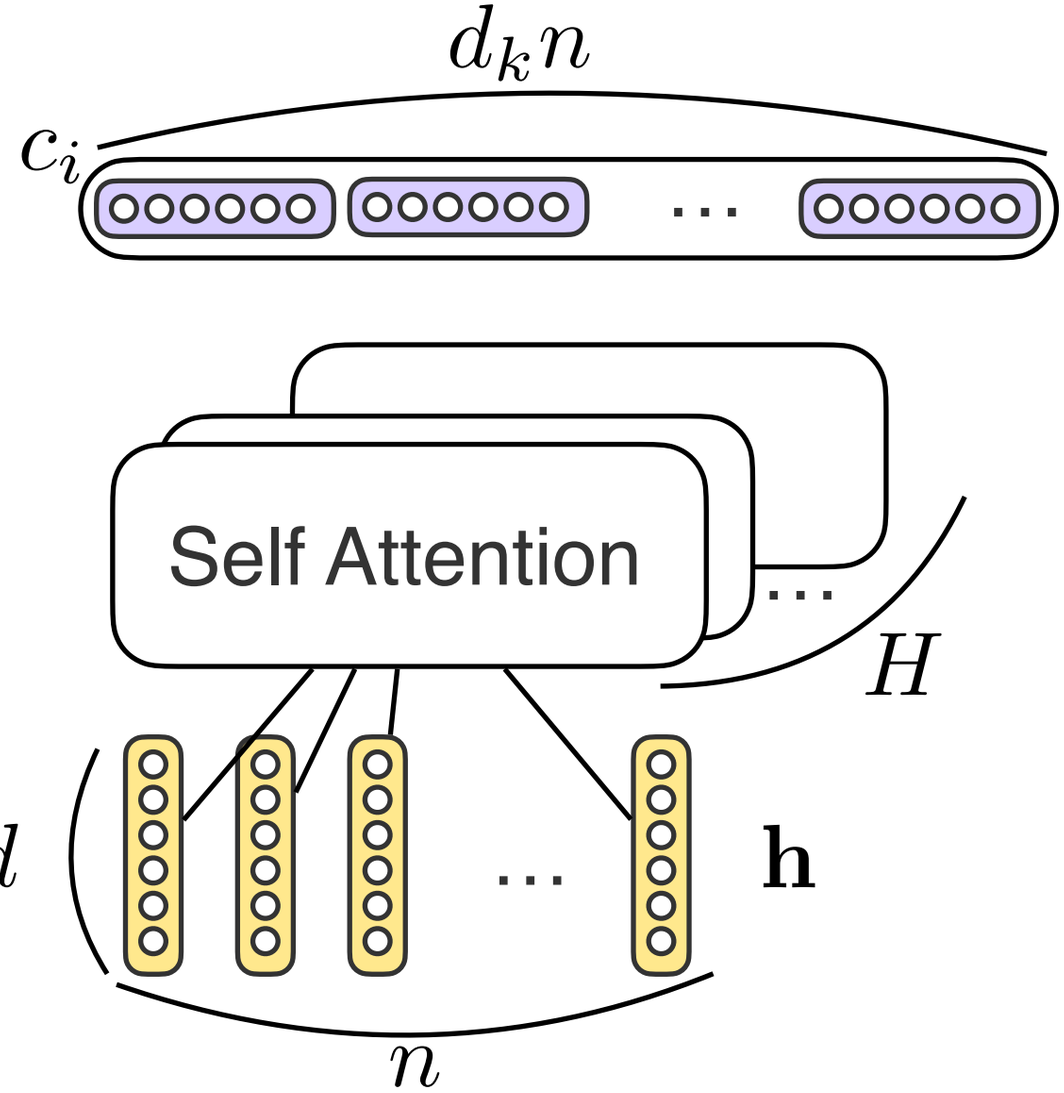
Multi-Head Attention

Multi-Head Attention

1. Compute attention over input vectors

$$c_i^{(j)} = \text{SelfAttention}_j(\mathbf{h})_i$$

$$c_i = \left[c_i^{(1)}; \ldots; c_i^{(H)}\right]$$



Multi-Head Attention

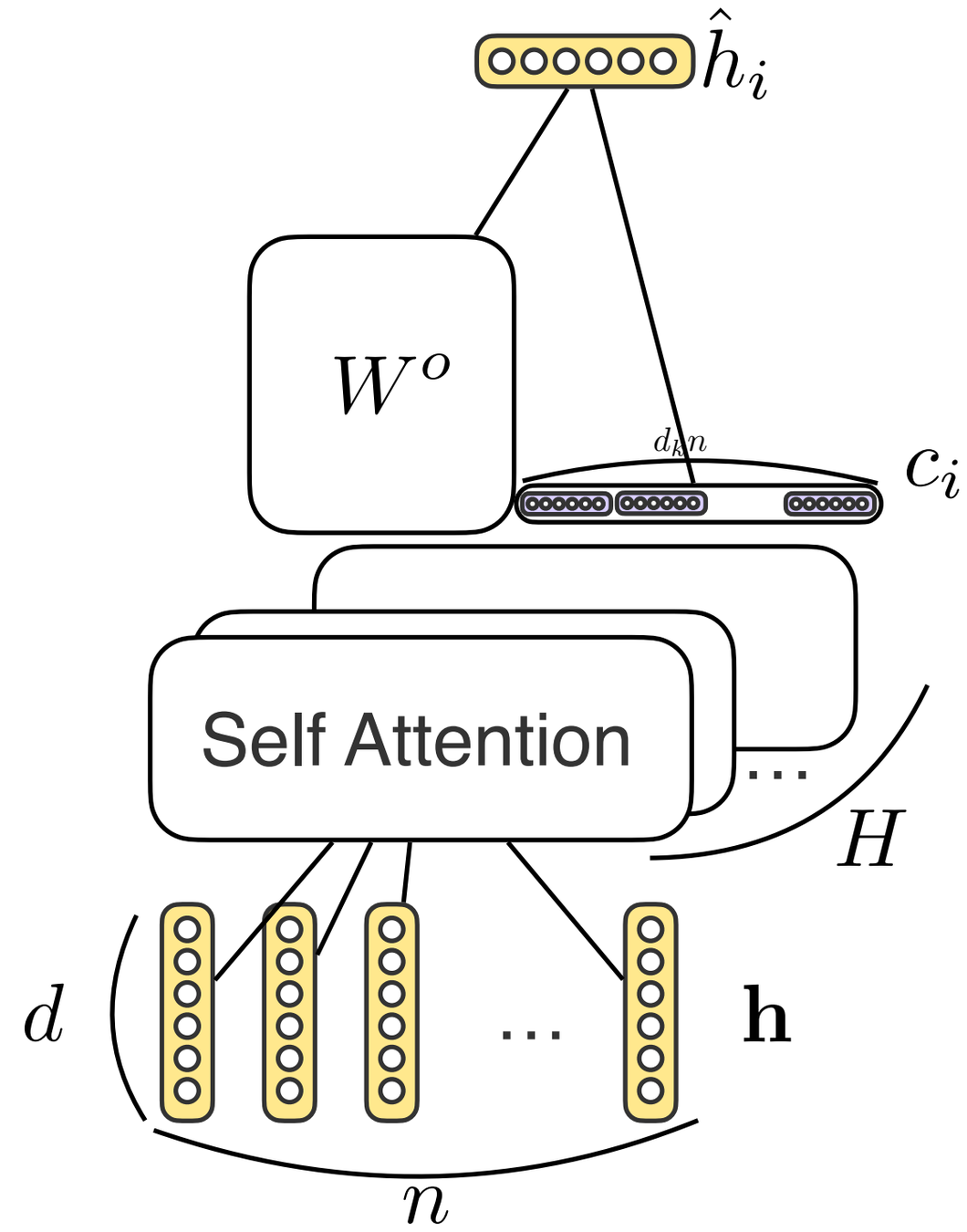
1. Compute attention over input vectors

$$c_i^{(j)} = \text{SelfAttention}_j(\mathbf{h})_i$$

$$c_i = \left[c_i^{(1)}; \ldots; c_i^{(H)}\right]$$

$$\hat{h}_i = W^o c_i$$

Multi-Head Attention



Multi-Head Attention

Multi-Head Attention

1. Compute attention over input vectors

$$c_i^{(j)} = \text{SelfAttention}_j(\mathbf{h})_i$$

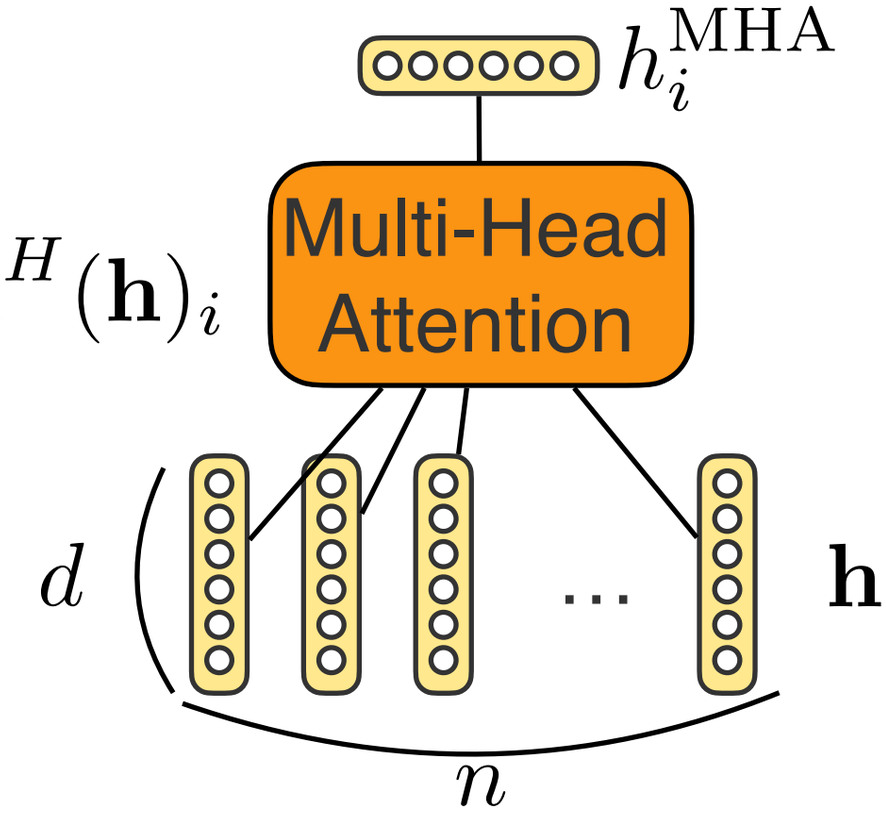
$$c_i = \left[c_i^{(1)}; \ldots; c_i^{(H)}\right]$$

$$\hat{h}_i = W^o c_i$$

$$h_i^{\text{MHA}} = \text{MultiHeadAttention}^H(\mathbf{h})_i$$

$$\theta_{\text{MultiHeadAttention}^H} = \{W^o\} \cup \{K_j, Q_j, V_j\}_{j=1}^H$$

Multi-headed attention allows us to look at multiple places in the input sequence at once.



Layer Normalization

Residual connection allows for smoother gradients.

Layer normalization cuts down on uninformative variation in activations, speeding up training.

1. Compute attention over input vectors

$$h_i^{\text{MHA}} = \text{MultiHeadAttention}^H(\mathbf{h})_i$$

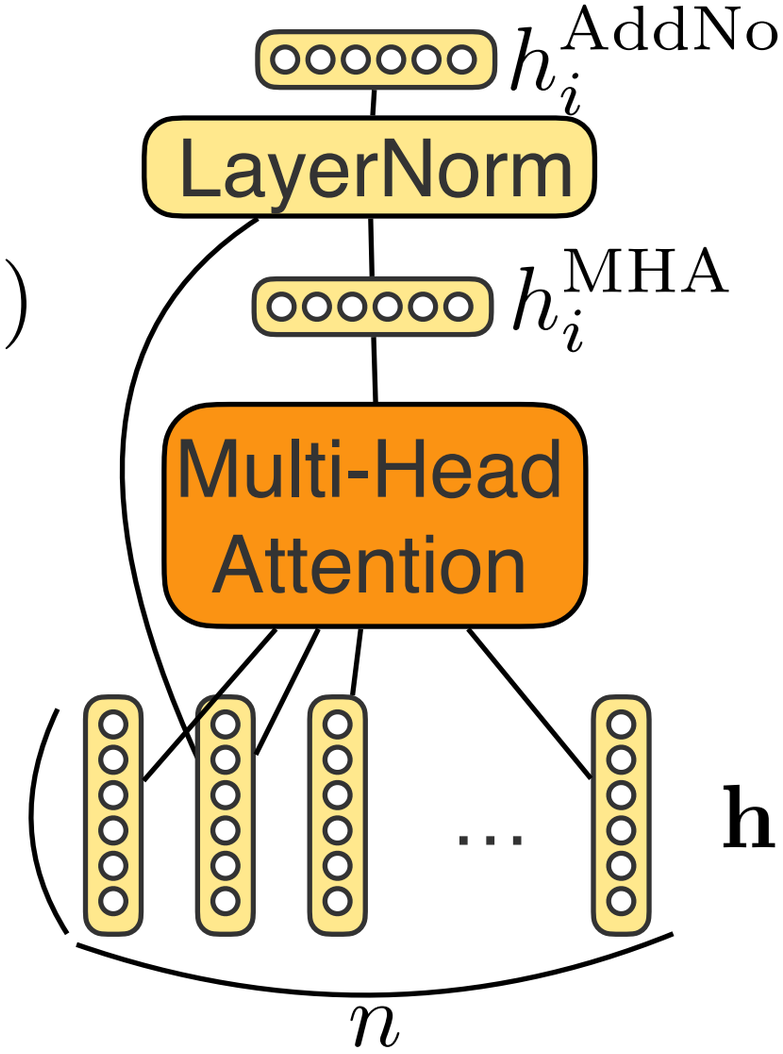
2. Add and norm

$$h_i^{\text{AddNorm}_1} = \text{LayerNorm}(h_i + h_i^{\text{MHA}})$$

$$\text{LayerNorm} : \mathbb{R}^d \to \mathbb{R}^d$$

$$\text{LayerNorm}(x) = \frac{g}{\sigma(x)}(x - \mu(x))$$

$$\mu(x) = \frac{1}{d}\sum_{i=1}^{d} x_i \quad \sigma(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$



Feedforward Network

1. Compute attention over input vectors

$$h_i^{\text{MHA}} = \text{MultiHeadAttention}^H(\mathbf{h})_i$$

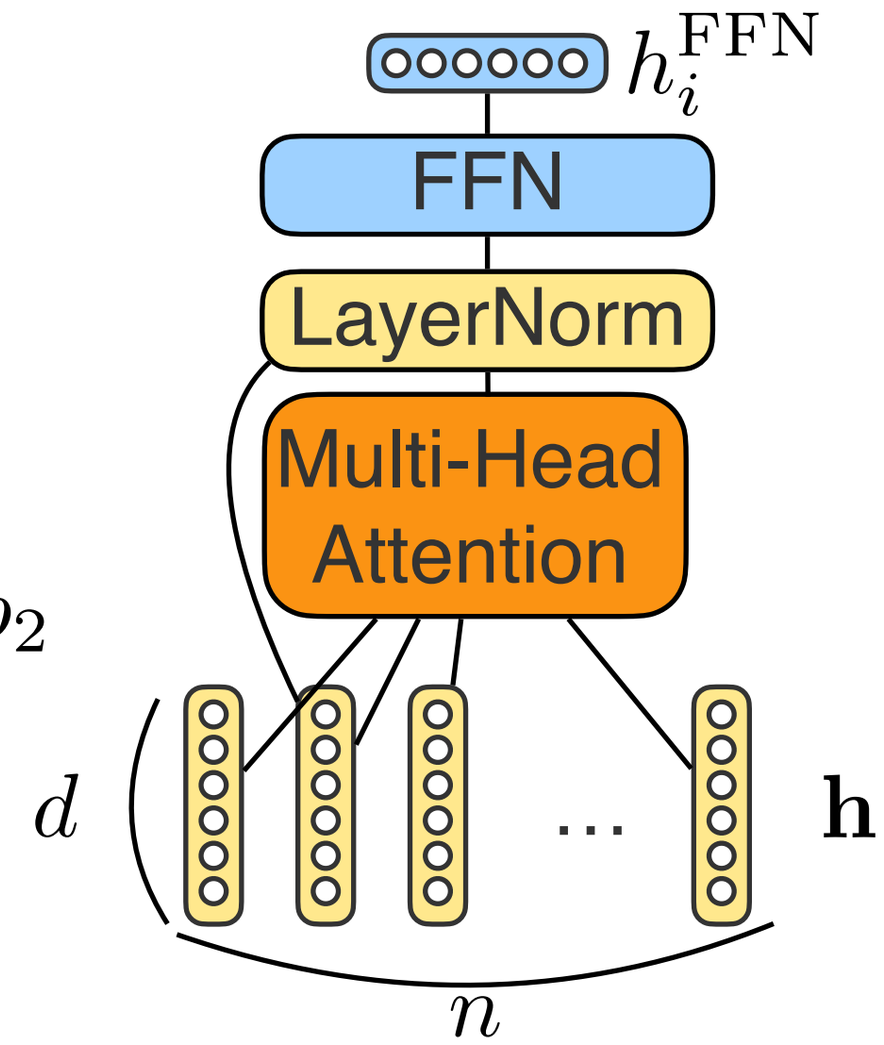
2. Add and norm

$$h_i^{\text{AddNorm}_1} = \text{LayerNorm}(h_i + h_i^{\text{MHA}})$$

3. Feedforward layer

$$h_i^{\text{FFN}} = \text{ReLU}(h_i^{\text{AddNorm}_1} W_1 + b_1)W_2 + b_2$$

Here is a nonlinearity that makes learning easier!



More Layer Normalization

1. Compute attention over input vectors

$$h_i^{\text{MHA}} = \text{MultiHeadAttention}^H(\mathbf{h})_i$$

2. Add and norm

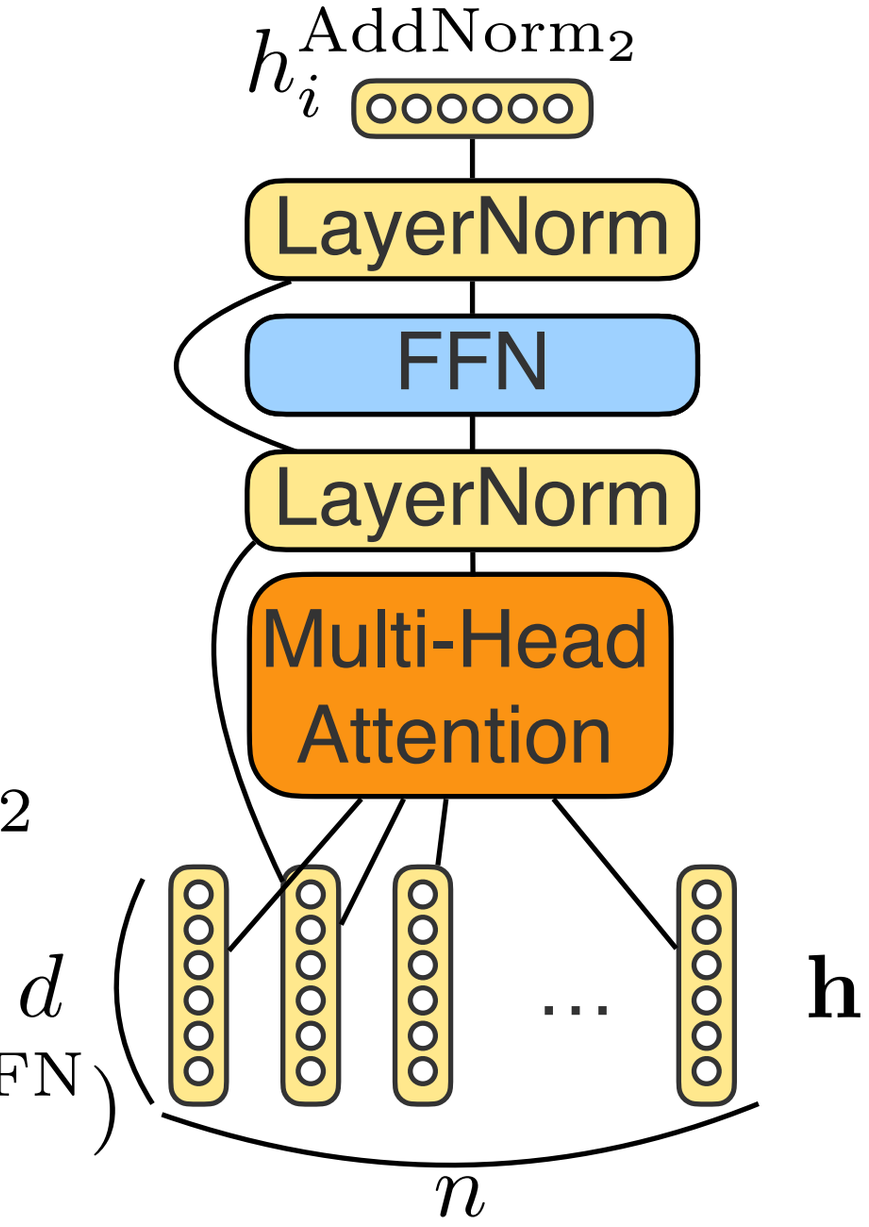
$$h_i^{\text{AddNorm}_1} = \text{LayerNorm}(h_i + h_i^{\text{MHA}})$$

3. Feedforward layer

$$h_i^{\text{FFN}} = \text{ReLU}(h_i^{\text{AddNorm}_1} W_1 + b_1)W_2 + b_2$$

4. Another add and norm

$$h_i^{\text{AddNorm}_2} = \text{LayerNorm}(h_i^{\text{AddNorm}_1} + h_i^{\text{FFN}})$$



One Transformer Block

1. Compute attention over input vectors

$$h_i^{\text{MHA}} = \text{MultiHeadAttention}^H(\mathbf{h})_i$$

2. Add and norm

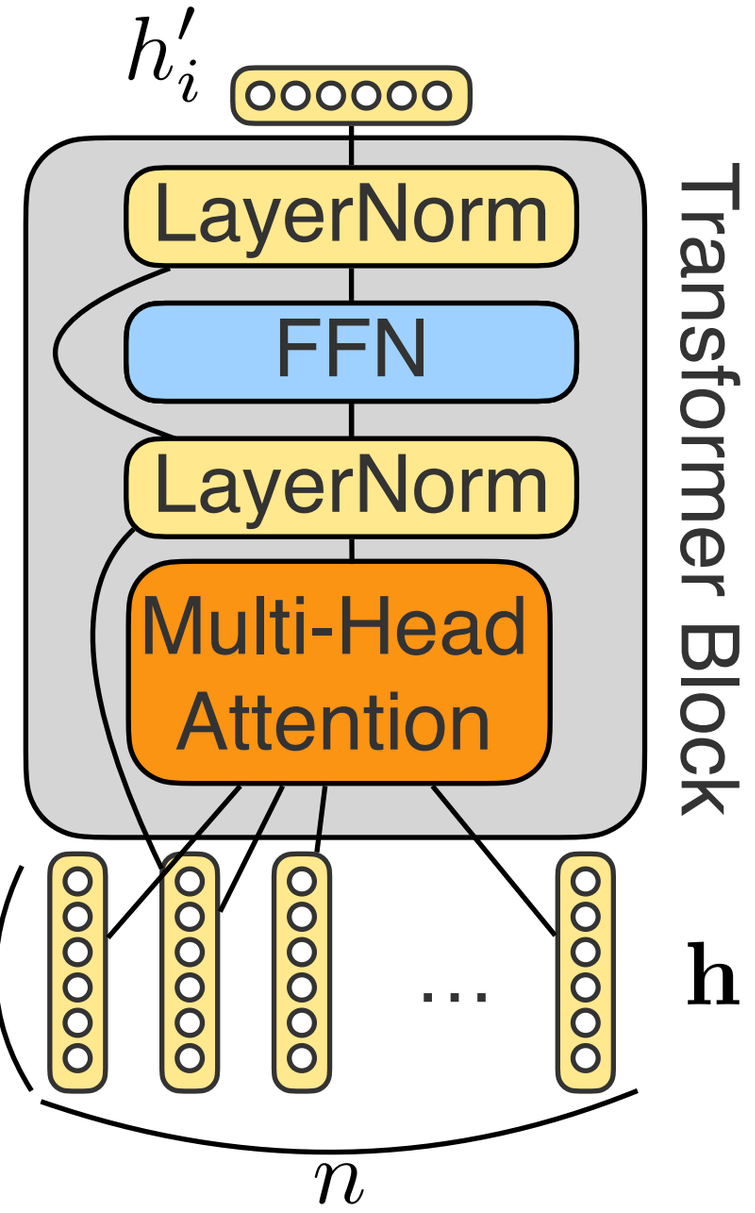
$$h_i^{\text{AddNorm}_1} = \text{LayerNorm}(h_i + h_i^{\text{MHA}})$$

3. Feedforward layer

$$h_i^{\text{FFN}} = \text{ReLU}(h_i^{\text{AddNorm}_1} W_1 + b_1) W_2 + b_2$$

4. Another add and norm

$$h_i' = \text{LayerNorm}(h_i^{\text{AddNorm}_1} + h_i^{\text{FFN}})$$



One Transformer Block

$$\mathbf{h}^{\text{MHA}} = \text{MultiHeadAttention}^{H}(\mathbf{h})$$

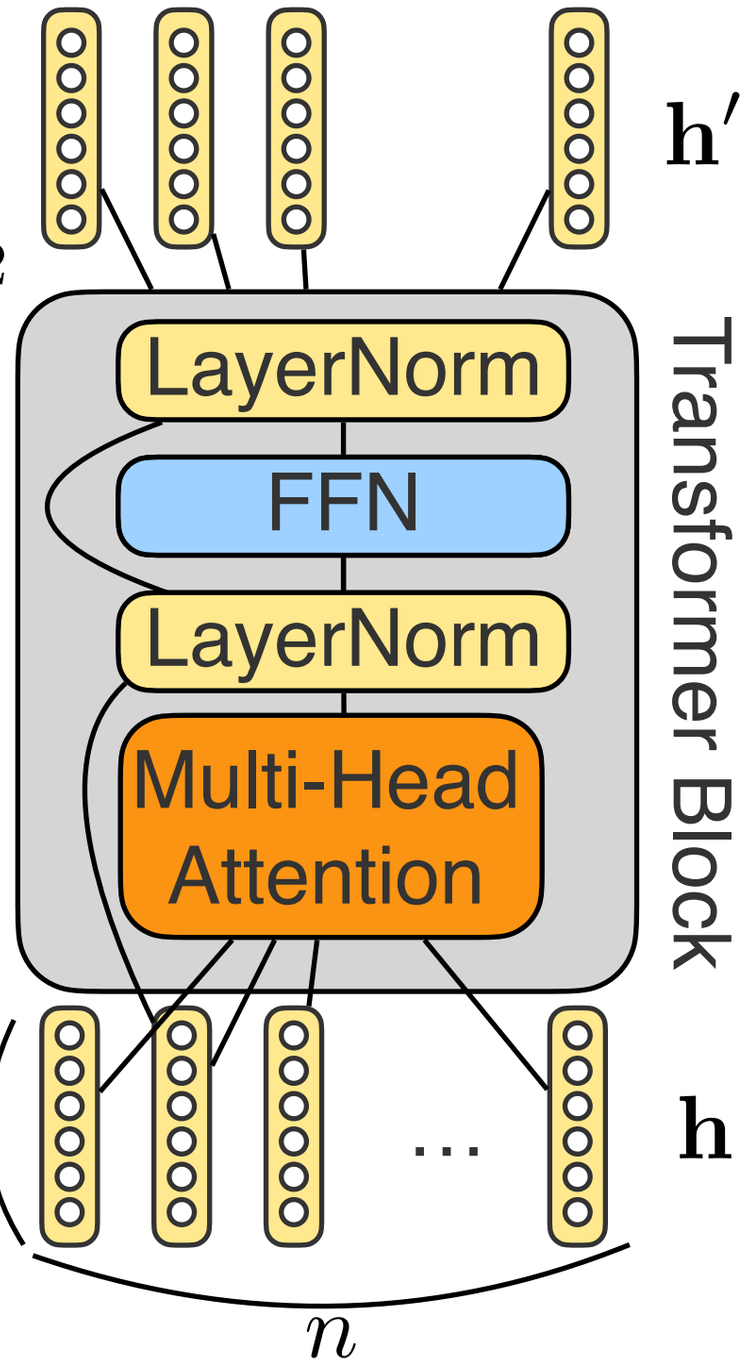
$$\mathbf{h}^{\text{AddNorm}} = \text{LayerNorm}(\mathbf{h} + \mathbf{h}^{\text{MHA}})$$

$$\mathbf{h}_i^{\text{FFN}} = \text{ReLU}(\mathbf{h}^{\text{AddNorm}} W_1 + b_1) W_2 + b_2$$

$$\mathbf{h}' = \text{LayerNorm}(\mathbf{h}^{\text{AddNorm}} + \mathbf{h}^{\text{FFN}})$$

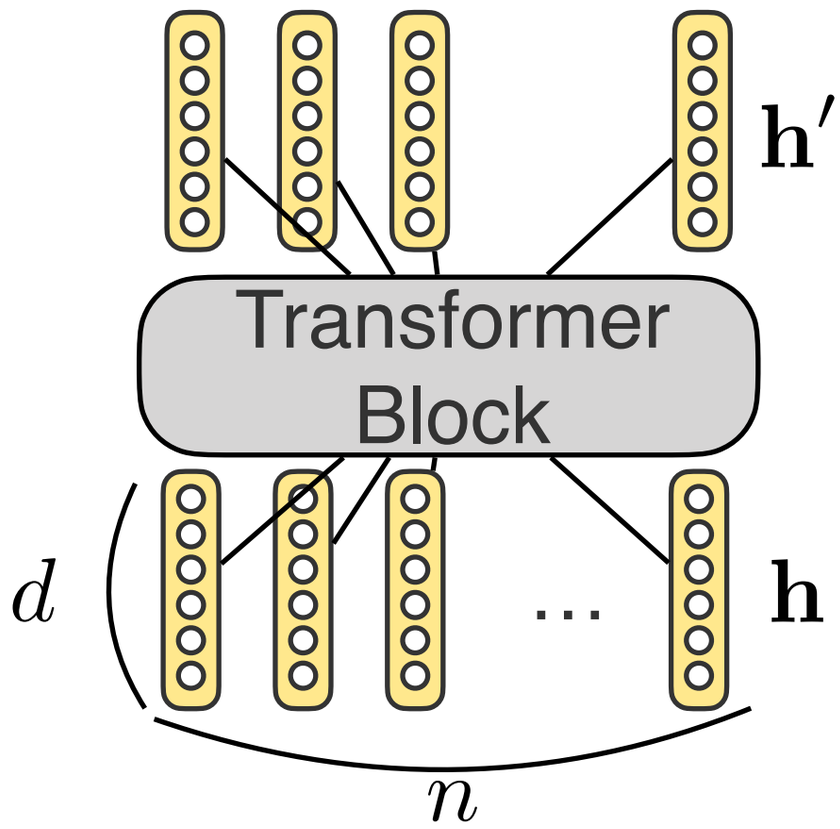
$$\mathbf{h}' = \text{TransformerBlock}(\mathbf{h})$$

$$\theta_{\text{TransformerBlock}} = \theta_{\text{MultiHeadAttention}^{H}} \cup \{W_1, W_2, b_1, b_2\}$$



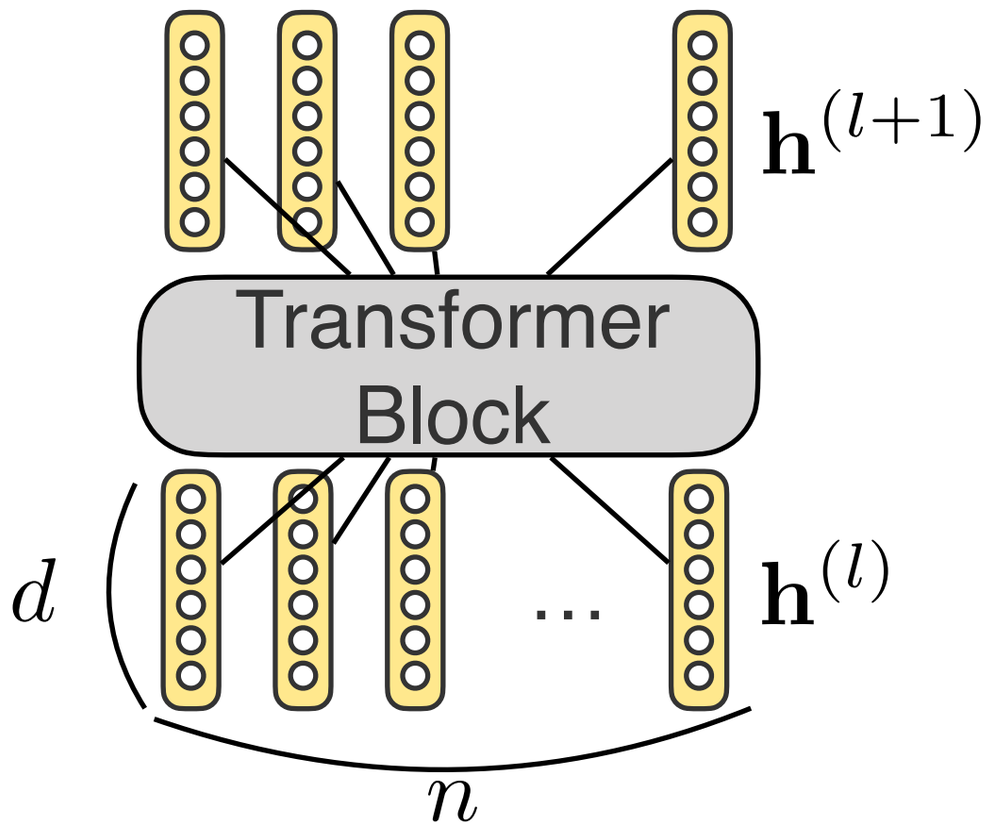
Multi-Layer Transformer

$$\mathbf{h}' = \text{TransformerBlock}(\mathbf{h})$$



Multi-Layer Transformer

$$\mathbf{h}^{(\mathrm{l}+1)} = \mathrm{TransformerBlock}_{l+1}(\mathbf{h}^{(l)})$$



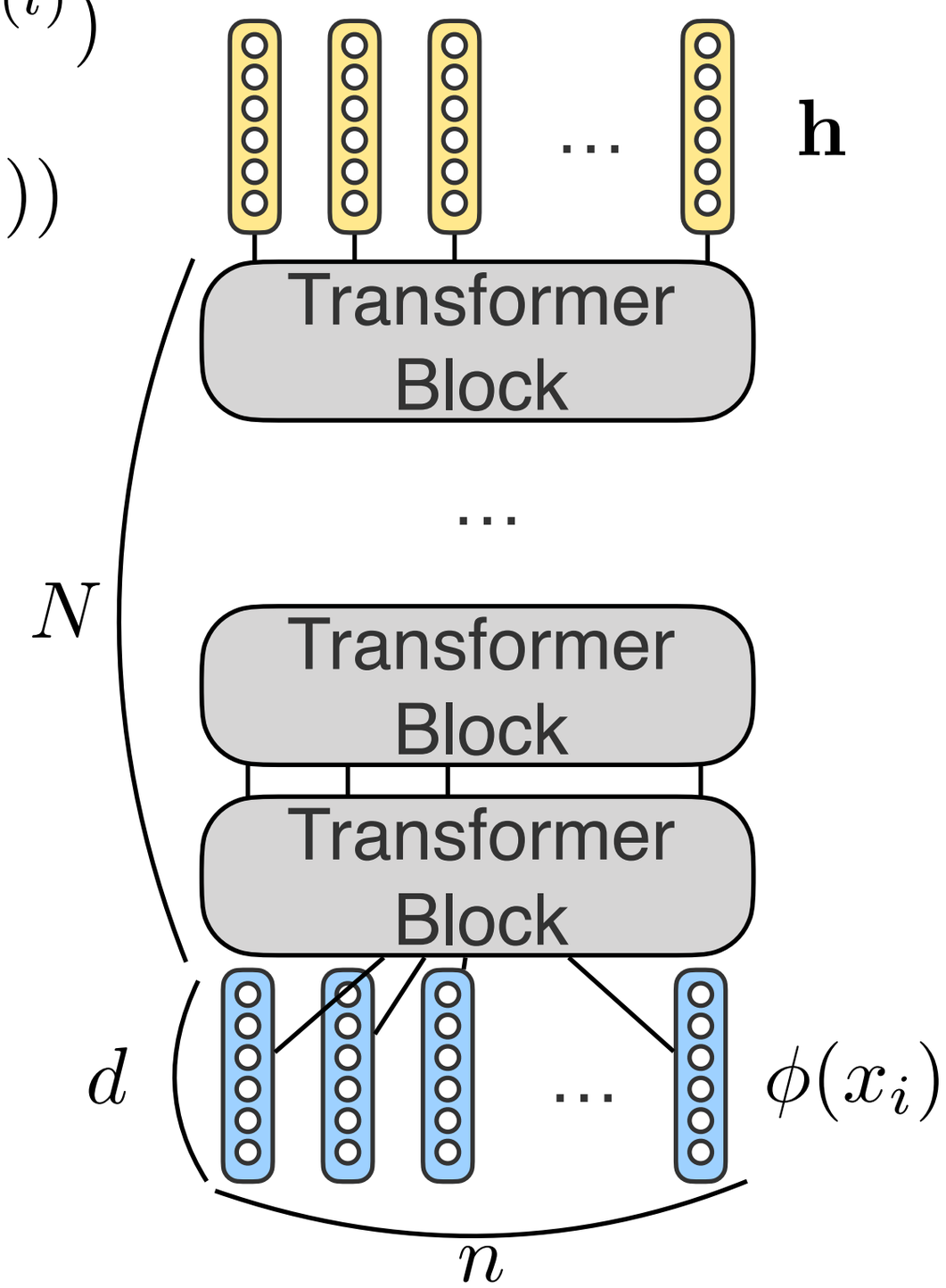
Transformer Encoder

$$\mathbf{h}^{(l+1)} = \text{TransformerBlock}_{l+1}(\mathbf{h}^{(l)})$$

$$\mathbf{h}^{N} = \underset{\text{Encoder}}{\text{Transformer}}(\phi(x_1), \ldots, \phi(x_n))$$

$$\mathbf{h}^{N} \in \mathbb{R}^{d \times n}$$

Input: $\phi(x_i) = \phi(x) + \phi(i)$
word + position embeddings
$\overline{x} = \langle x_1, \ldots, x_n \rangle$



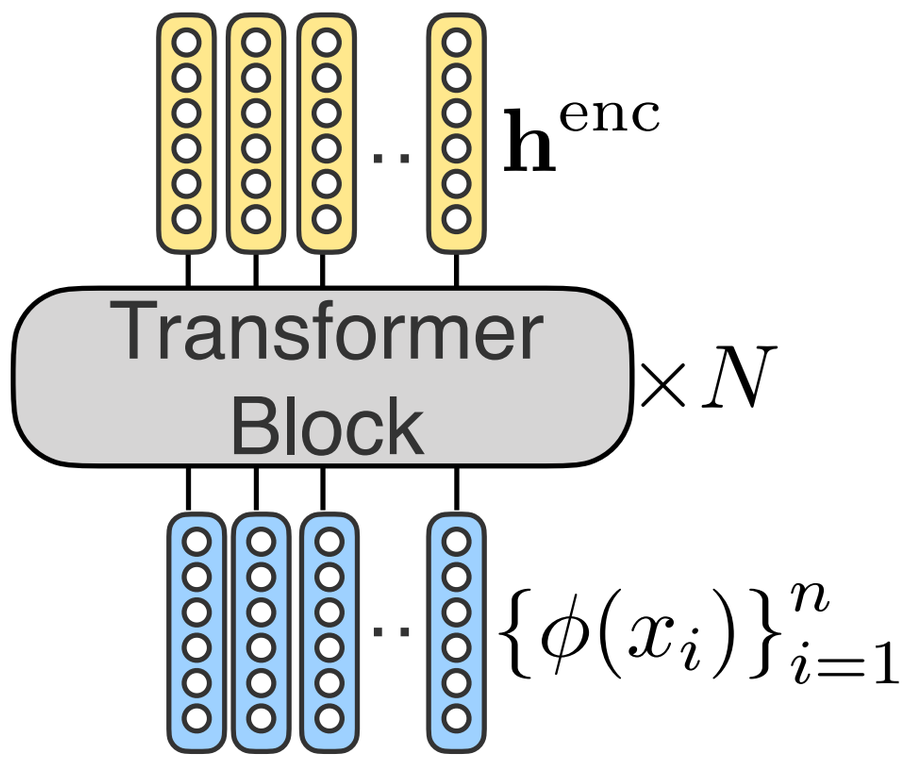
Transformer Encoder

$$\mathbf{h}^N = \underset{\text{Encoder}}{\text{Transformer}}(\phi(x_1), \ldots, \phi(x_n))$$

$$\underset{\text{Encoder}}{\text{Transformer}} : \mathcal{V}^+ \rightarrow \mathbb{R}^{d \times n}$$

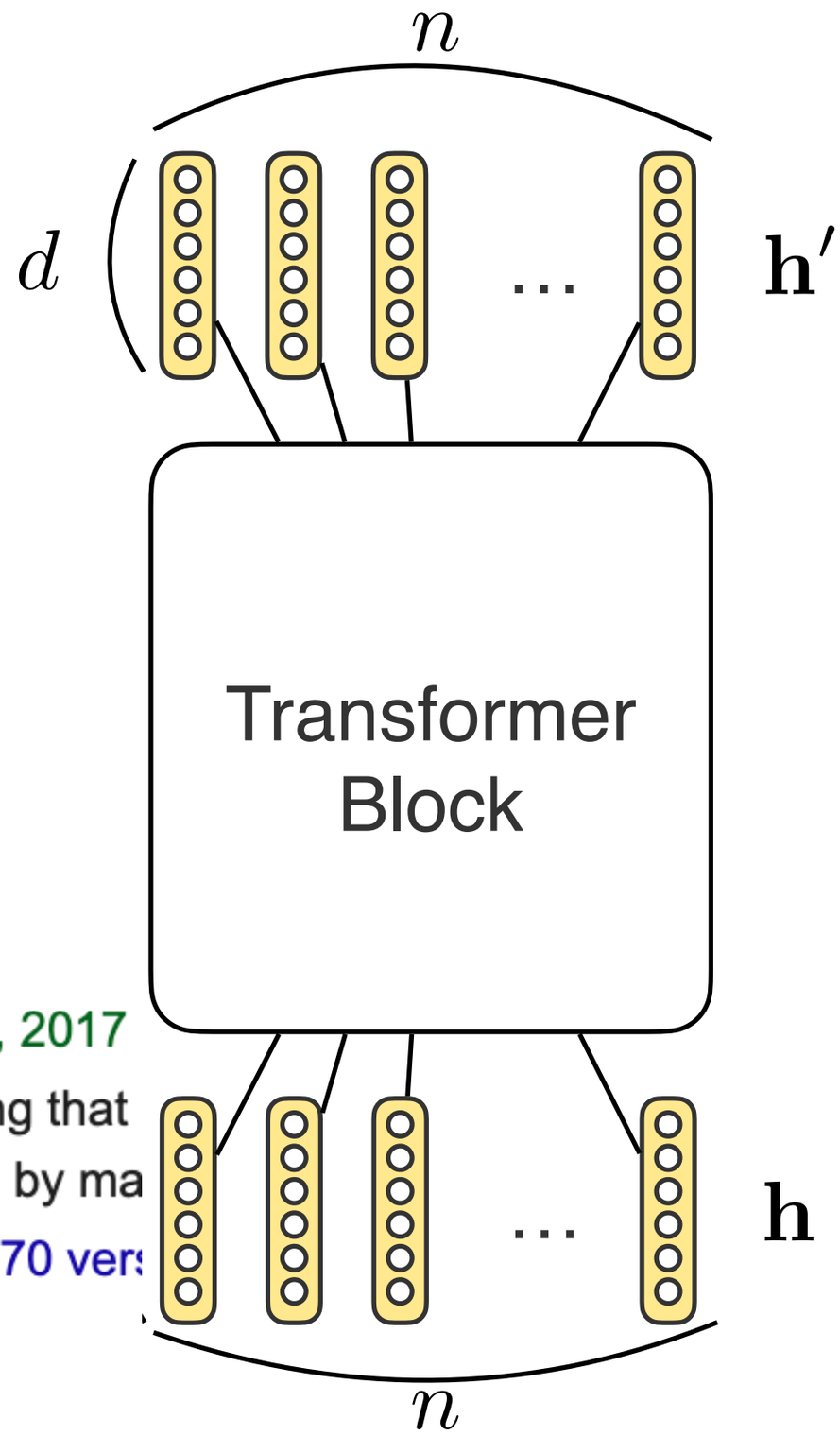
The transformer encoder maps from a sequence of tokens to a set of vectors, each corresponding to a token in the input sequence

This is exactly what we got from an RNN encoder and used for conditional generation in a decoder!



Building a Transformer Block

- A transformer block takes as input a sequence of input vectors $\mathbf{h} \in \mathbb{R}^{d \times n}$
- It generates a sequence of output vectors $\mathbf{h}' \in \mathbb{R}^{d \times n}$



Attention is all you need

A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017

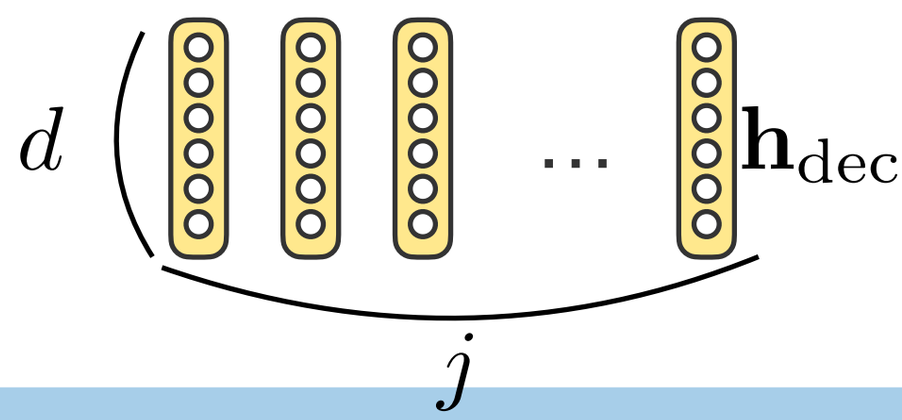
… to attend to **all** positions in the decoder up to and including that

… **We** implement this inside of scaled dot-product **attention** by ma

☆ Save 99 Cite Cited by 196986 Related articles All 70 vers

Decoder Block

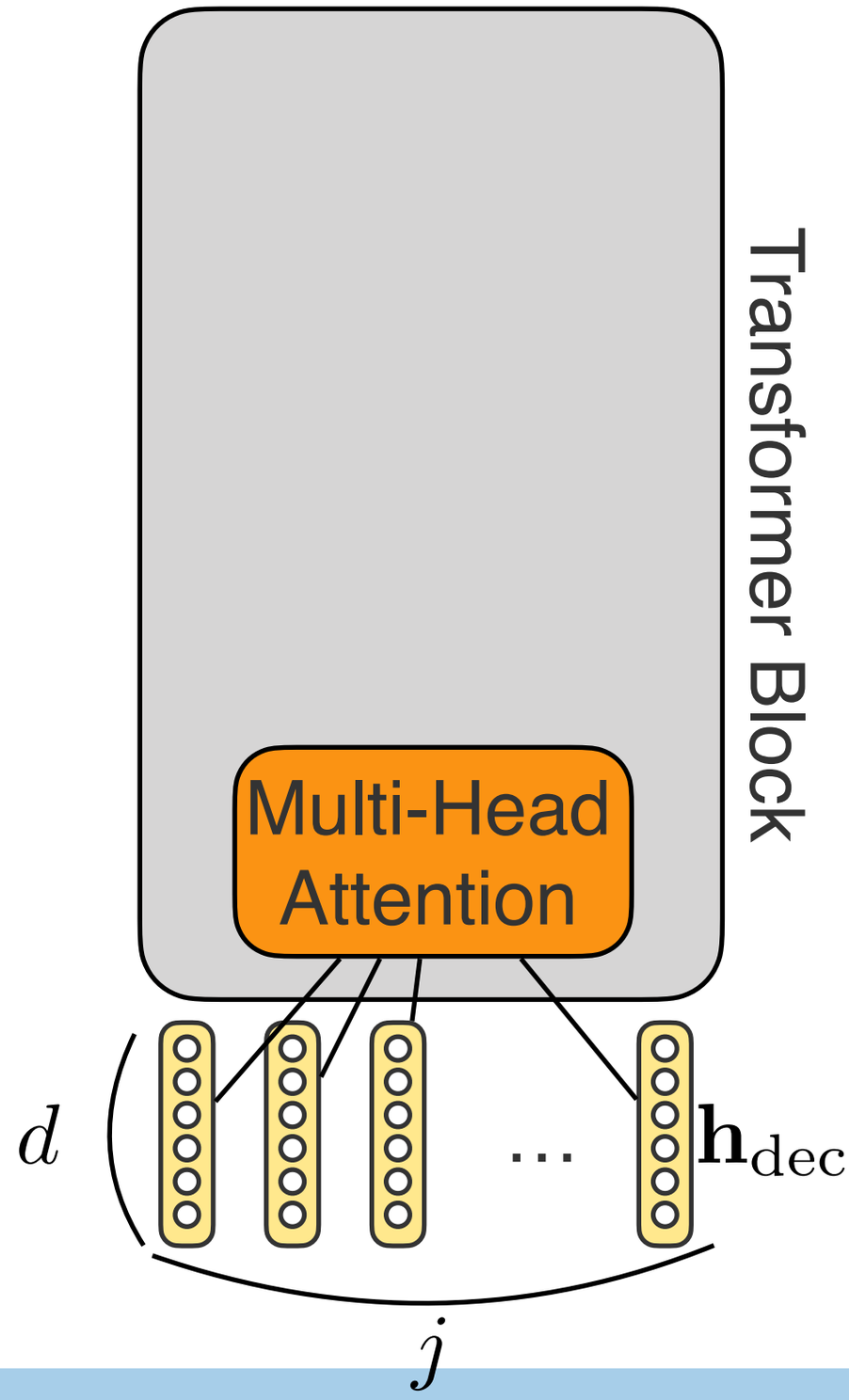
We’re at generation step *j*, and have our previous decoder steps available $\mathbf{h}_{\text{dec}}$



Decoder Block

$$\mathbf{h}^{\mathrm{MHA}_1} = \mathrm{MultiHeadAttention}^{H}(\mathbf{h}_{\mathrm{dec}})$$

First, we do self-attention over the tokens we've already generated

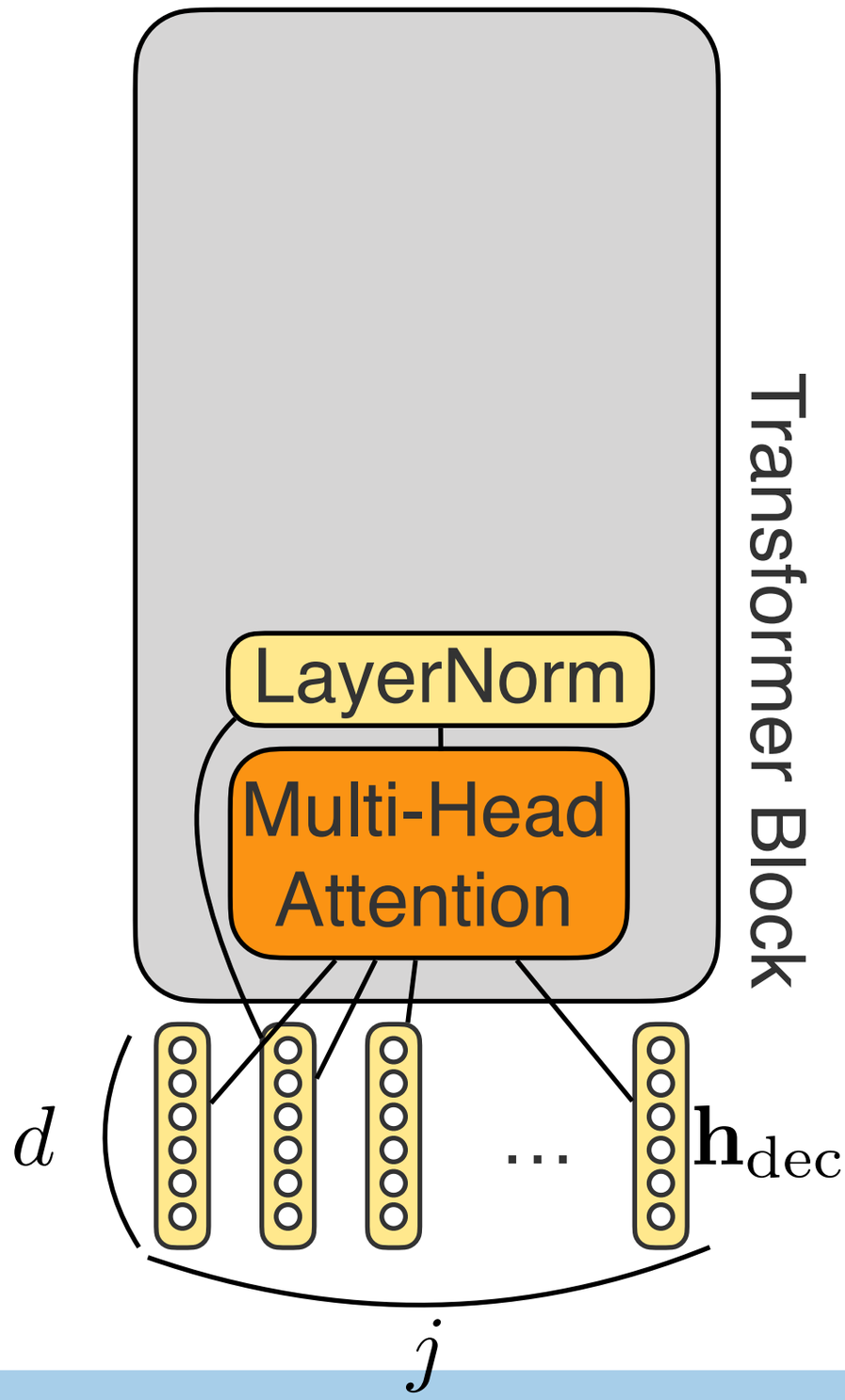


Decoder Block

$$\mathbf{h}^{\mathrm{MHA}_1} = \mathrm{MultiHeadAttention}^{H}(\mathbf{h}_{\mathrm{dec}})$$

$$\mathbf{h}^{\mathrm{AddNorm}_1} = \mathrm{LayerNorm}(\mathbf{h}^{\mathrm{dec}} + \mathbf{h}^{\mathrm{MHA}_1})$$

Layer norm on the output of the self-attention



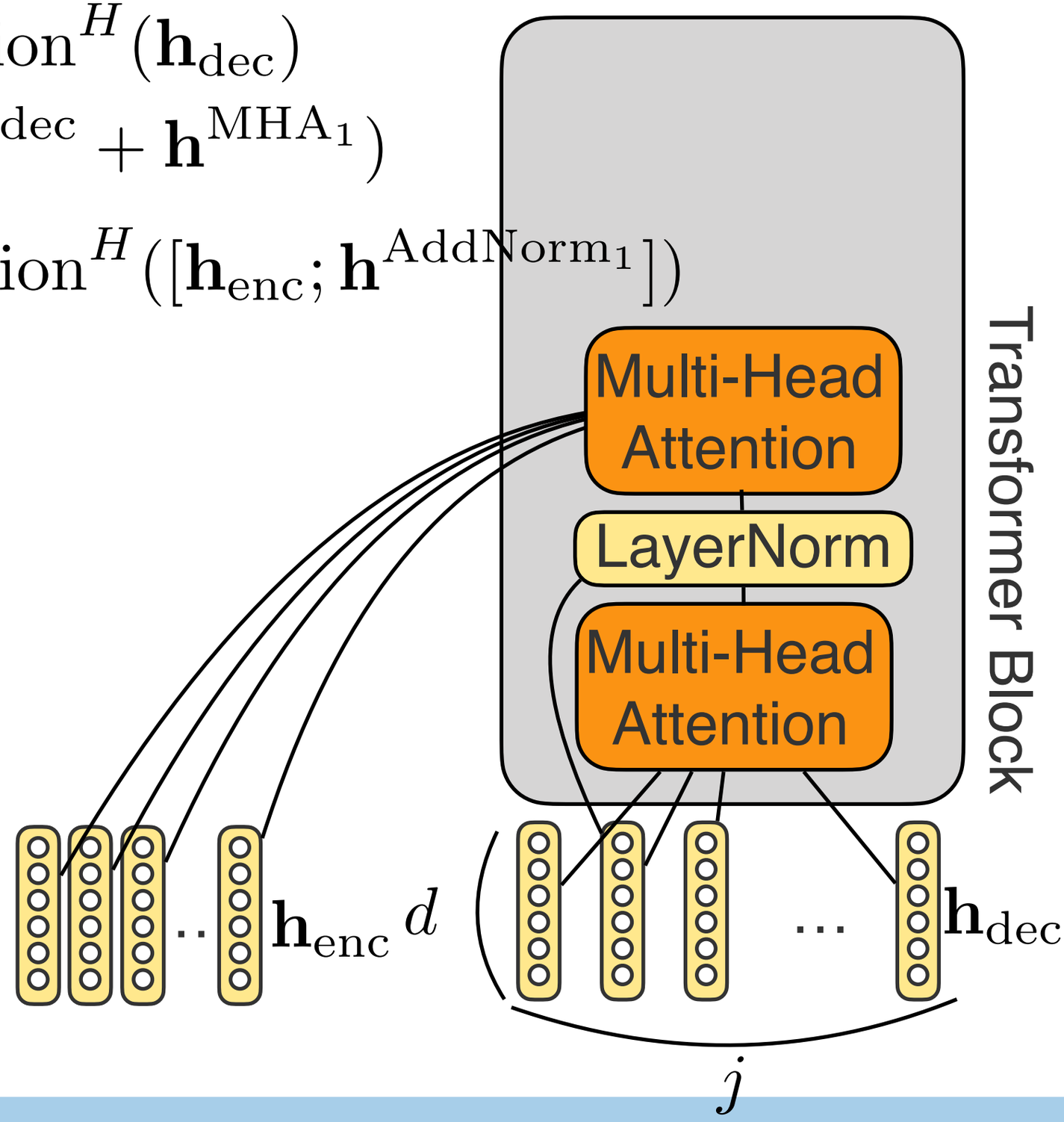
Decoder Block

$$\mathbf{h}^{\mathrm{MHA}_1} = \mathrm{MultiHeadAttention}^H(\mathbf{h}_{\mathrm{dec}})$$

$$\mathbf{h}^{\mathrm{AddNorm}_1} = \mathrm{LayerNorm}(\mathbf{h}^{\mathrm{dec}} + \mathbf{h}^{\mathrm{MHA}_1})$$

$$\mathbf{h}^{\mathrm{MHA}_2} = \mathrm{MultiHeadAttention}^H([\mathbf{h}_{\mathrm{enc}}; \mathbf{h}^{\mathrm{AddNorm}_1}])$$

Then we attend to the encoder hidden states using the output of the previous layer normalization



Decoder Block

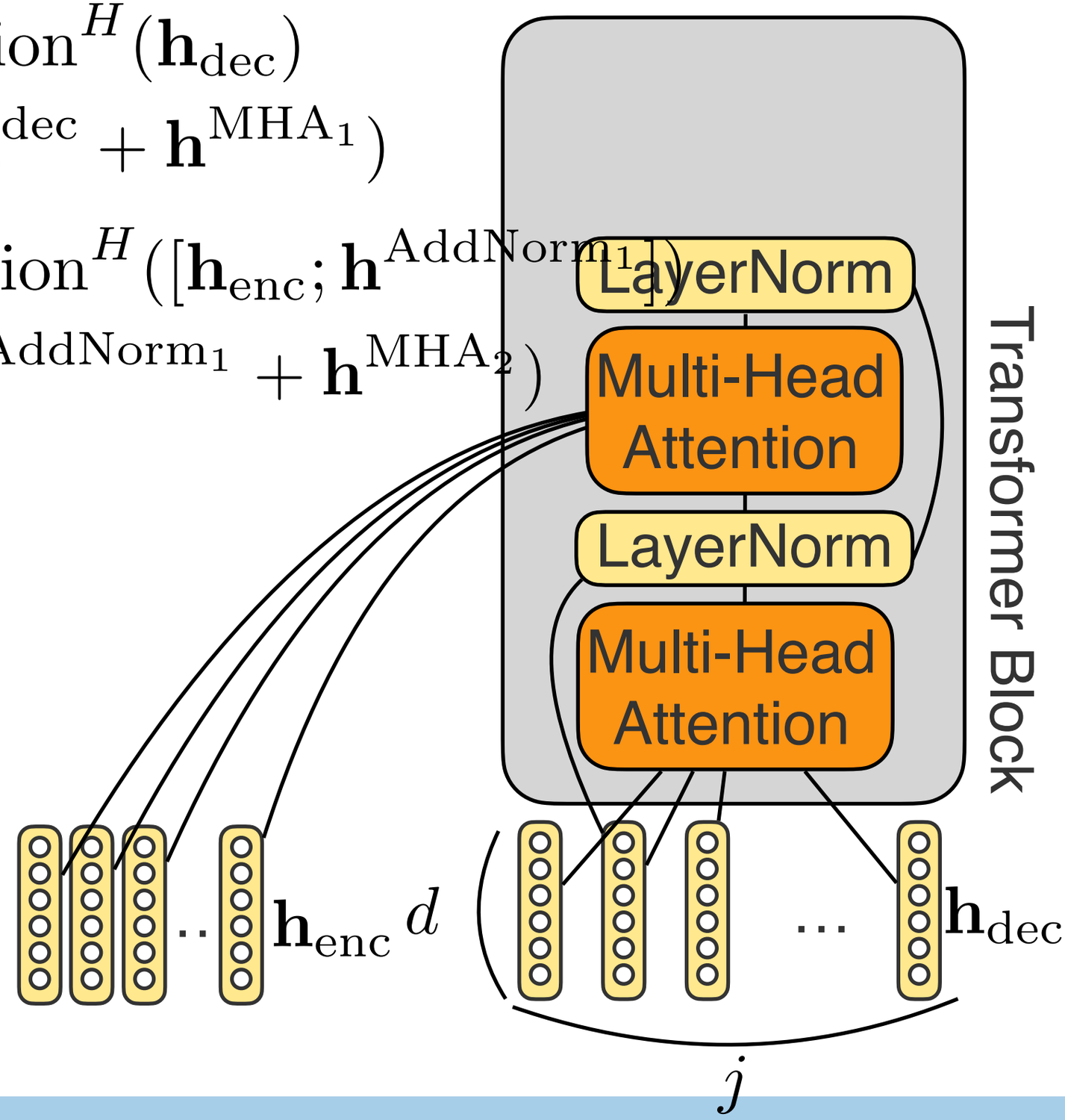
$$\mathbf{h}^{\mathrm{MHA}_1} = \mathrm{MultiHeadAttention}^H(\mathbf{h}_{\mathrm{dec}})$$

$$\mathbf{h}^{\mathrm{AddNorm}_1} = \mathrm{LayerNorm}(\mathbf{h}^{\mathrm{dec}} + \mathbf{h}^{\mathrm{MHA}_1})$$

$$\mathbf{h}^{\mathrm{MHA}_2} = \mathrm{MultiHeadAttention}^H([\mathbf{h}_{\mathrm{enc}}; \mathbf{h}^{\mathrm{AddNorm}_1}])$$

$$\mathbf{h}^{\mathrm{AddNorm}_2} = \mathrm{LayerNorm}(\mathbf{h}^{\mathrm{AddNorm}_1} + \mathbf{h}^{\mathrm{MHA}_2})$$

Yet another layer norm



Decoder Block

$$\mathbf{h}^{\mathrm{MHA}_1} = \mathrm{MultiHeadAttention}^H(\mathbf{h}_{\mathrm{dec}})$$

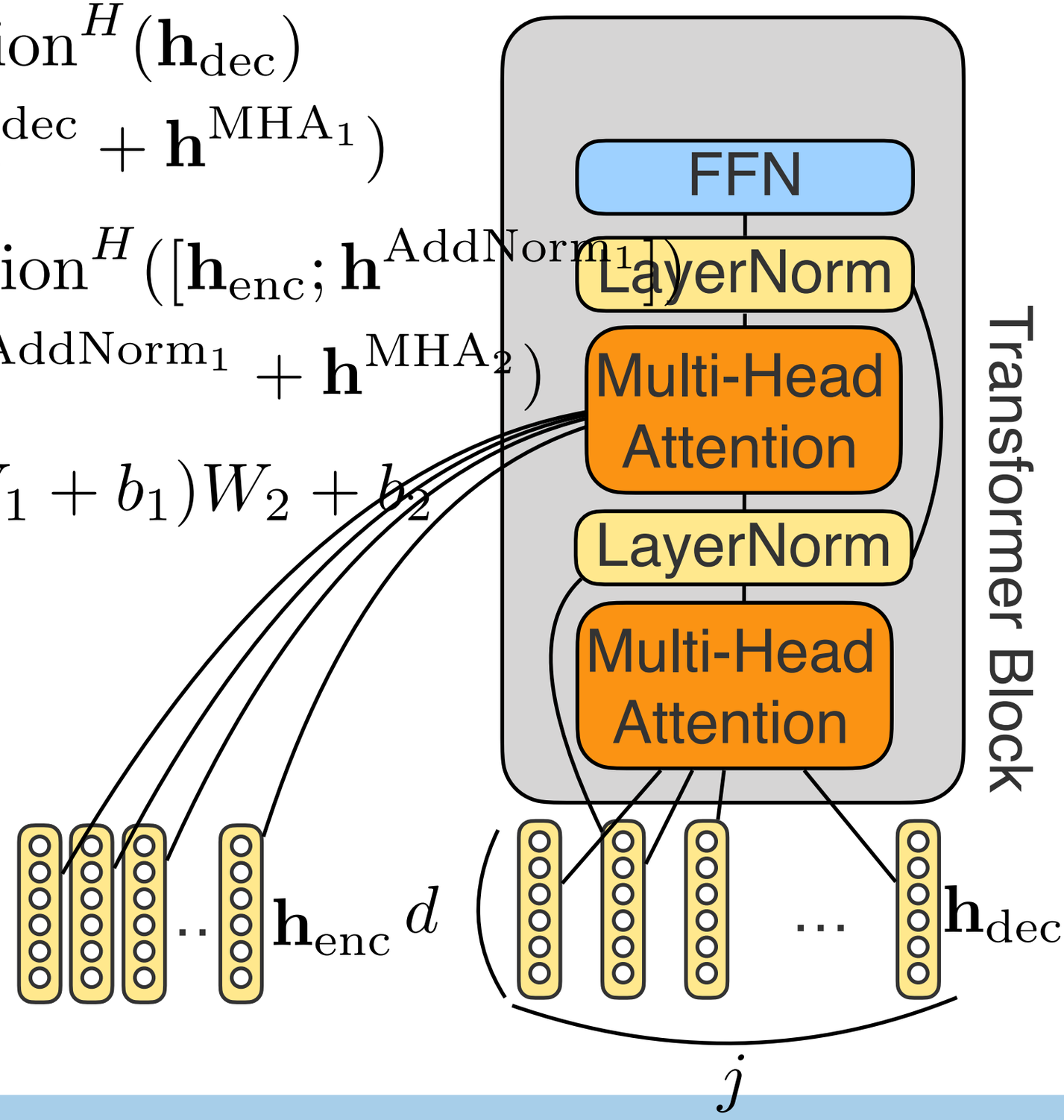
$$\mathbf{h}^{\mathrm{AddNorm}_1} = \mathrm{LayerNorm}(\mathbf{h}^{\mathrm{dec}} + \mathbf{h}^{\mathrm{MHA}_1})$$

$$\mathbf{h}^{\mathrm{MHA}_2} = \mathrm{MultiHeadAttention}^H([\mathbf{h}_{\mathrm{enc}}; \mathbf{h}^{\mathrm{AddNorm}_1}])$$

$$\mathbf{h}^{\mathrm{AddNorm}_2} = \mathrm{LayerNorm}(\mathbf{h}^{\mathrm{AddNorm}_1} + \mathbf{h}^{\mathrm{MHA}_2})$$

$$\mathbf{h}^{\mathrm{FFN}} = \mathrm{ReLU}(\mathbf{h}^{\mathrm{AddNorm}_2} W_1 + b_1) W_2 + b_2$$

Feedforward network



Decoder Block

$$\mathbf{h}^{\text{MHA}_1} = \text{MultiHeadAttention}^H(\mathbf{h}_{\text{dec}})$$

$$\mathbf{h}^{\text{AddNorm}_1} = \text{LayerNorm}(\mathbf{h}^{\text{dec}} + \mathbf{h}^{\text{MHA}_1})$$

$$\mathbf{h}^{\text{MHA}_2} = \text{MultiHeadAttention}^H([\mathbf{h}_{\text{enc}}; \mathbf{h}^{\text{AddNorm}_1}])$$

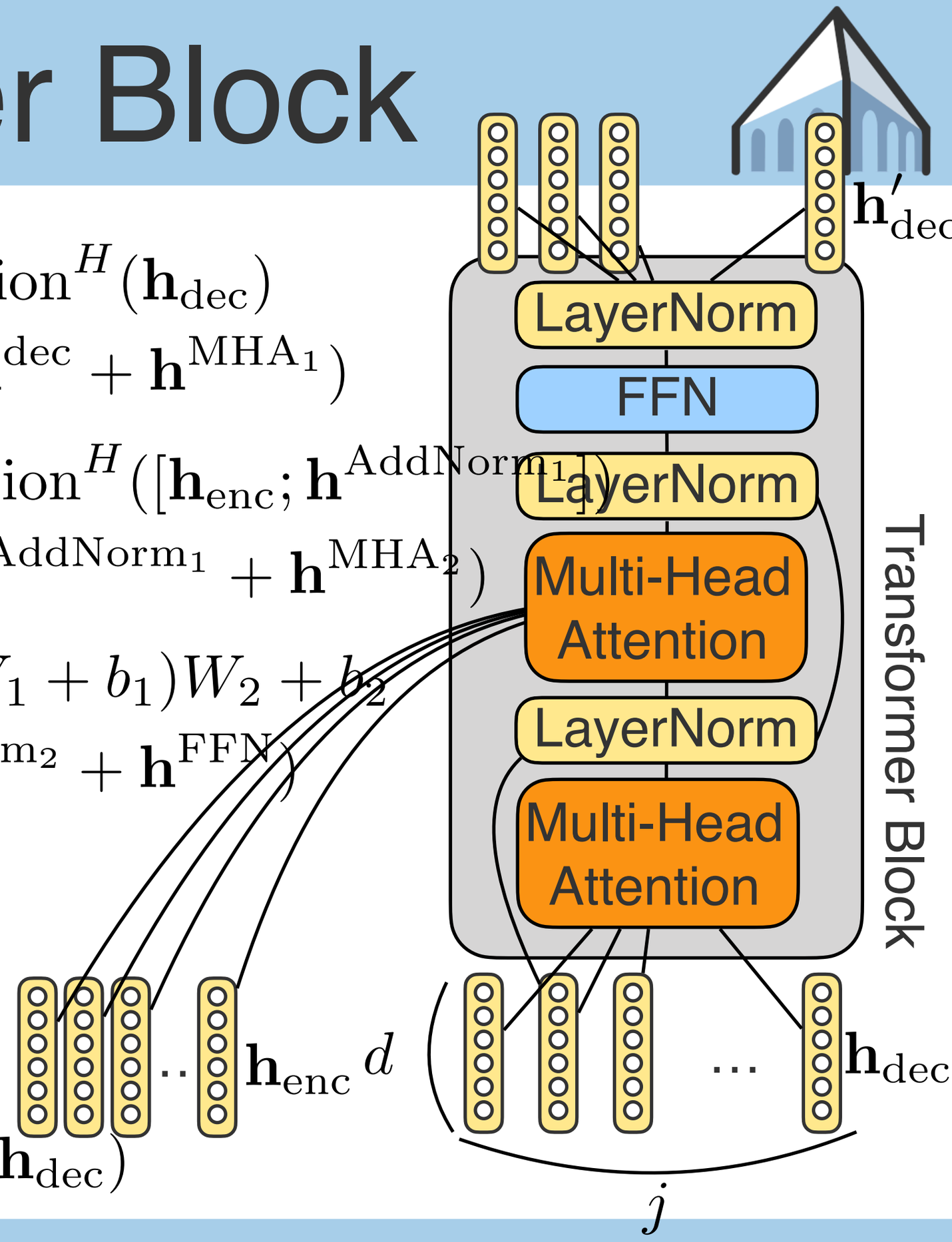
$$\mathbf{h}^{\text{AddNorm}_2} = \text{LayerNorm}(\mathbf{h}^{\text{AddNorm}_1} + \mathbf{h}^{\text{MHA}_2})$$

$$\mathbf{h}^{\text{FFN}} = \text{ReLU}(\mathbf{h}^{\text{AddNorm}_2} W_1 + b_1) W_2 + b_2$$

$$\mathbf{h}'_{\text{dec}} = \text{LayerNorm}(\mathbf{h}^{\text{AddNorm}_2} + \mathbf{h}^{\text{FFN}})$$

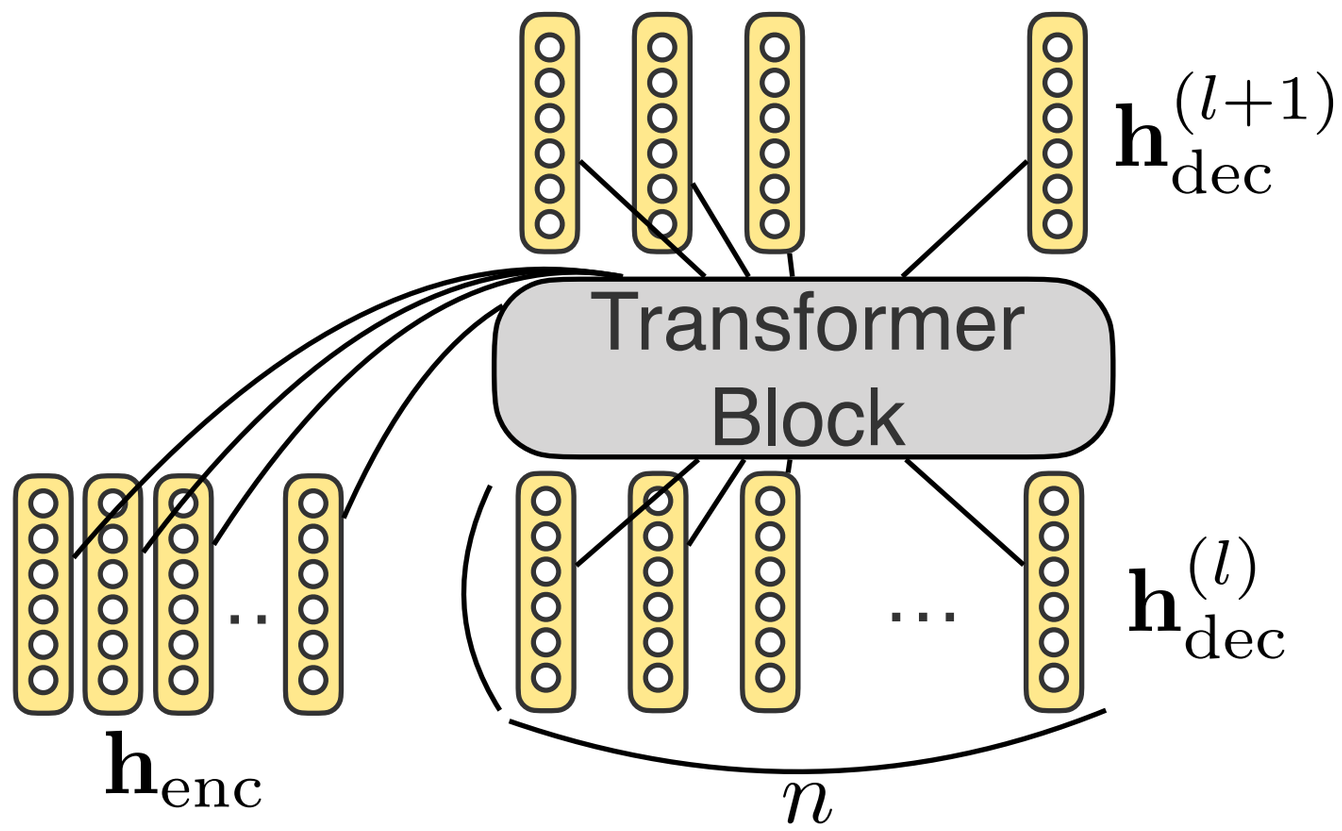
One last layer norm to get our final output vectors for each token generated so far!

$$\mathbf{h}'_{\text{dec}} = \text{DecoderBlock}(\mathbf{h}_{\text{enc}}, \mathbf{h}_{\text{dec}})$$



Multi-Layer Decoder

$$\mathbf{h}_{\text{dec}}^{(l+1)} = \text{DecoderBlock}_{l+1}(\mathbf{h}_{\text{enc}}, \mathbf{h}_{\text{dec}}^{(l+1)})$$



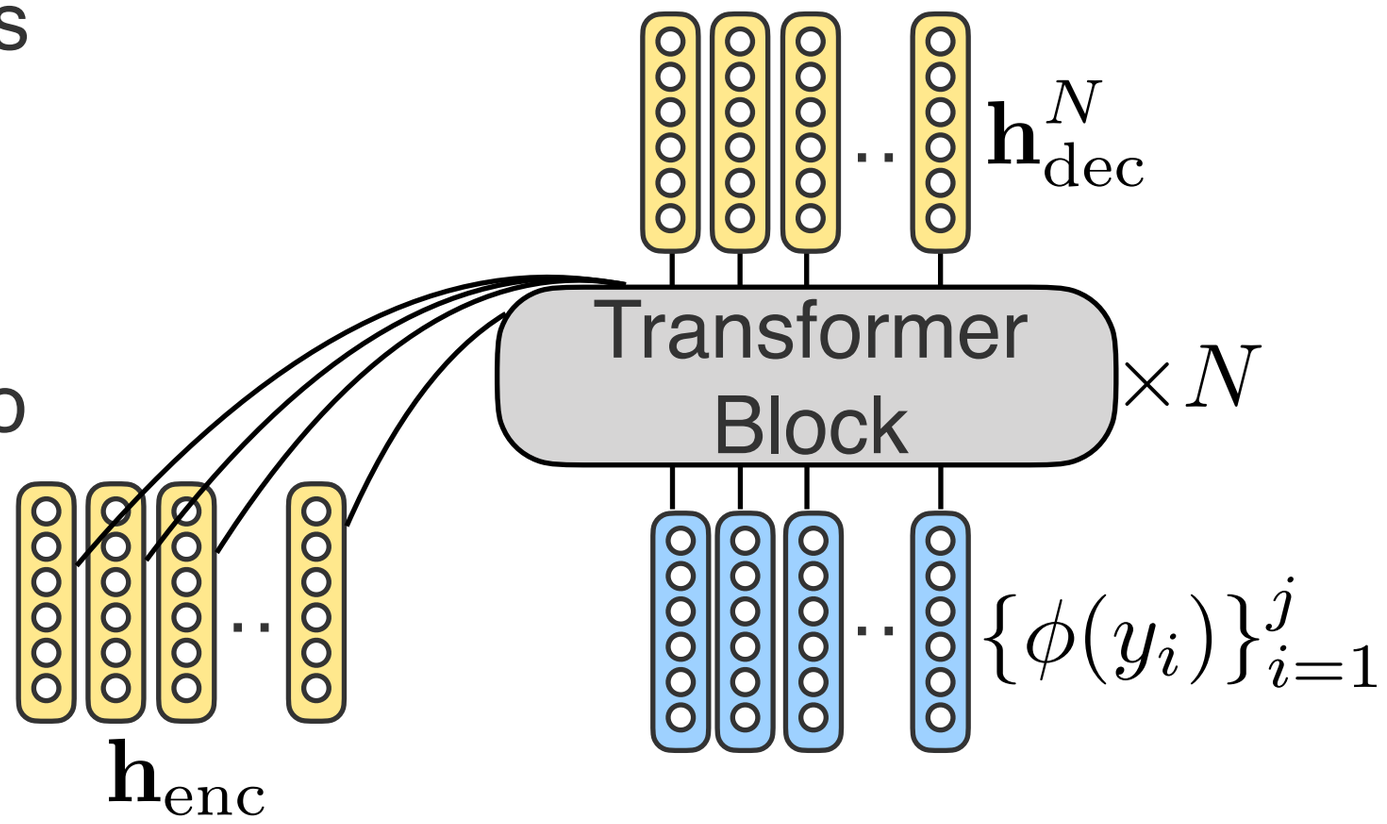
Transformer Decoder

$$\mathbf{h}_{\text{dec}}^{(l+1)} = \text{DecoderBlock}_{l+1}(\mathbf{h}_{\text{enc}}, \mathbf{h}_{\text{dec}}^{(l+1)})$$

$$\mathbf{h}_{\text{dec}}^{N} = \underset{\text{Decoder}}{\text{Transformer}}(\mathbf{h}_{\text{enc}}, \phi(y_1), \ldots, \phi(y_j))$$

$$\underset{\text{Decoder}}{\text{Transformer}} : \mathbb{R}^{d \times n} \times \mathcal{V}^j \rightarrow \mathbb{R}^{d \times j}$$

The transformer decoder maps from a set of input encodings and a sequence of tokens generated so far to a set of vectors, each corresponding to a token in the currently-generated sequence



Decoding

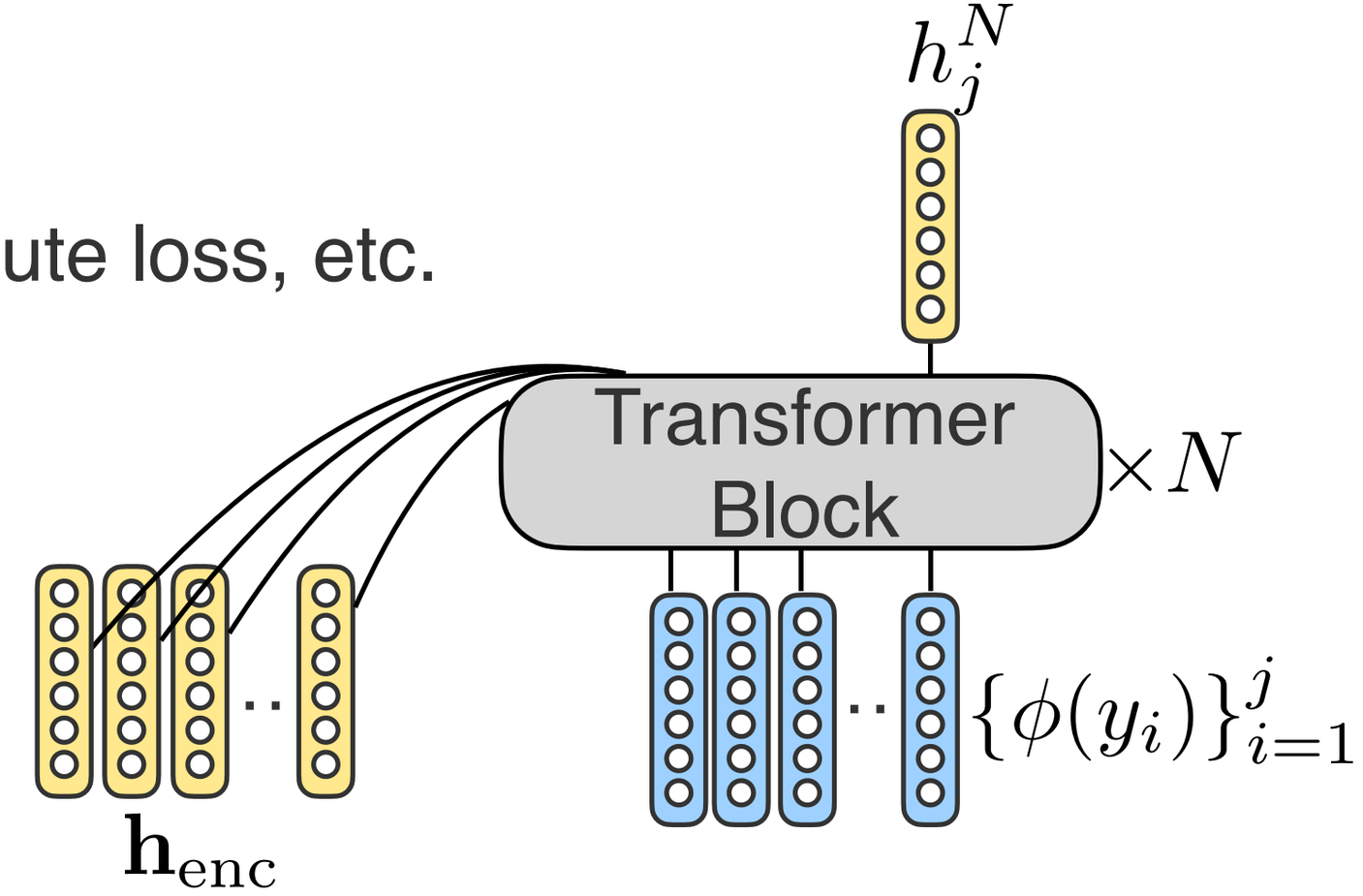
- Let's say we want to predict the word at index *j* + 1

- All we need to do is transform this into a distribution over the vocabulary

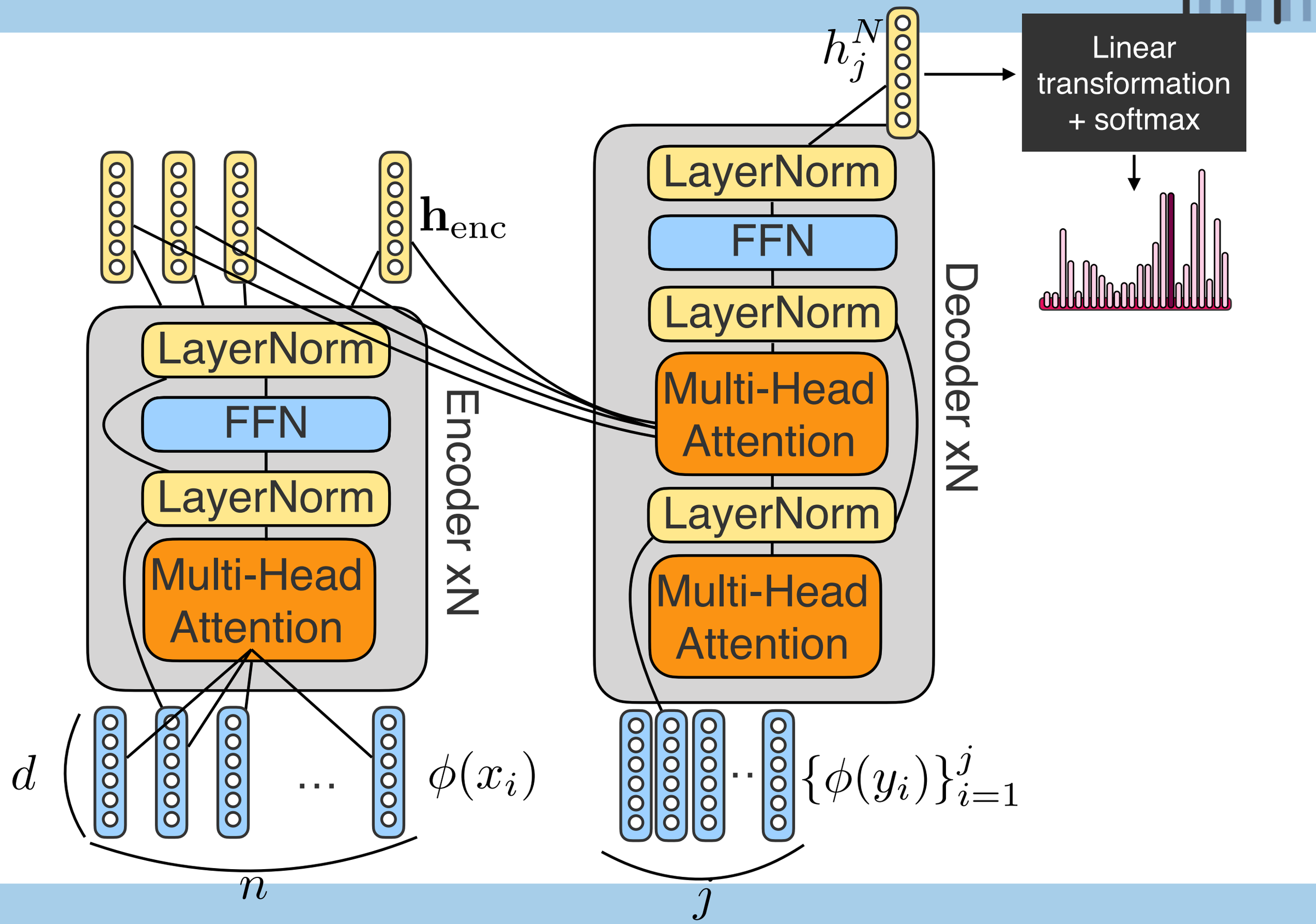
$$p(Y_{j+1} \mid x_1, \ldots, x_n, y_1, \ldots, y_j) = \text{softmax}(W h_j^N)$$

$$W \in \mathbb{R}^{|\mathcal{V}| \times d}$$

- Then we can sample, compute loss, etc.



Encoder-Decoder



Training

- Given some paired data $(\overline{x}, \overline{y})$:

$$\mathbf{h}_{\text{enc}} = \underset{\text{Encoder}}{\text{Transformer}}(\overline{x})$$

We can compute these in parallel on the GPU

$$p(y_{i+1}) = \underset{\text{Decoder}}{\text{Transformer}}(\mathbf{h}_{\text{enc}}, y_1, \ldots, y_i)$$

Probability of a token should only depend on the ones that come before it! But we still want to take advantage of parallelism…

Training

- Given some paired data $(\overline{x}, \overline{y})$:

$$\mathbf{h}_{\text{enc}} = \underset{\text{Encoder}}{\text{Transformer}}(\overline{x})$$

We can compute these in parallel on the GPU

$$p(y_{i+1}) = \underset{\text{Decoder}}{\text{Transformer}}(\mathbf{h}_{\text{enc}}, y_1, \ldots, y_i)$$

Probability of a token should only depend on the ones that come before it! But we still want to take advantage of parallelism…

- **Masking:**
  - When doing the forward pass on the decoder, self-attention can be parallelized
  - But we don't want the model to learn to rely on "future" words in the sequence!
  - Solution: during training, set $s_{ij} = -\infty$ if *i* (query index) < *j* (key/value index)

Unconditional Sequence Generation

- Recall the autoregressive approximation of sequence probability:

$$p(\overline{x}) = \prod_{i=1}^{|\overline{x}|} p(x_i \mid x_1, \ldots, x_{i-1})$$

- Can we do this with a transformer model?

Unconditional Sequence Generation

- Recall the autoregressive approximation of sequence probability:

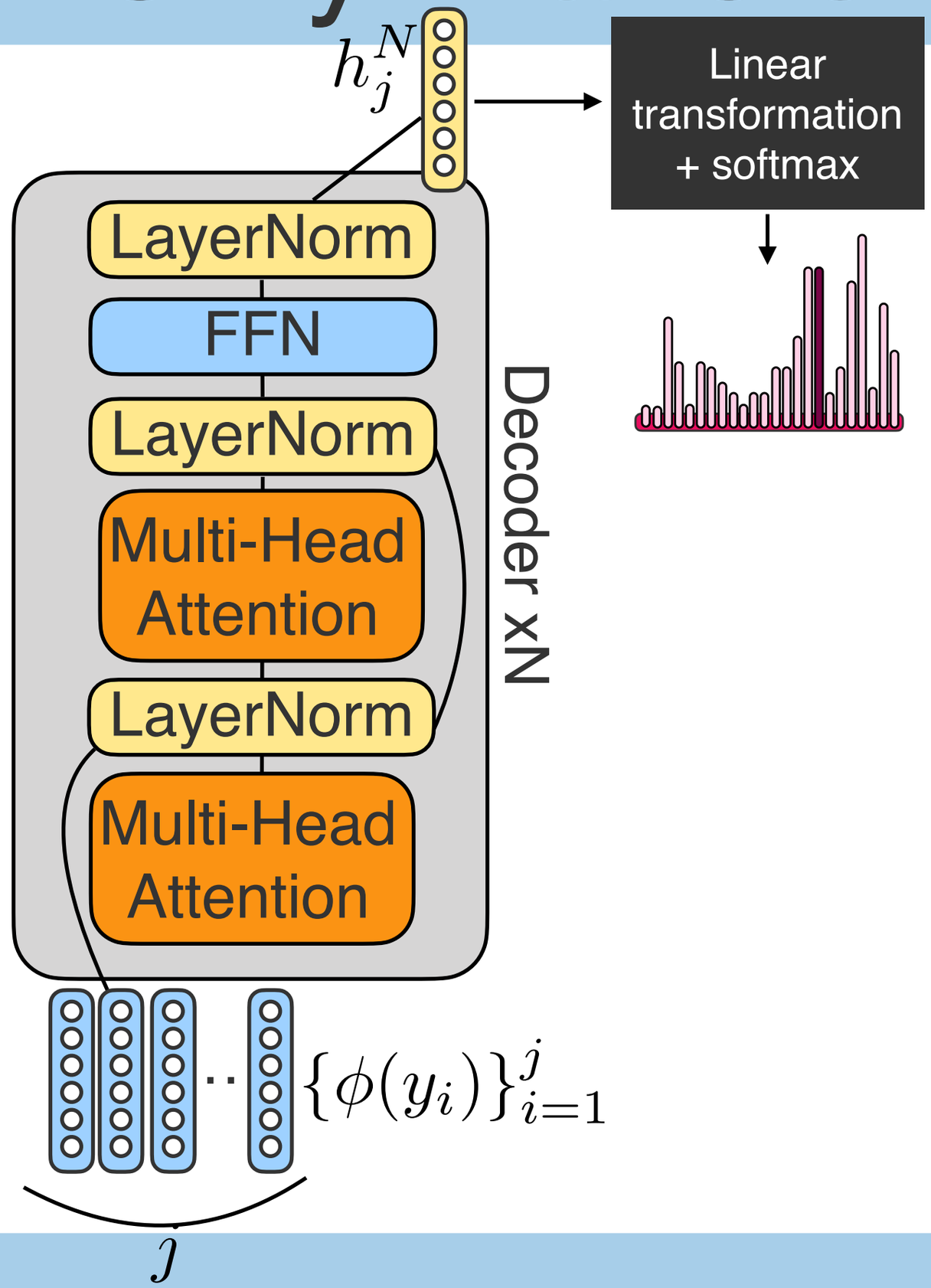
$$p(\overline{x}) = \prod_{i=1}^{|\overline{x}|} p(x_i \mid x_1, \ldots, x_{i-1})$$

- Can we do this with a transformer model?

- **Yes!** We just don't have an encoder (i.e., decoder-only LLM).

- We still have to be careful about masking while training.

Decoder-Only Transformer



Sequence Encoding

- What if we just want some representation of a sentence, and we don't really care about being able to generate sentences? $\mathrm{Enc}(\overline{x})$
- We get this with the transformer encoder:

$$\mathbf{h}_{\mathrm{enc}} = \underset{\mathrm{Encoder}}{\mathrm{Transformer}}(\overline{x})$$

- But how might we train it?

Masked Language Modeling

Masked language modeling:

- The probability of a sequence is a product of local token probabilities
- The probability of a token depends on the ones that came before *and after* it

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1} \ldots x_n)$$

The name *anteater* refers to the species' [redacted], which consists mainly of ants and termites.

Masked Language Modeling

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1} \ldots x_n)$$

- From our transformer encoder, we get

$$h_i = \underset{\text{Encoder}}{\text{Transformer}}(\overline{x})_i \in \mathbb{R}^d$$

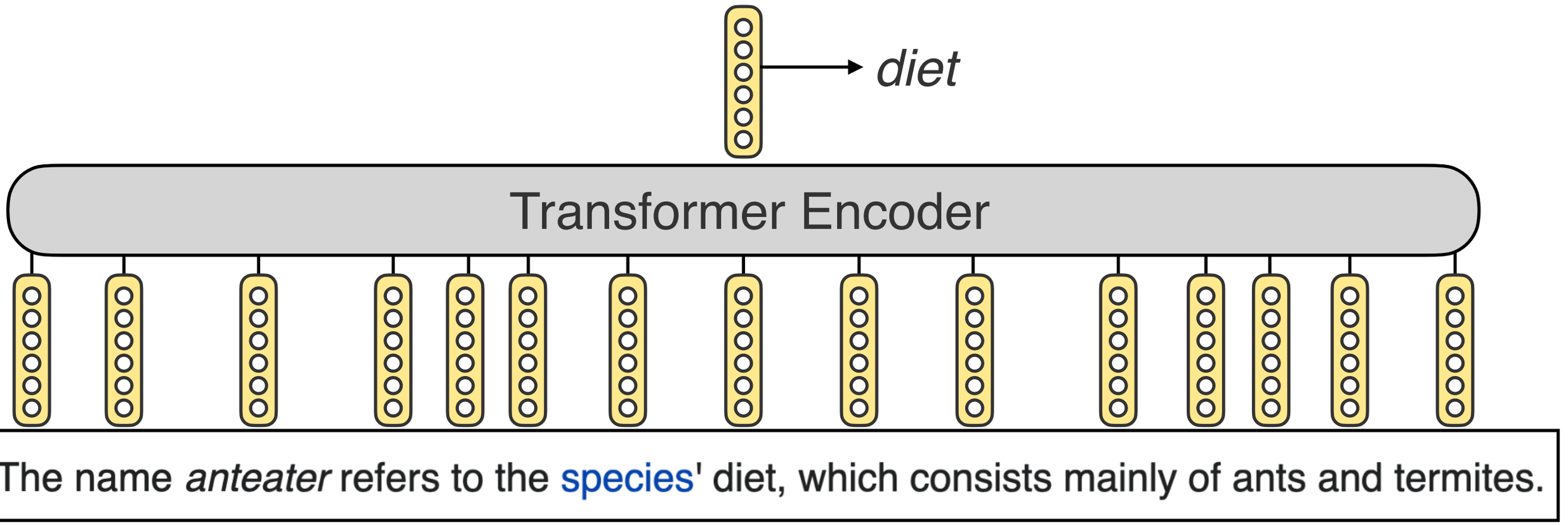
- We could just transform this to the vocabulary space, and compute the loss from there:

$$p(X_i) = \text{softmax}(W h_i)$$

- Do you anticipate any problems?

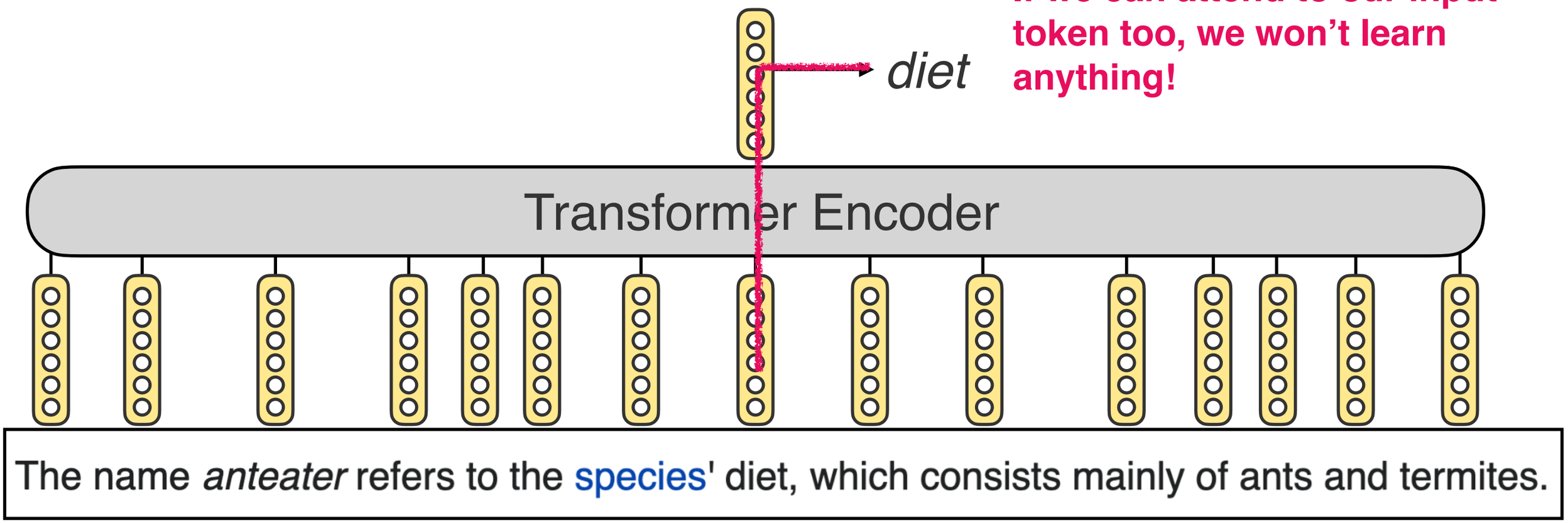
Masked Language Modeling

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1} \ldots x_n)$$



Masked Language Modeling

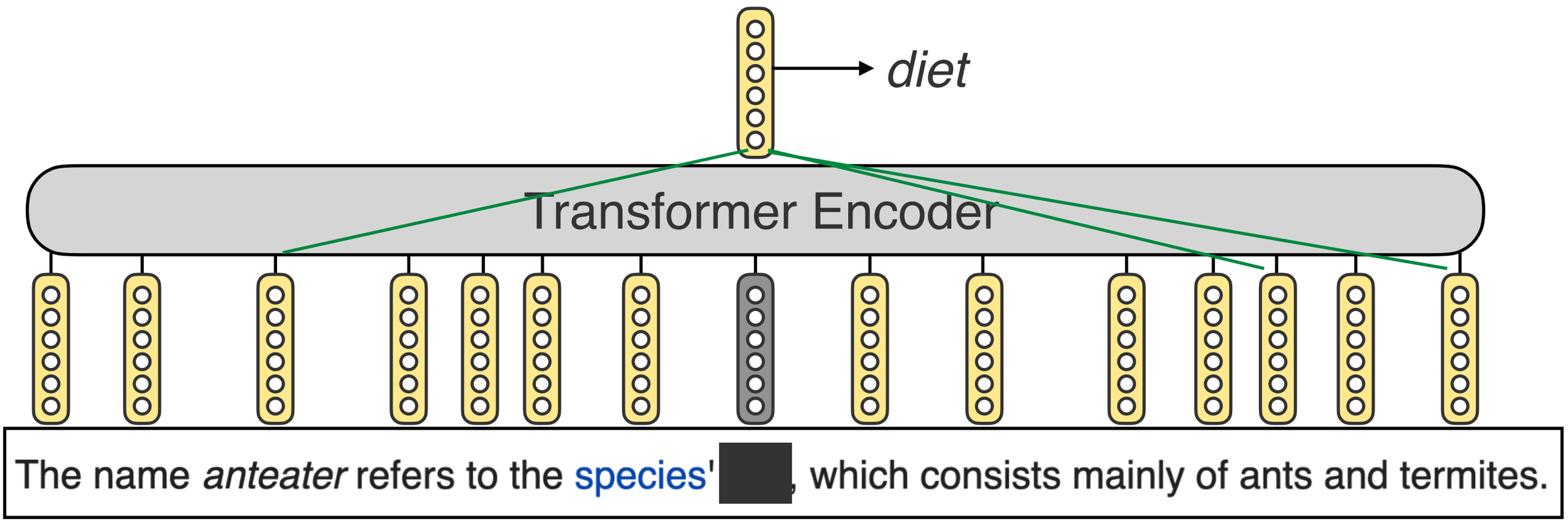
$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1} \ldots x_n)$$



If we can attend to our input token too, we won't learn anything!

Masked Language Modeling

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1} \ldots x_n)$$



Instead: randomly sample some tokens to replace with a MASK placeholder token in the input, then train the model to predict those words

Masked Language Modeling

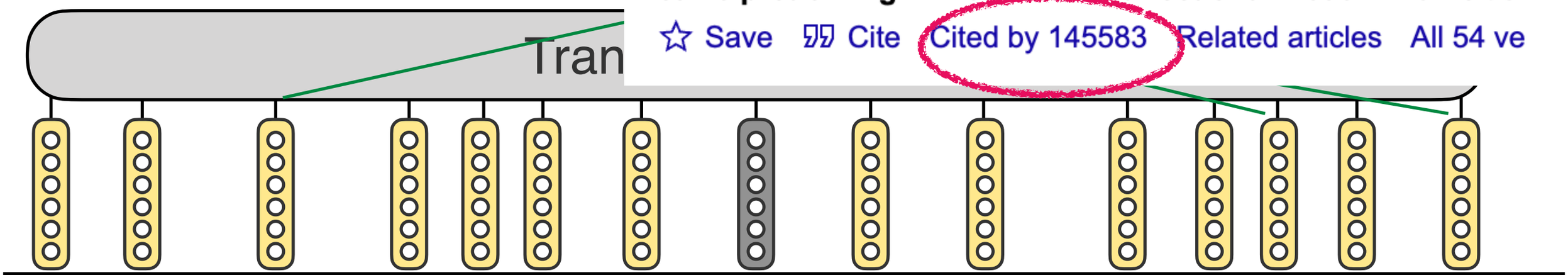
$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1} \ldots x_n)$$

Bert: Pre-training of deep bidirectional transfor understanding

J Devlin, MW Chang, K Lee… - … : human **language** …, 2019 - a

… **deep** bidirectionality of **BERT** by evaluating two **pretraining** ob same **pretraining** … No NSP: A **bidirectional** model which is trai

☆ Save 99 Cite Cited by 145583 Related articles All 54 ve



Instead: randomly sample some tokens to replace with a MASK placeholder token in the input, then train the model to predict those words