Attention

EECS 183/283a: Natural Language Processing

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

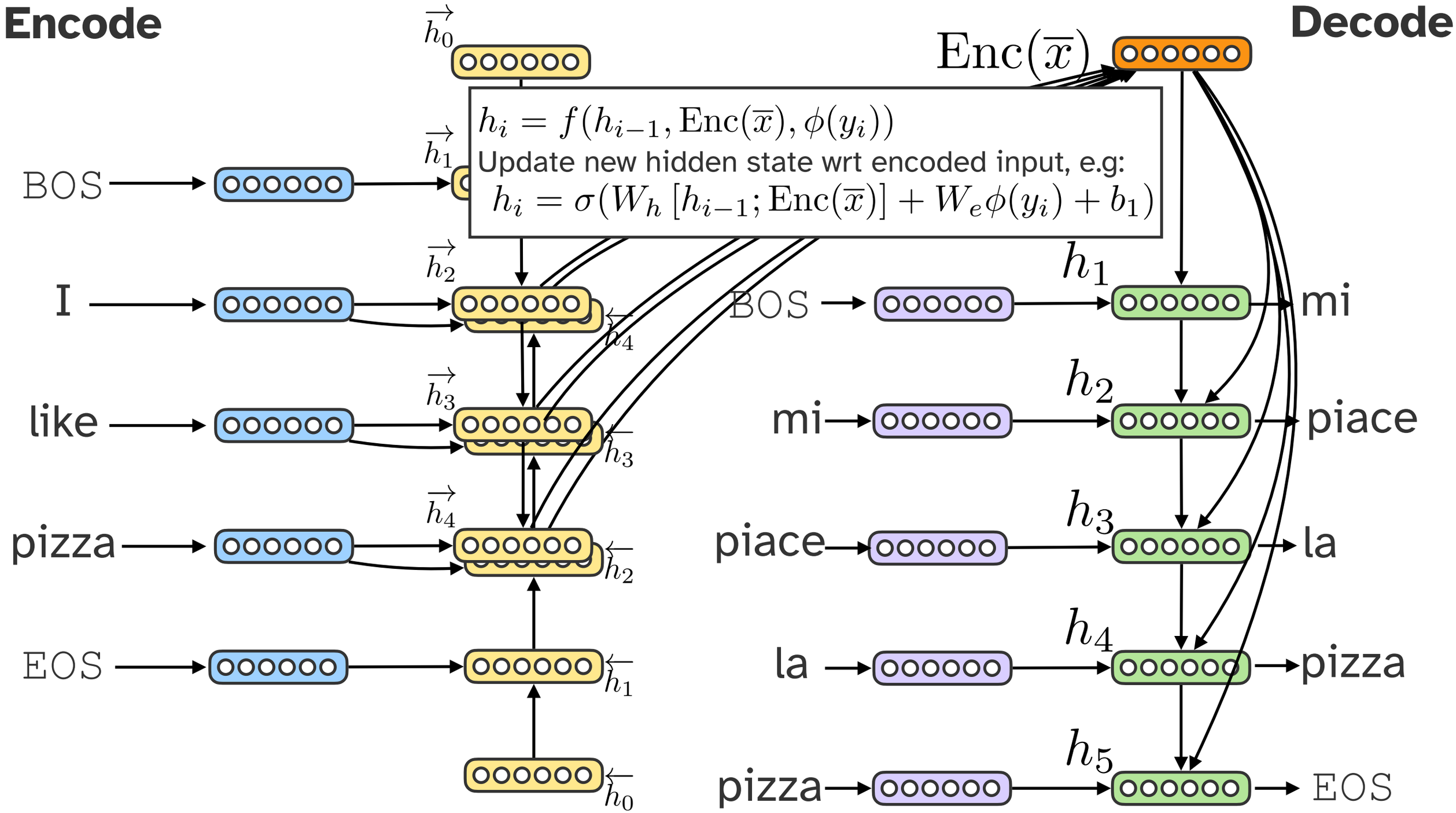
Problem 1
Long-distance dependencies are harder to represent than shorter ones.

*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**.*

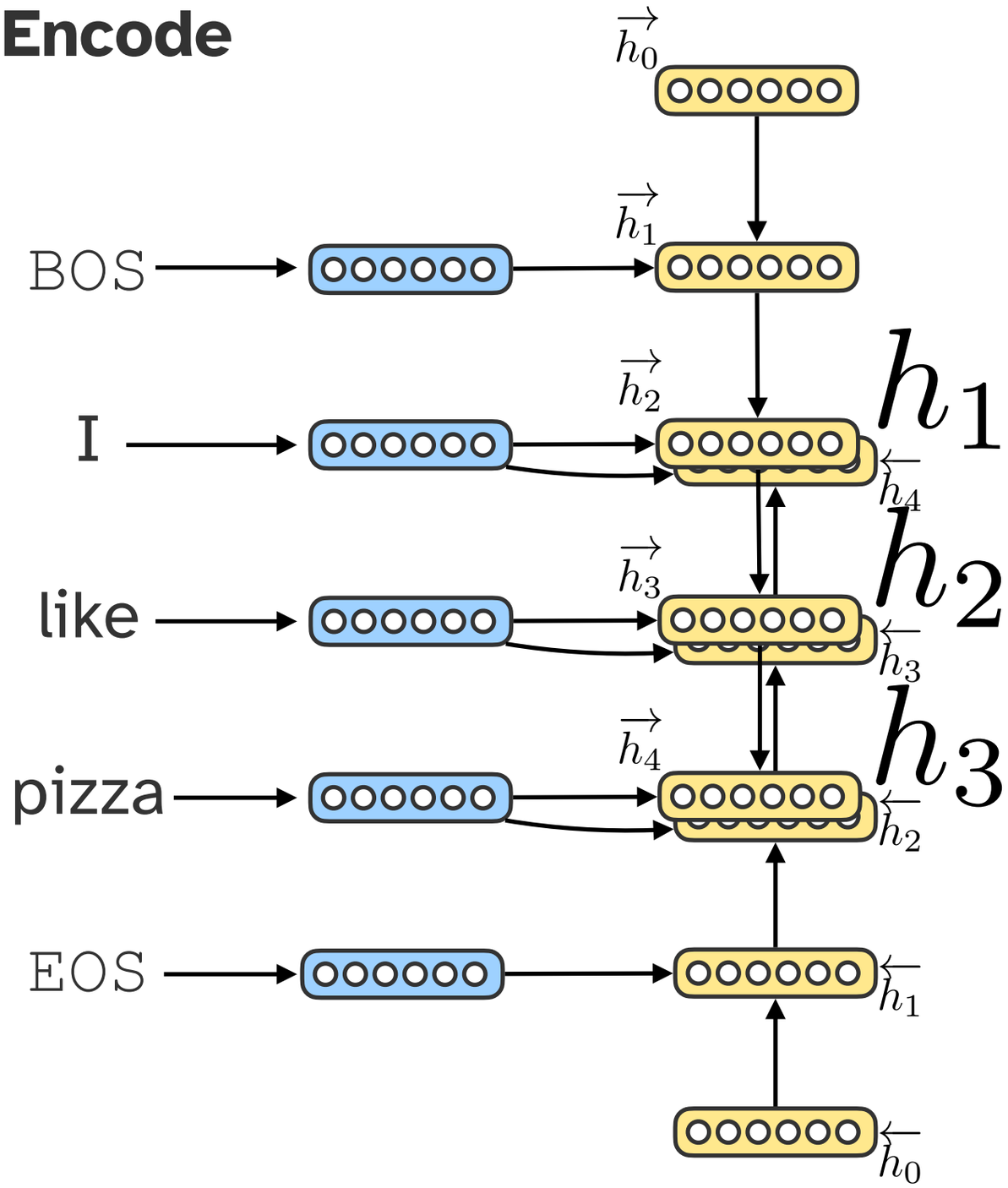
Problem 3
We have to encode the input sequence token-by-token O(n)

Recap:
Problem 1: Long-Distance Dependencies



Recap:
Problem 2: Fixed-Size Sequence Embedding

Encode



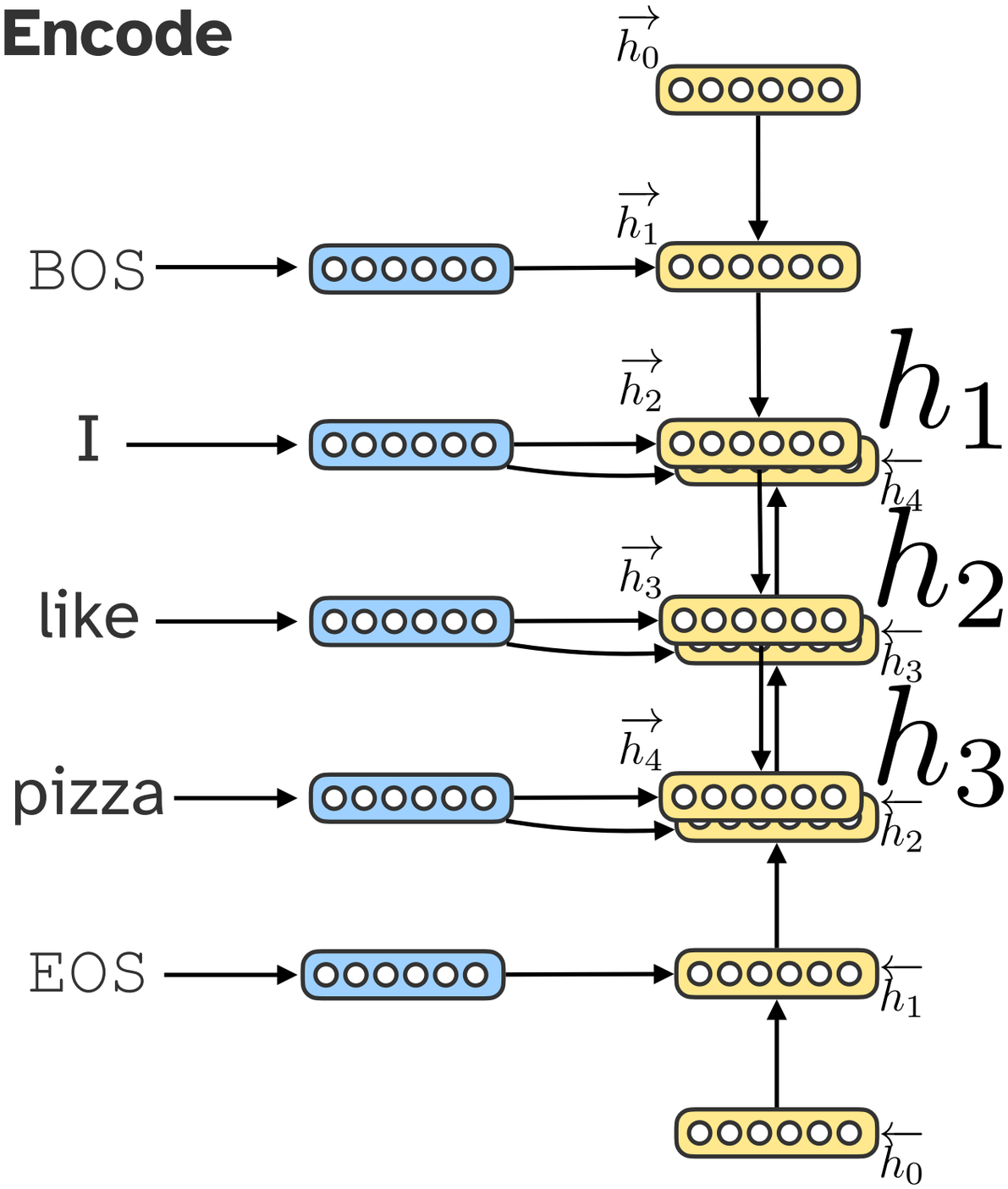
$$\text{Enc}(\overline{x}) = \sum_{i=1}^{|\overline{x}|} p(i) h_i$$

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

- 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

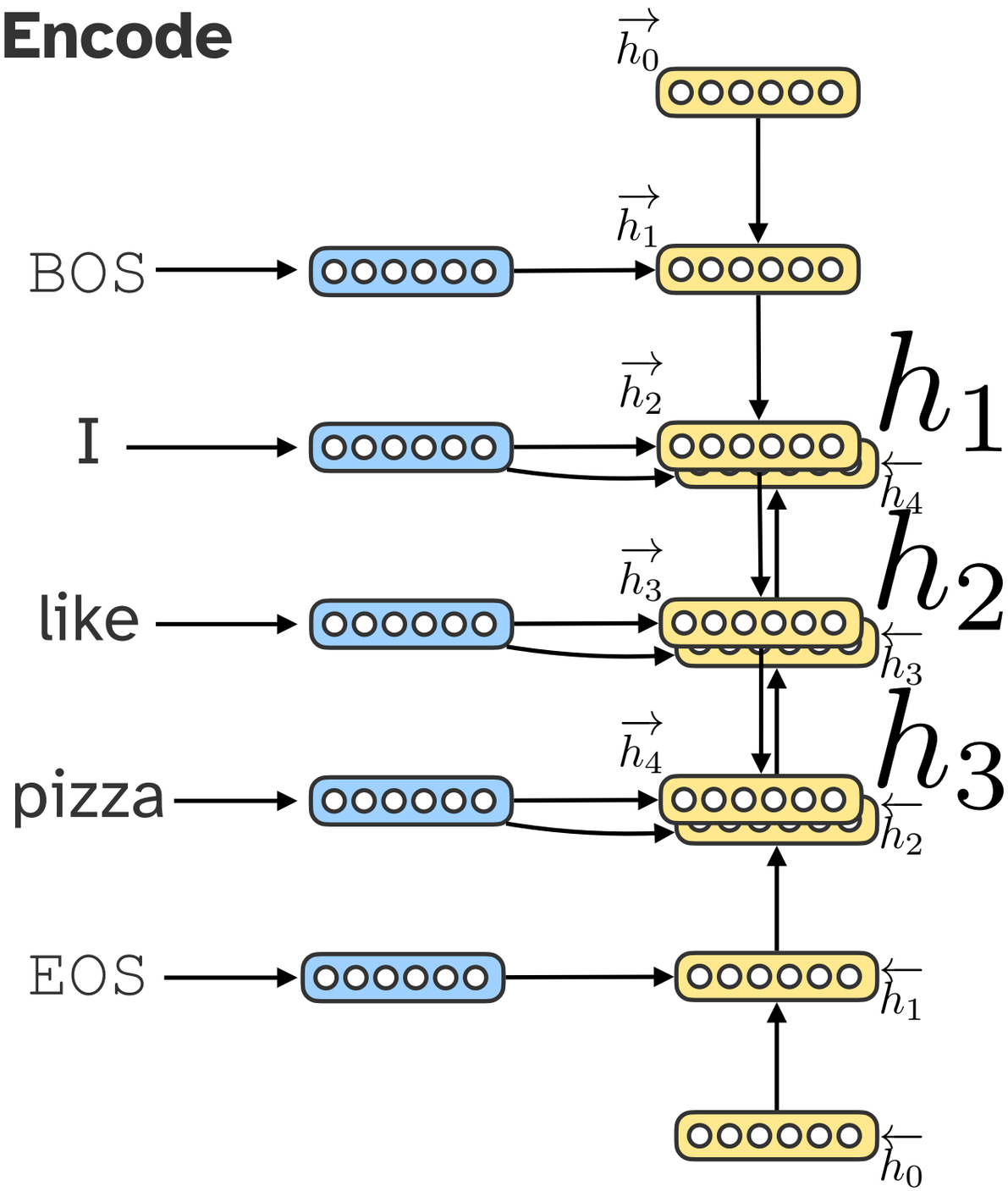


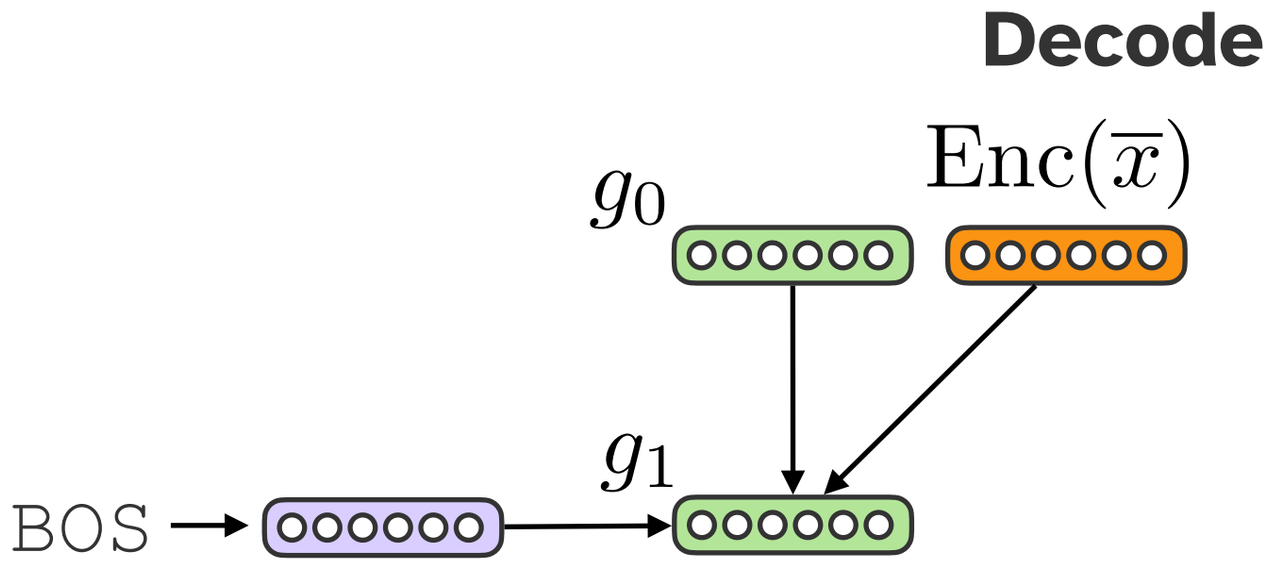
$$\mathrm{Enc}(\overline{x}, g) = \sum_{i=1}^{|\overline{x}|} p(i \mid g) h_i$$

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

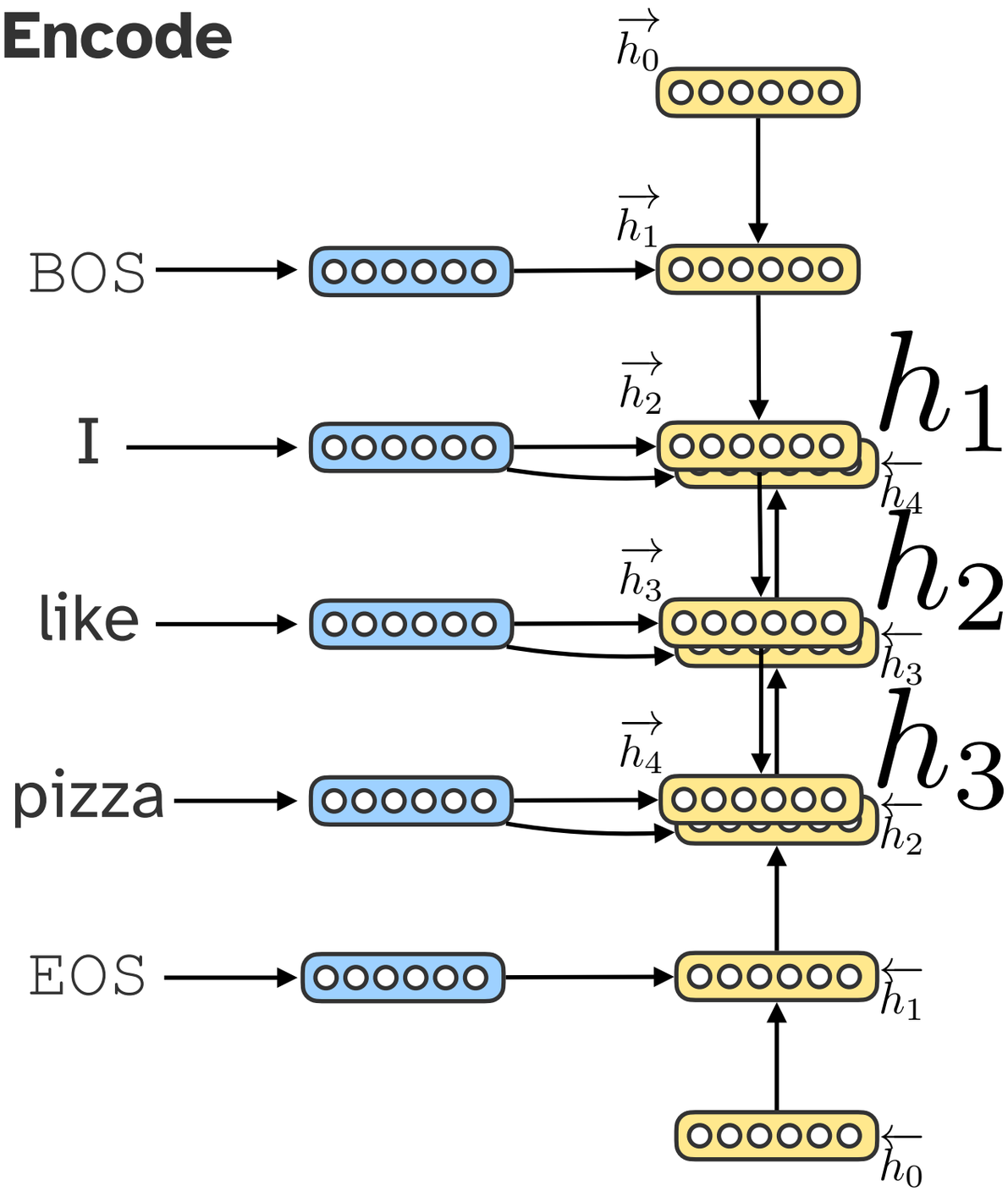
- Compute a new distribution over input indices depending on some other vector *g*
- Two questions:
  - Where does *g* come from?
  - How to implement *p*?

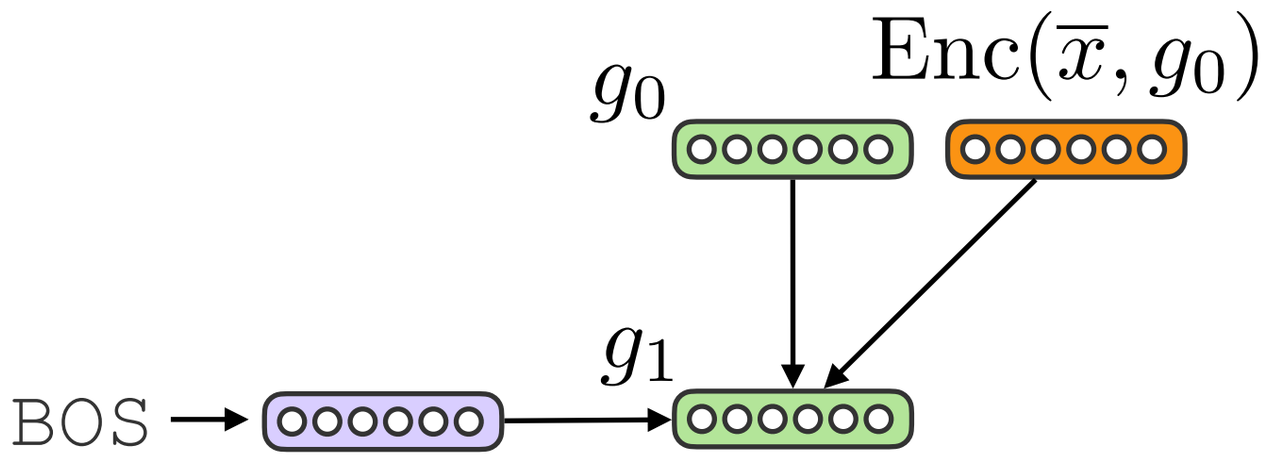
Recap: Attention





Recap: Attention





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

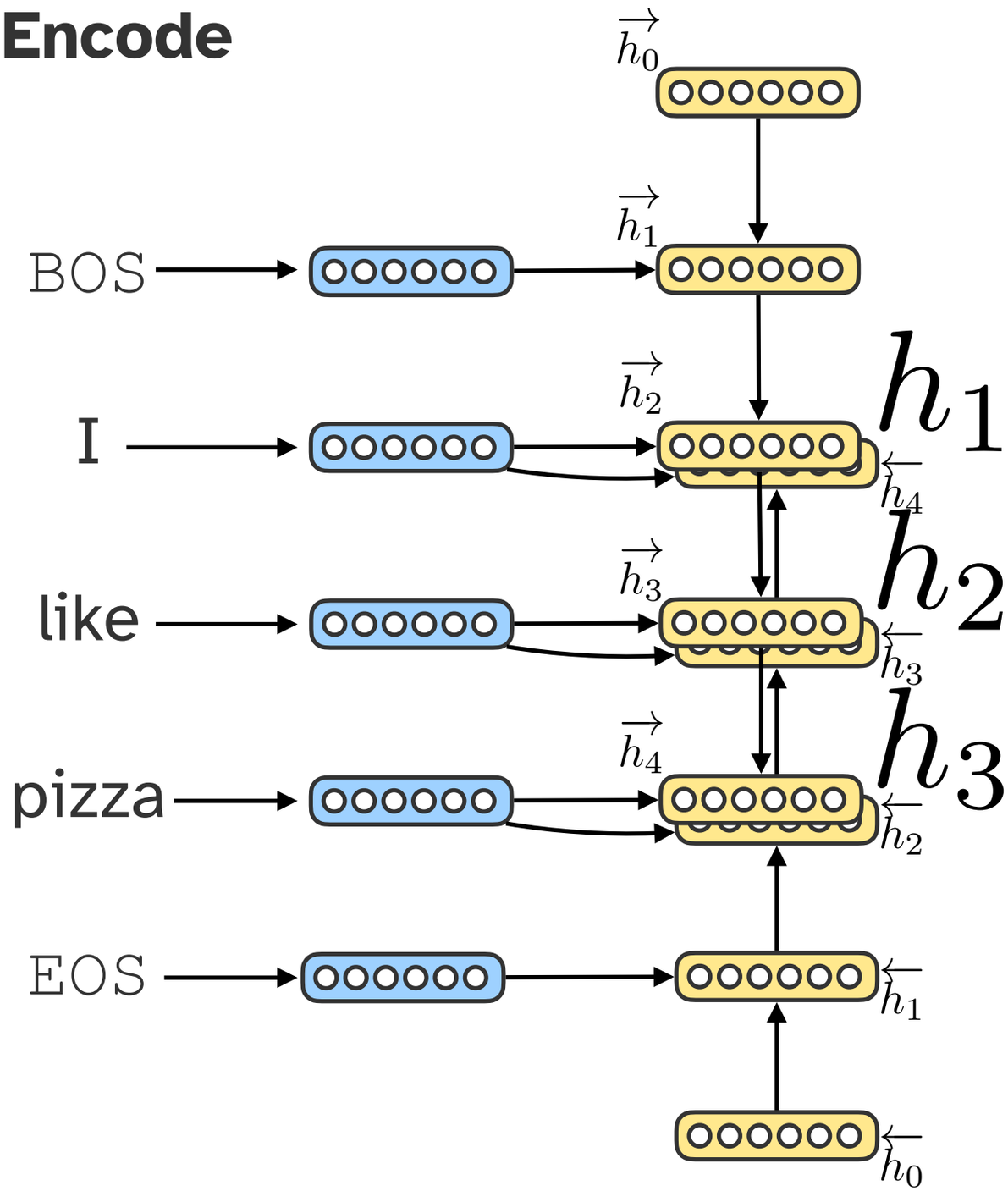
text encoding also depends on the previous state!

Recall:

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

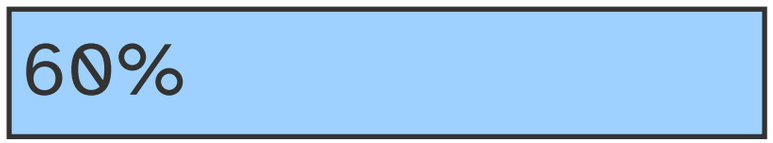
Recap: Attention

Encode



Decode

$p(I \mid g_0)$



h_1

h_2

h_3

Weighted sum:

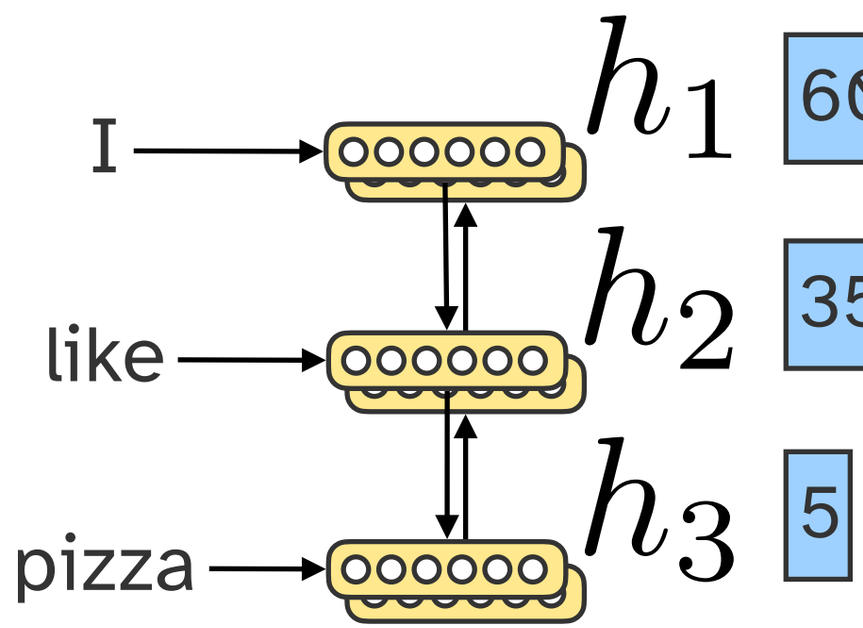
$$\text{Enc}(\overline{x}, g_0) = \sum_{i=1}^{|\overline{x}|} p(i \mid g_0) h_i$$

Recall:

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

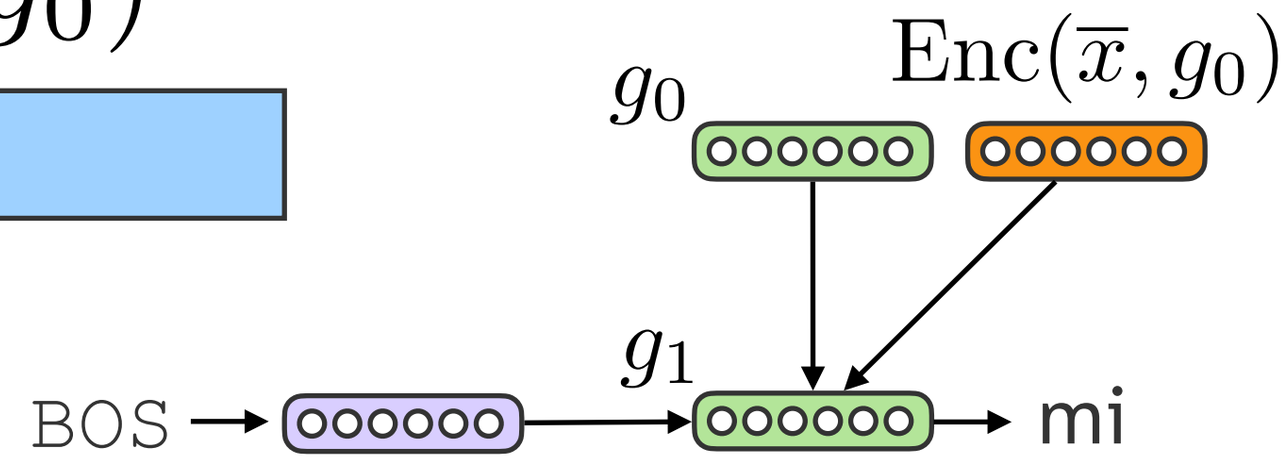
Recap: Attention

Encode



$p(I \mid g_0)$

Decode

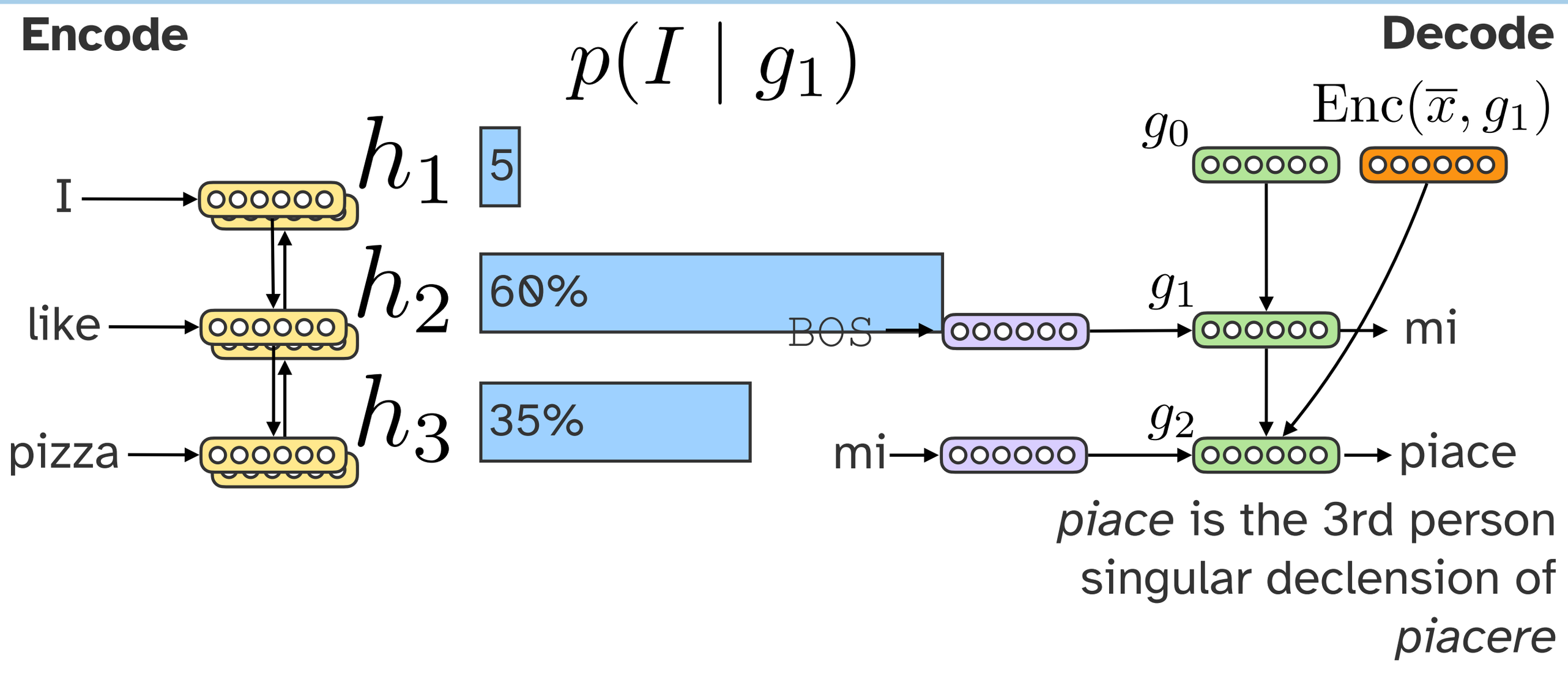


mi is the 1st person, objective case pronoun

Recall:

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

Recap: Attention



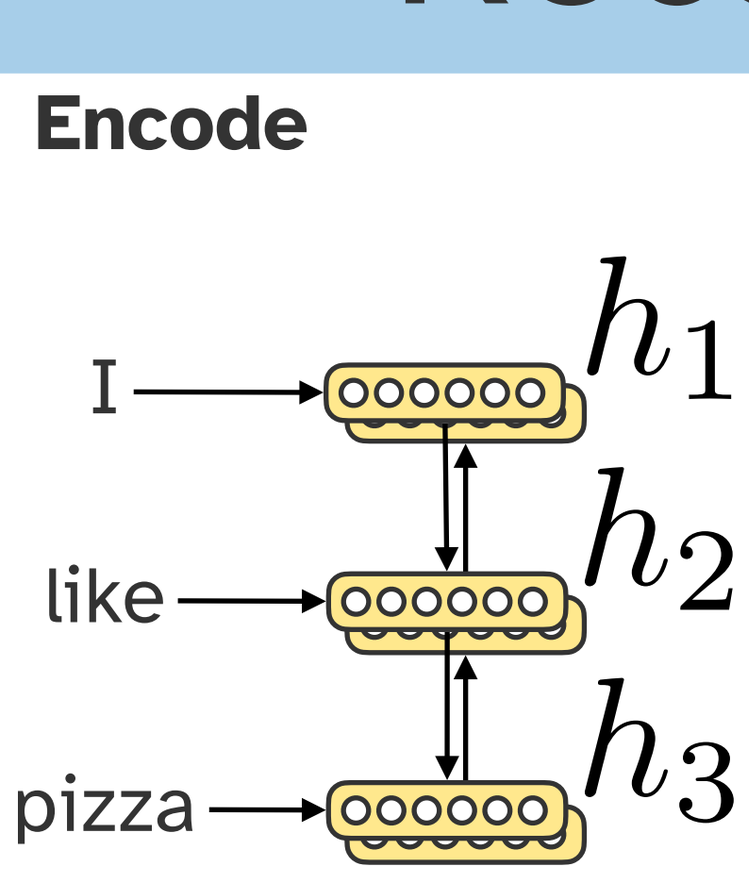
piace is the 3rd person singular declension of *piacere*

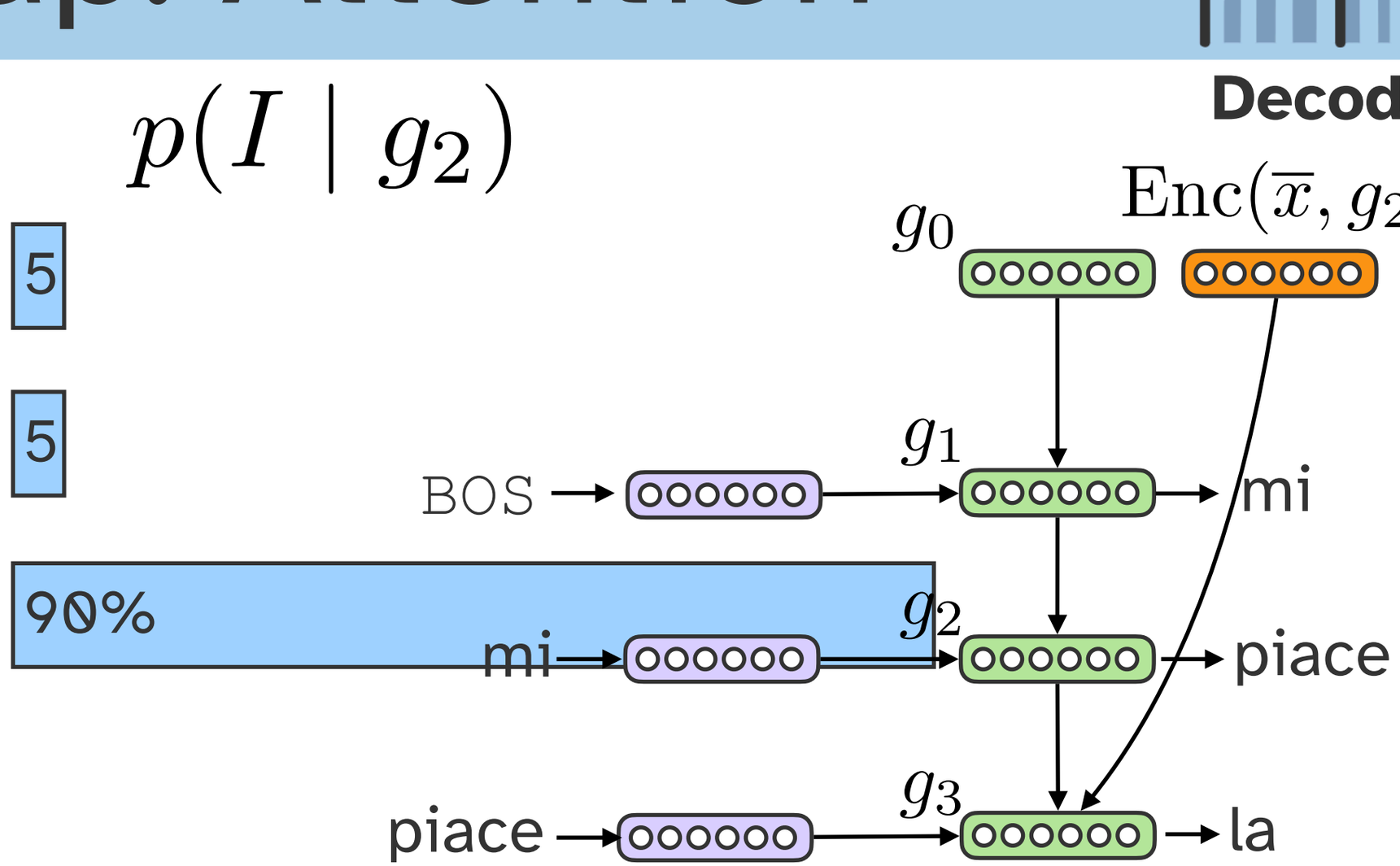
Recall:

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

Recap: Attention

$$p(I \mid g_2)$$





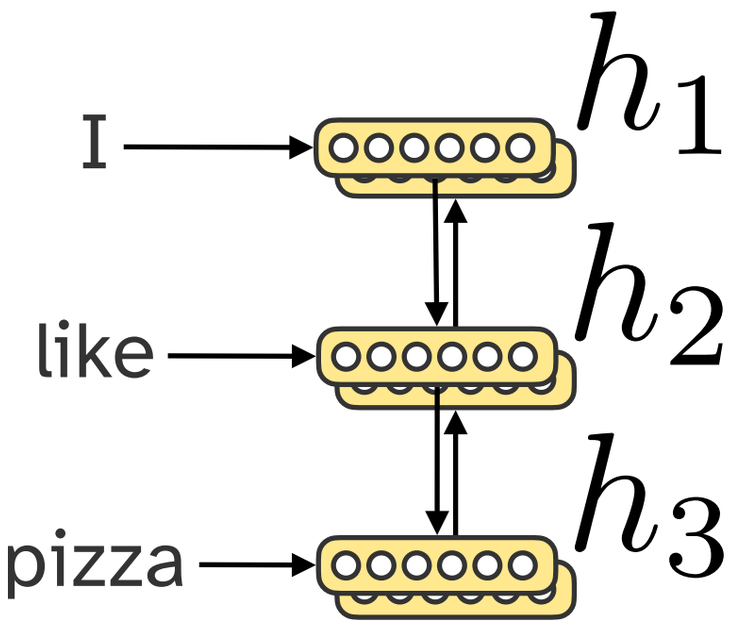
Recall:

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

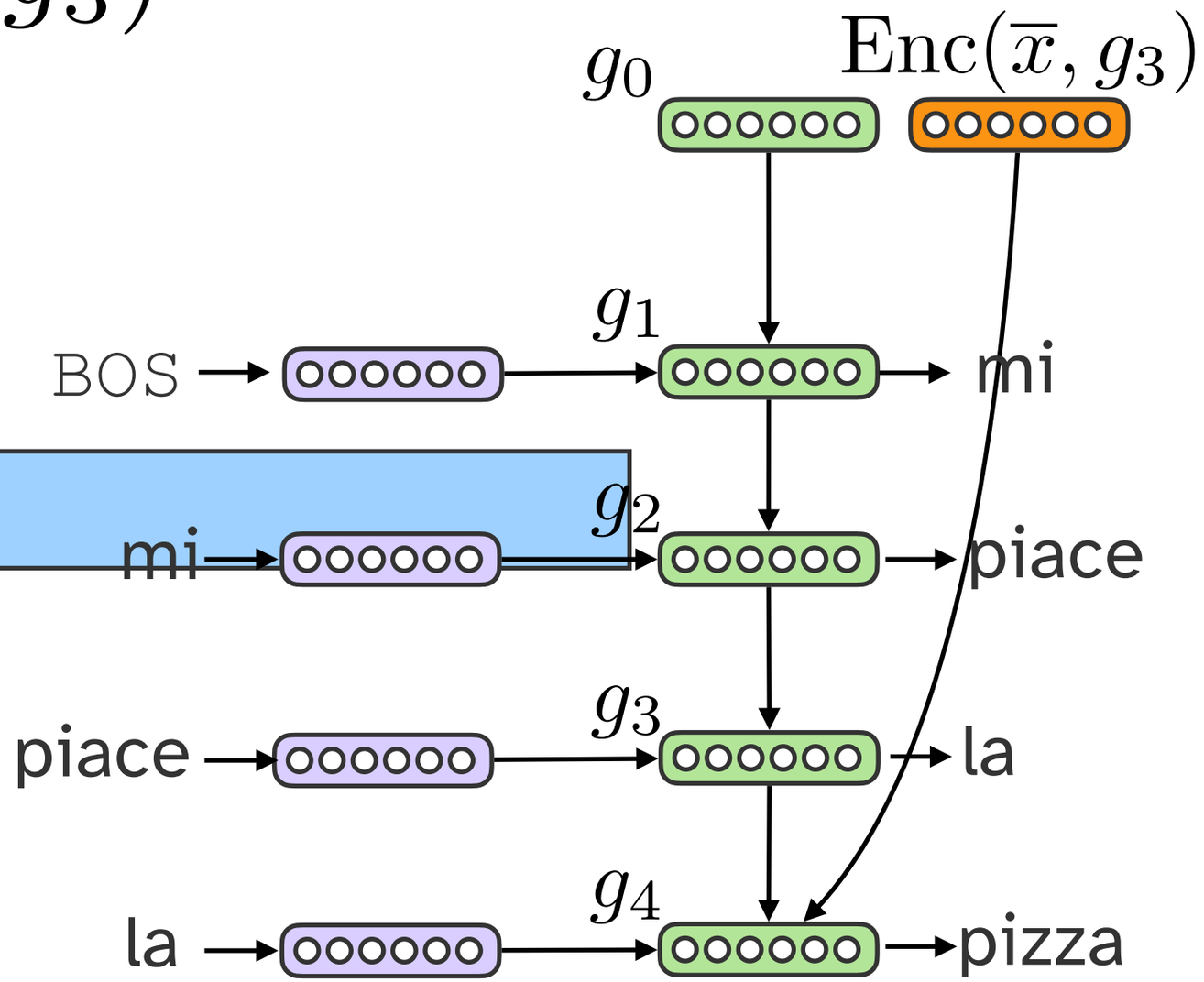
Recap: Attention

$$p(I \mid g_3)$$

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 $p(I \mid g)$?

Computing Attention

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

We compute weighted mean pooling over input hidden states, where the weights are dependent on some vector *g*

(usually the current decoder hidden state)

weights are determined by a probability distribution over input token indices

- Question 1: where does *g* come from? → hidden state of decoder
- Question 2: how do we compute $p(i \mid g)$?

Query, Key, Value

- An analogy: databases
- A **query** is some search term
- A table has **keys** paired with **values**
- We want to return the **values** of the **keys** most related to the **query**

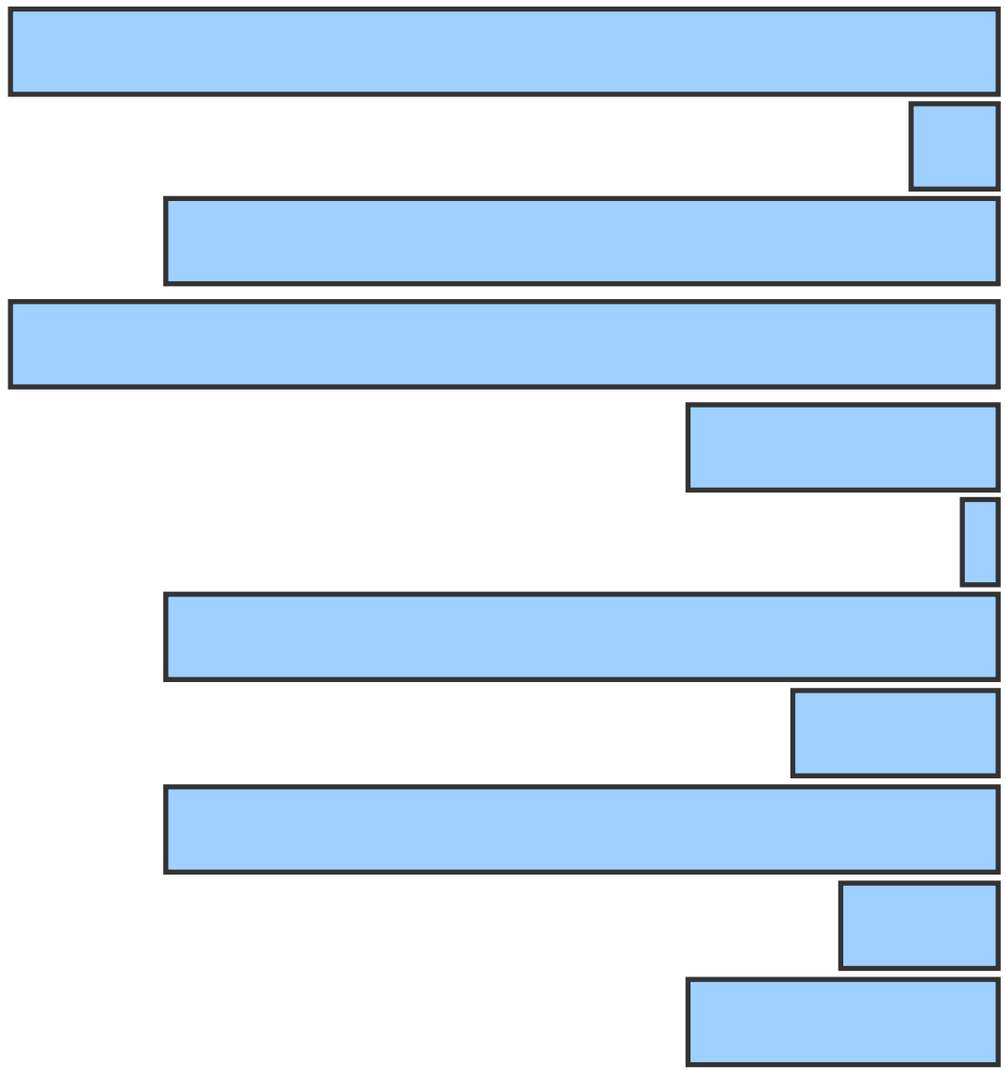
Query: what are some words for mammals in Turkish?

i	Key	Value
1	sheep	koyun
2	river	ırmak
3	author	yazar
4	bull	boğa
5	fly	sinek
6	radio	radyo
7	farmer	çiftçi
8	plant	bitki
9	actress	aktris
10	home	ev
11	snail	salyangoz

Query, Key, Value

$$p(i \mid q, k_i) \propto \mathrm{sim}(\mathrm{Enc}(q), \phi(k_i))$$

Weight of each item is proportional to the similarity of its key with the query



Query: what are some words for mammals in Turkish?

i	Key	Value
1	sheep	koyun
2	river	ırmak
3	author	yazar
4	bull	boğa
5	fly	sinek
6	radio	radyo
7	farmer	çiftçi
8	plant	bitki
9	actress	aktris
10	home	ev
11	snail	salyangoz

Query, Key, Value

$$p(i \mid q, k_i) \propto \mathrm{sim}(\mathrm{Enc}(q), \phi(k_i))$$

Weight of each item is proportional to the similarity of its key with the query

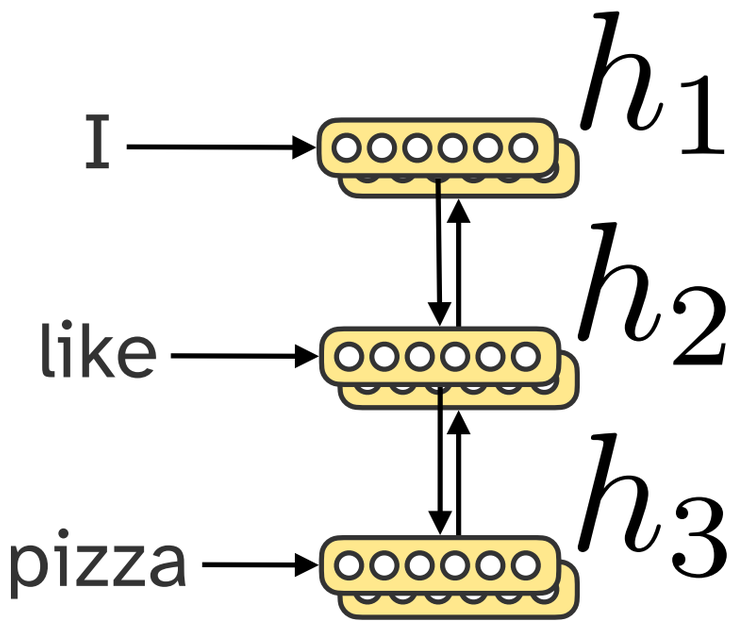
Query: what are some words for mammals in Turkish?

$$p(i \mid q, k_i) v_i$$

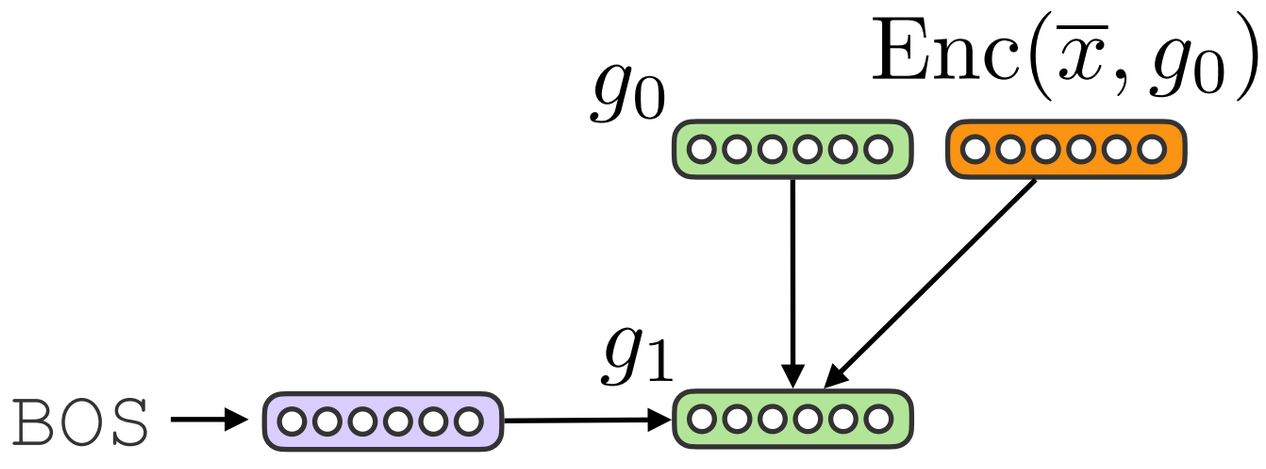
Weight each value accordingly

Back to RNNs...

Encode



Decode



$$p(i \mid q, k_i) \propto \text{sim}(\text{Enc}(q), \phi(k_i)) \equiv p(i \mid g, h_i) \propto \text{sim}(g, h_i)$$

Query Key

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

Value

Back to RNNs...

- **Query:** current decoder hidden state
- **Key:** encoder hidden state (at some index *i*)
- **Value:** also the encoder hidden state at index *i*

$$p(i \mid g, h_i) \propto \text{sim}(g, h_i)$$

Query Key

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

Value

Query: g_1

i	Key	Value
1	h_1	h_1
2	h_2	h_2
3	h_3	h_3

Until I say otherwise, let's assume our keys and values are identical

Similarity Between Query and Key

$$p(i \mid q, k_i) \propto \text{sim}(q, k_i)$$

$$= \frac{e^{\text{sim}(q,k_i)}}{\sum_{i=1}^{|\overline{x}|} e^{\text{sim}(q,k_{i'})}}$$

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

$$\mathbf{k} \in \mathbb{R}^{d' \times |\overline{x}|}$$

Similarity Between Query and Key

$$p(i \mid q, k_i) \propto \mathrm{sim}(q, k_i)$$

$$= \frac{e^{\mathrm{sim}(q, k_i)}}{\sum_{i=1}^{|\overline{x}|} e^{\mathrm{sim}(q, k_{i'})}}$$

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

$$\mathbf{k} \in \mathbb{R}^{d' \times |\overline{x}|}$$

$$\mathrm{sim}(q, \mathbf{k}) = q^\intercal W \mathbf{k}$$

Bilinear attention

$$W \in \mathbb{R}^{d \times d'}$$

Similarity Between Query and Key

$$p(i \mid q, k_i) \propto \mathrm{sim}(q, k_i)$$

$$= \frac{e^{\mathrm{sim}(q,k_i)}}{\sum_{i=1}^{|\overline{x}|} e^{\mathrm{sim}(q,k_{i'})}}$$

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

$$\mathbf{k} \in \mathbb{R}^{d' \times |\overline{x}|}$$

$$\mathrm{sim}(q, \mathbf{k}) = w_2^\intercal \tanh(W_1 [q; \mathbf{k}])$$

MLP attention

$$W_1 \in \mathbb{R}^{(d+d') \times d''}$$

$$w_2 \in \mathbb{R}^{d''}$$

Similarity Between Query and Key

$$p(i \mid q, k_i) \propto \text{sim}(q, k_i)$$

$$= \frac{e^{\text{sim}(q,k_i)}}{\sum_{i=1}^{|\overline{x}|} e^{\text{sim}(q,k_{i'})}}$$

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

$$\mathbf{k} \in \mathbb{R}^{d' \times |\overline{x}|}$$

$$\text{sim}(q, \mathbf{k}) = q^\intercal \mathbf{k}$$

Dot product attention

$$d = d'$$

But: scale of dot product increases as dimensions get larger

Similarity Between Query and Key

$$p(i \mid q, k_i) \propto \mathrm{sim}(q, k_i)$$

$$= \frac{e^{\mathrm{sim}(q, k_i)}}{\sum_{i=1}^{|\overline{x}|} e^{\mathrm{sim}(q, k_{i'})}}$$

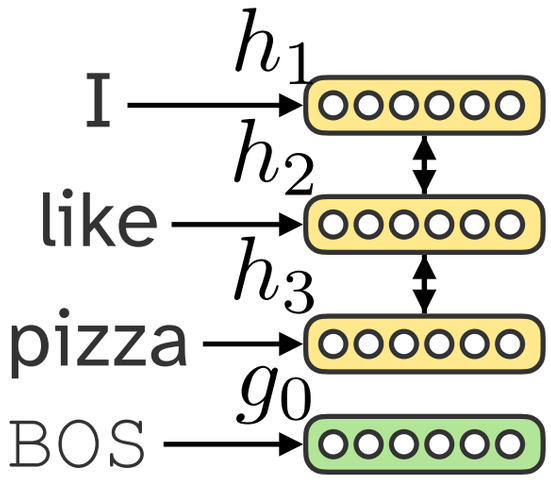
$$q \in \mathbb{R}^d$$

$$\mathbf{k} \in \mathbb{R}^{d' \times |\overline{x}|}$$

$$\mathrm{sim}(q, \mathbf{k}) = \frac{q^\intercal \mathbf{k}}{\sqrt{d'}}$$

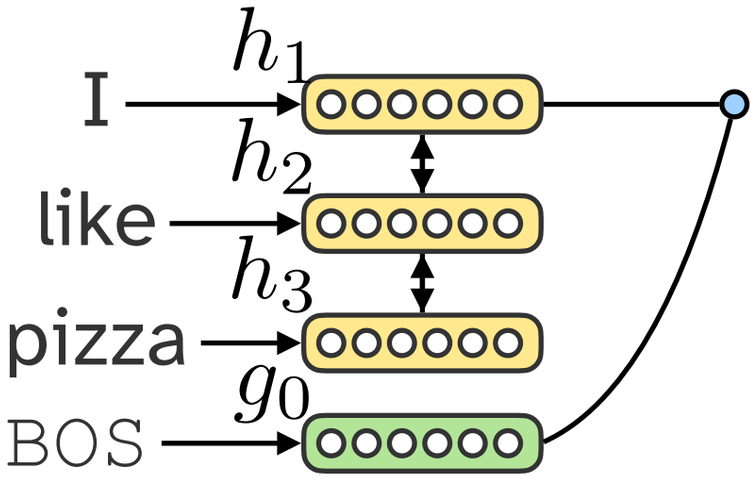
Scaled dot product attention

Illustration:
Dot Product Attention



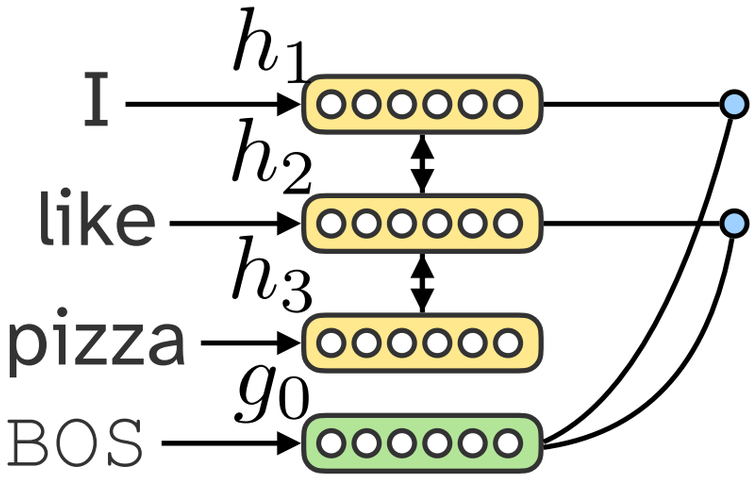
(note: many possible ways to initialize g_0 *)*

Illustration:
Dot Product Attention



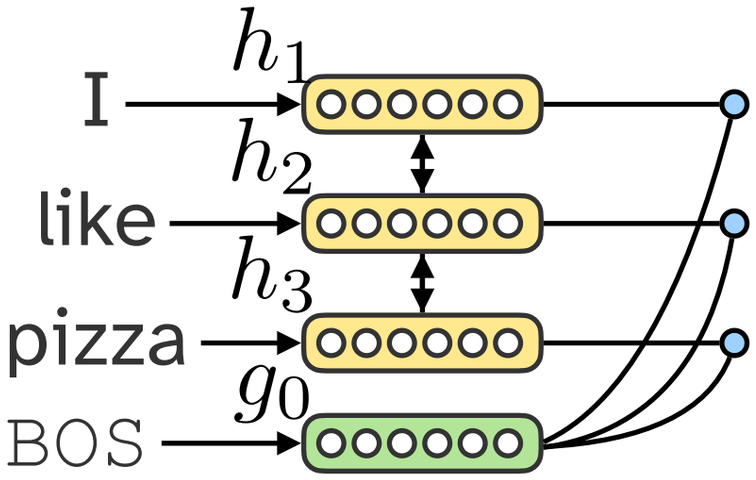
$$s_1 = g_0^\intercal h_1$$

Illustration:
Dot Product Attention



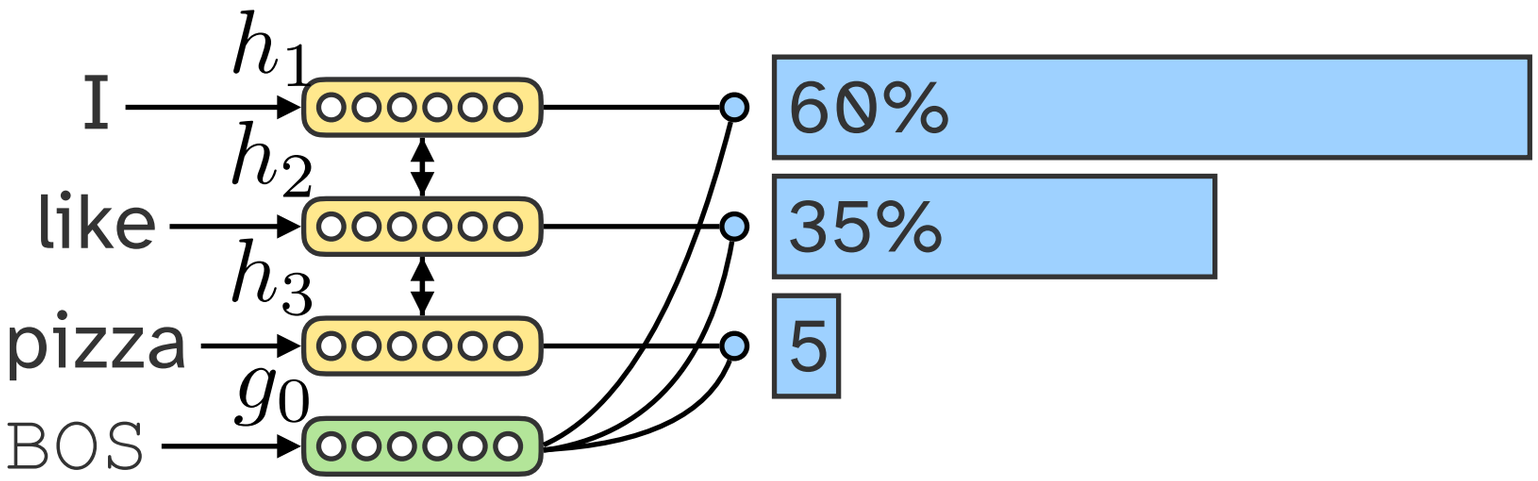
$$s_2 = g_0^\intercal h_2$$

Illustration:
Dot Product Attention



$$s_3 = g_0^\intercal h_3$$

Illustration: Dot Product Attention



Similarities between query and keys

$$s_0 = g_0^\intercal \mathbf{h} \in \mathbb{R}^{|\overline{x}|}$$

Weights over keys (weights sum to 1)

$$\alpha_0 = p(\cdot \mid g_0, \mathbf{h}) = \mathrm{softmax}(s_0) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$$

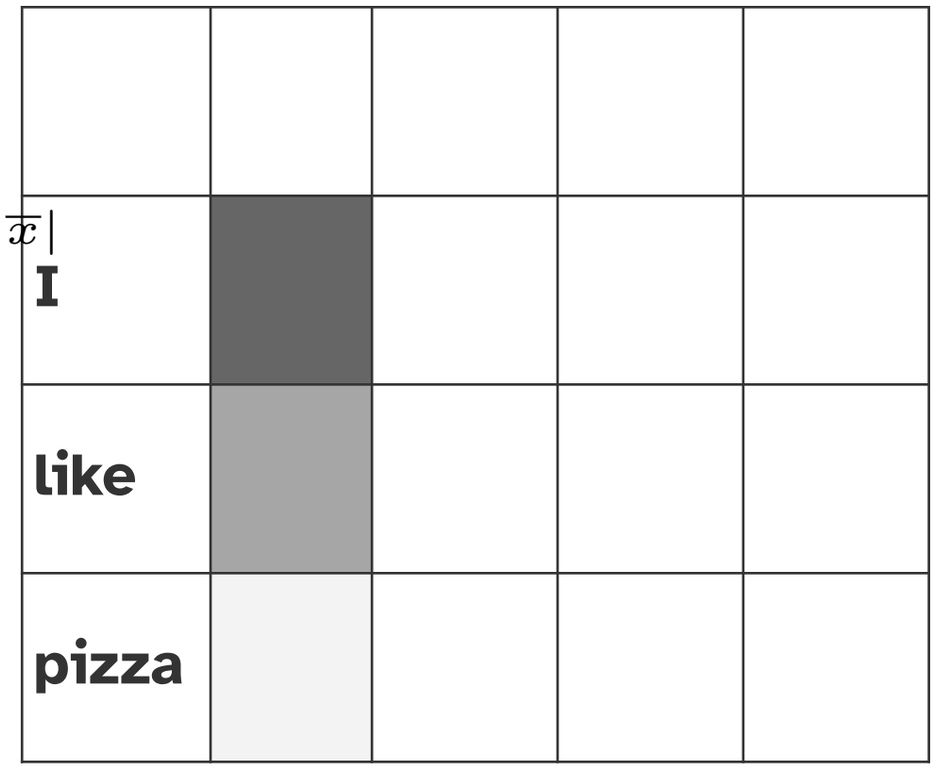
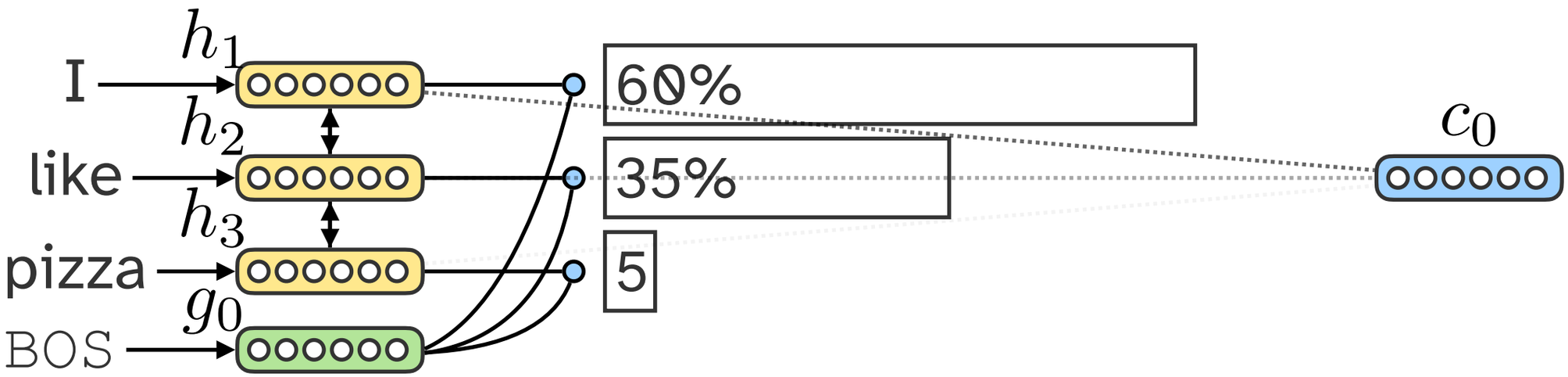


Illustration: Dot Product Attention



Similarities between query and keys

$$s_0 = g_0^\intercal \mathbf{h} \in \mathbb{R}^{|\overline{x}|}$$

Weights over keys (weights sum to 1)

$$\alpha_0 = p(\cdot \mid g_0, \mathbf{h}) = \mathrm{softmax}(s_0) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$$

Encoding is weighted sum over values

$$c_0 = \mathrm{Enc}(\overline{x}, g_0) = \sum_{i=1}^{|\overline{x}|} \alpha_{0,i} h_i \in \mathbb{R}^d$$

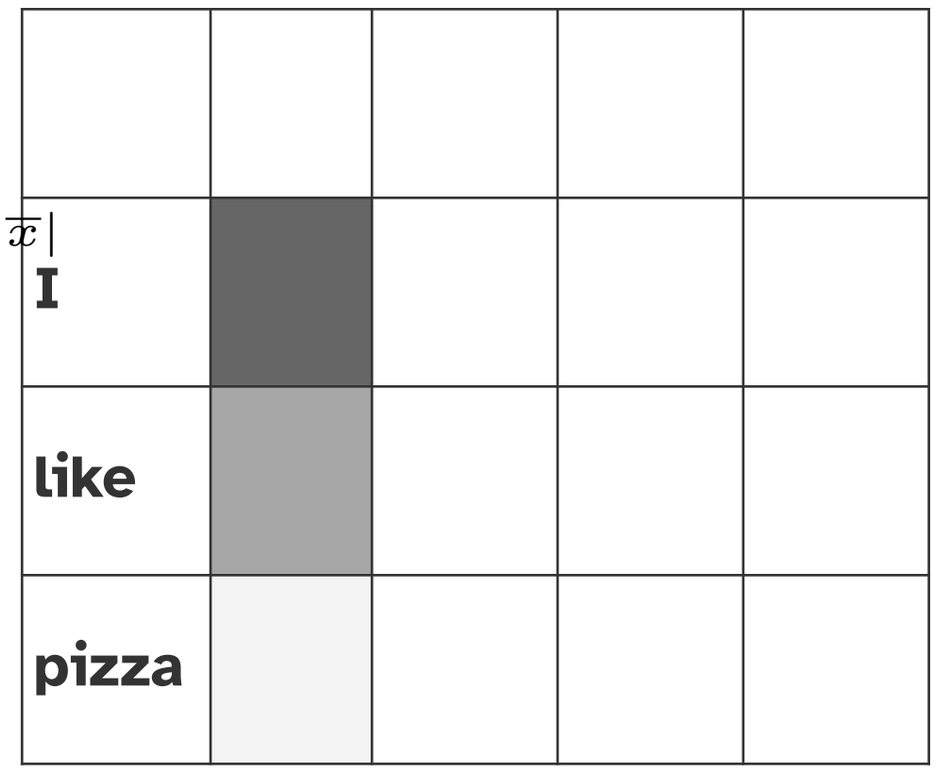
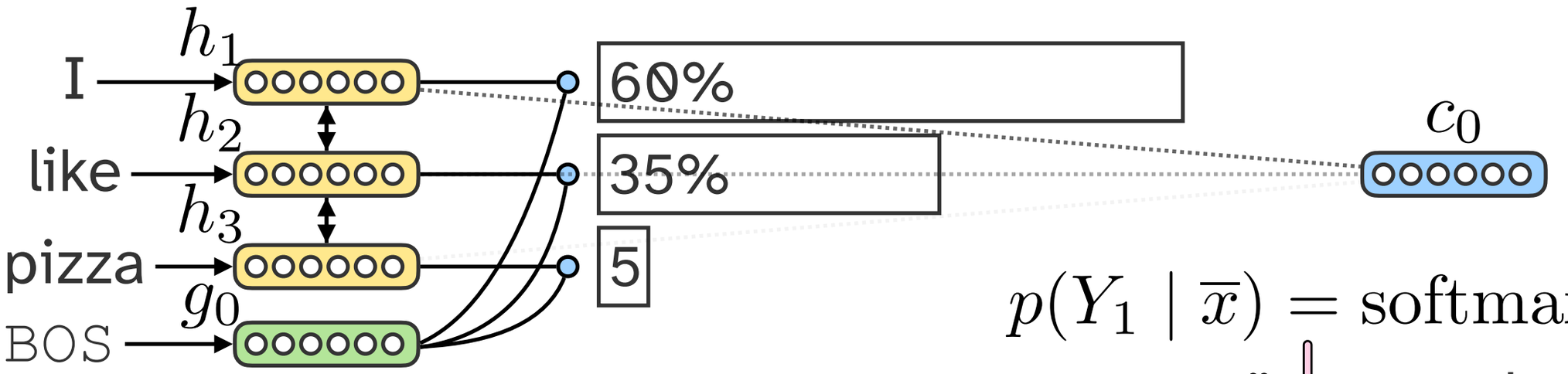
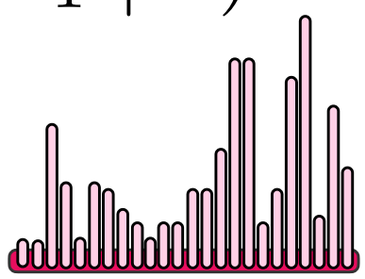


Illustration:
Dot Product Attention

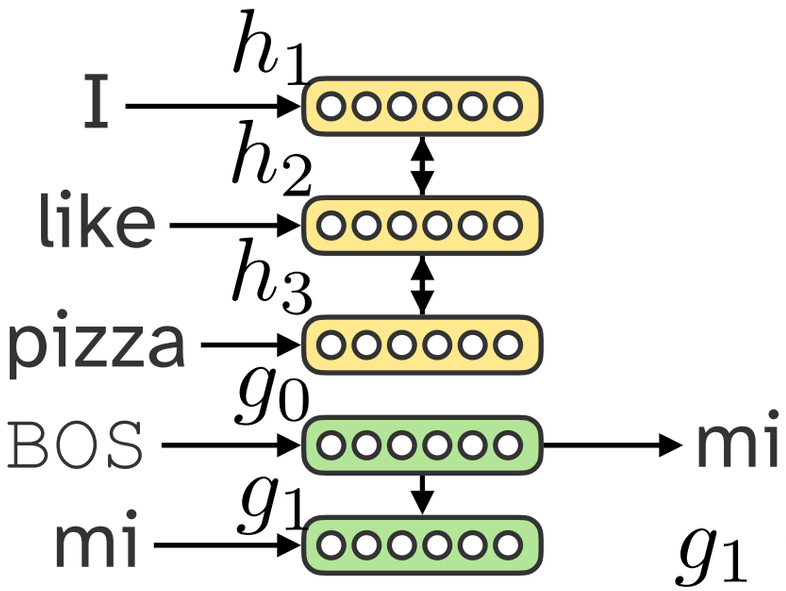


$$p(Y_1 \mid \overline{x}) = \text{softmax}(f(c_0, g_0))$$



	mi			
I				
like				
pizza				

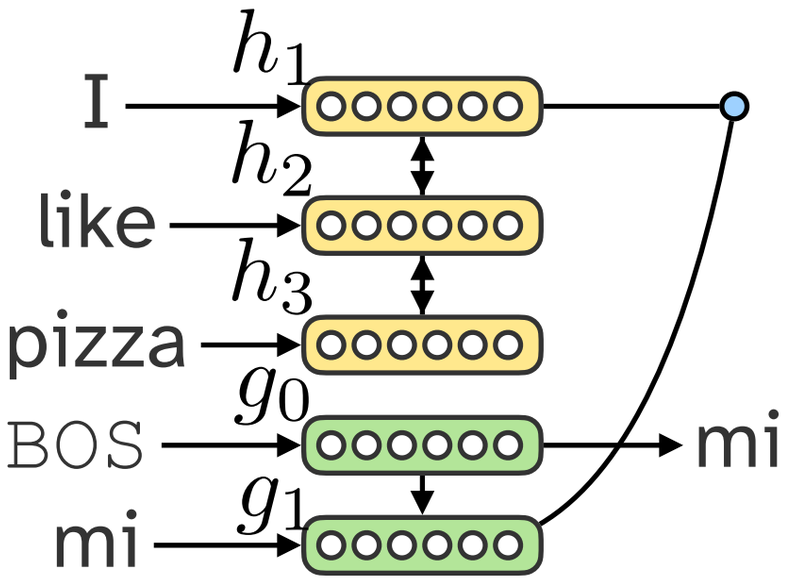
Illustration:
Dot Product Attention



$$g_1 = g(g_0, c_0, y_1)$$

	mi			
I				
like				
pizza				

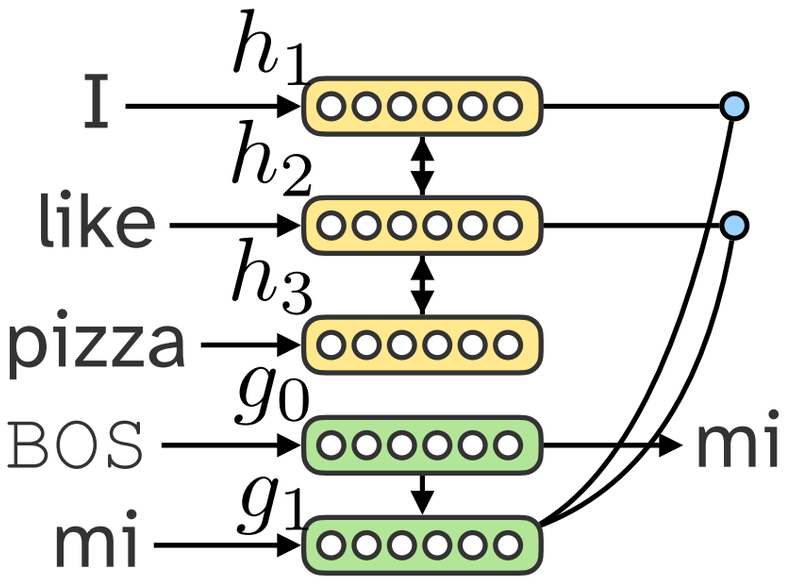
Illustration:
Dot Product Attention



$$s_{1,1} = g_1^\intercal h_1$$

	mi			
I				
like				
pizza				

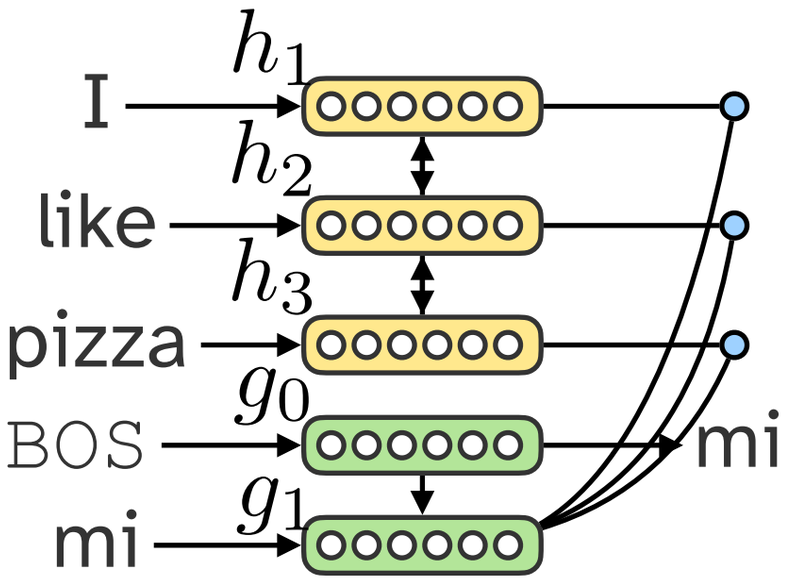
Illustration:
Dot Product Attention



$$s_{1,2} = g_1^\intercal h_2$$

	mi			
I				
like				
pizza				

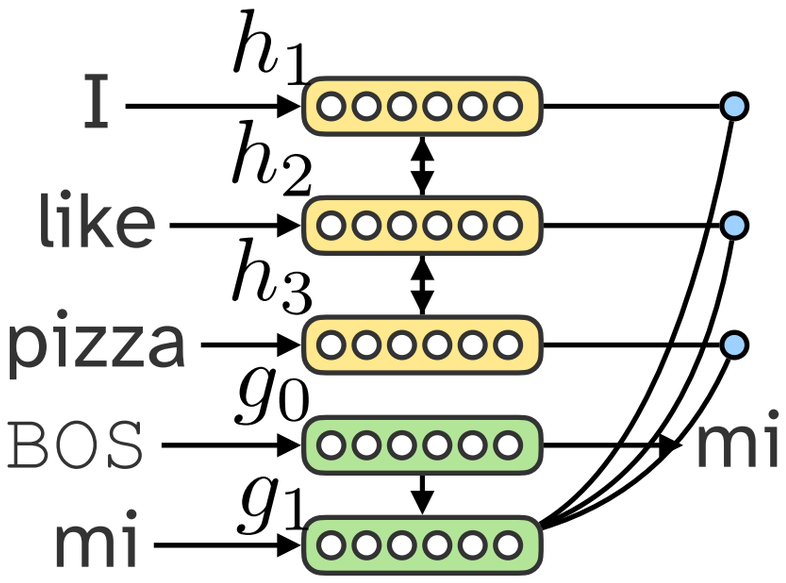
Illustration:
Dot Product Attention



$$s_{1,3} = g_1^\intercal h_3$$

	mi			
I				
like				
pizza				

Illustration:
Dot Product Attention

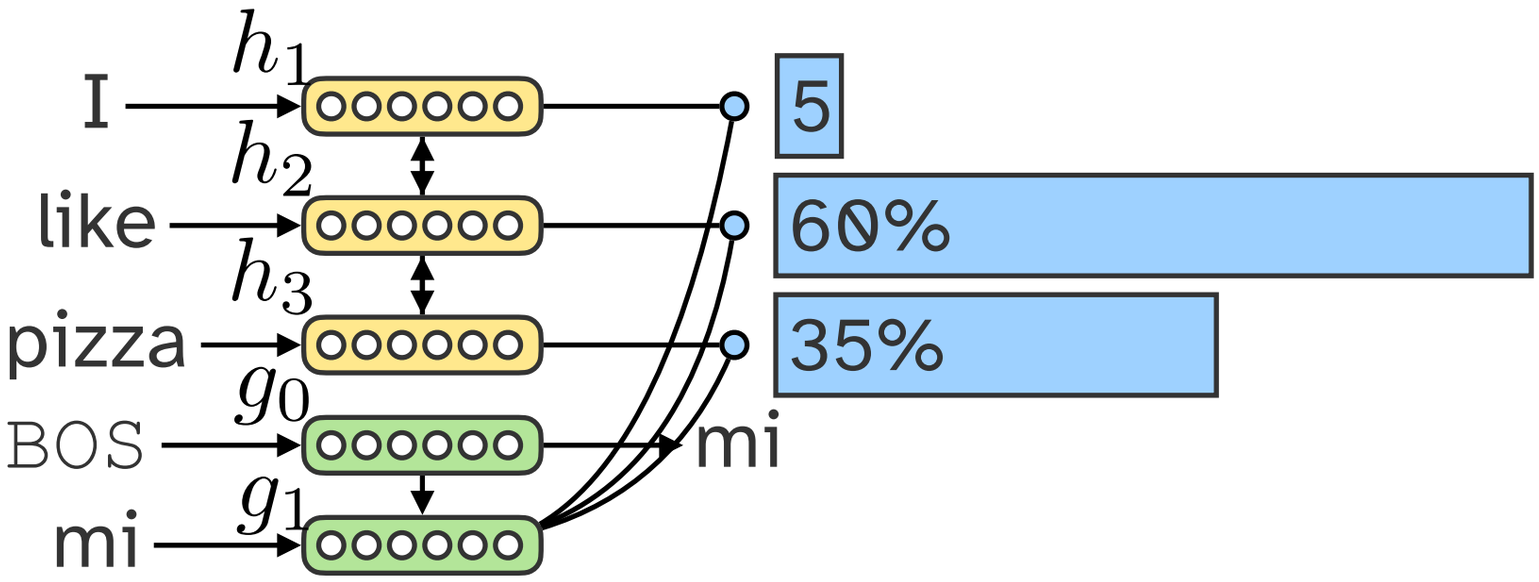


Similarities between query and keys

$s_1 = g_1^\intercal \mathbf{h} \in \mathbb{R}^{|\overline{x}|}$

	mi			
I				
like				
pizza				

Illustration:
Dot Product Attention



Similarities between query and keys

$$s_1 = g_1^\intercal \mathbf{h} \in \mathbb{R}^{|\overline{x}|}$$

Weights over keys (weights sum to 1)

$$\alpha_1 = p(\cdot \mid g_1, \mathbf{h}) = \mathrm{softmax}(s_1) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$$

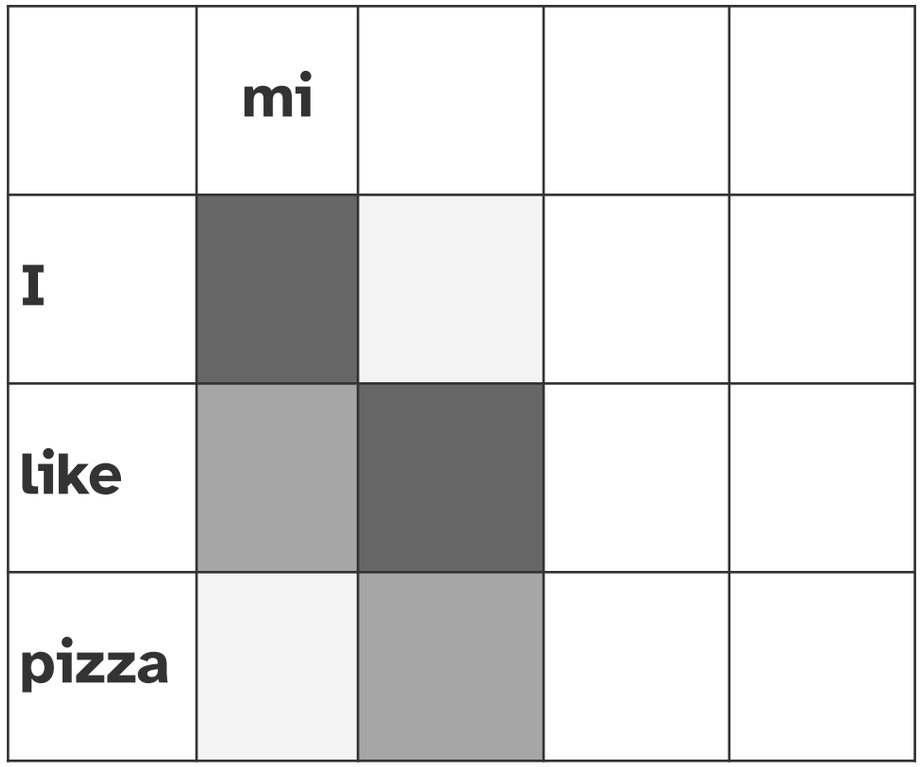
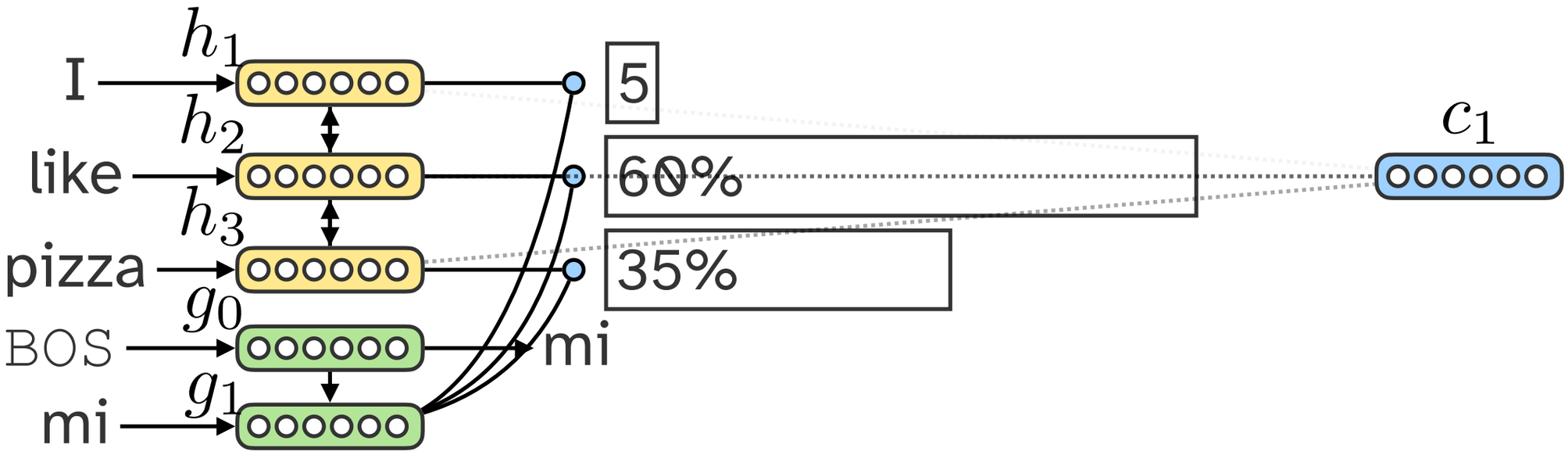


Illustration: Dot Product Attention



Similarities between query and keys

$$s_1 = g_1^\intercal \mathbf{h} \in \mathbb{R}^{|\overline{x}|}$$

Weights over keys (weights sum to 1)

$$\alpha_1 = p(\cdot \mid g_1, \mathbf{h}) = \mathrm{softmax}(s_1) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$$

Encoding is weighted sum over values

$$c_1 = \mathrm{Enc}(\overline{x}, g_1) = \sum_{i=1}^{|\overline{x}|} \alpha_{1,i} h_i \in \mathbb{R}^d$$

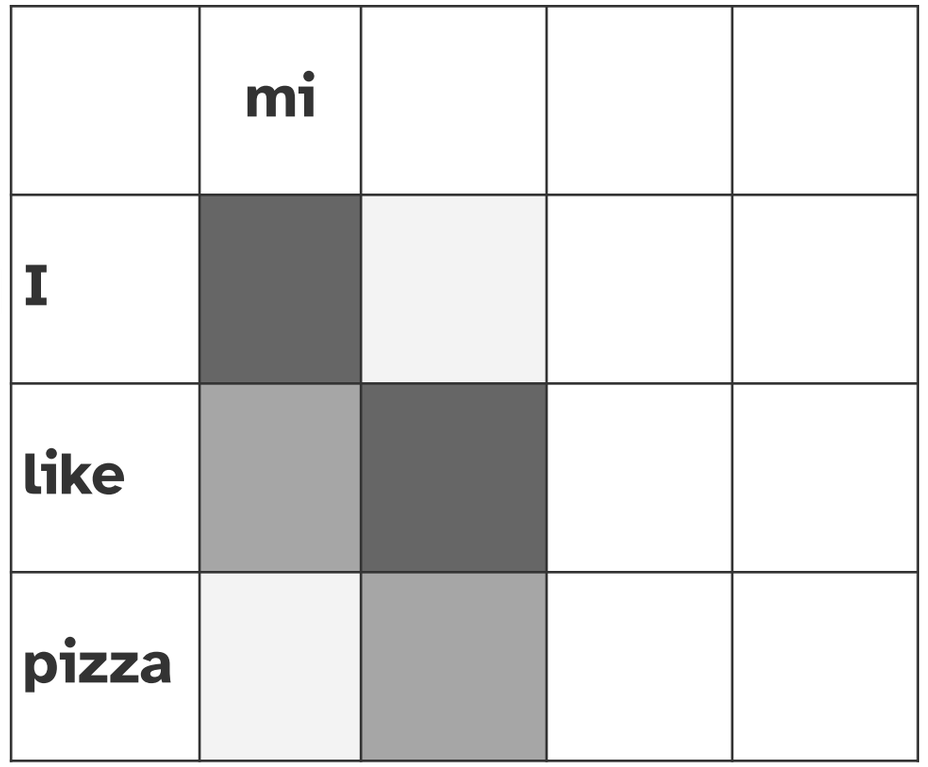
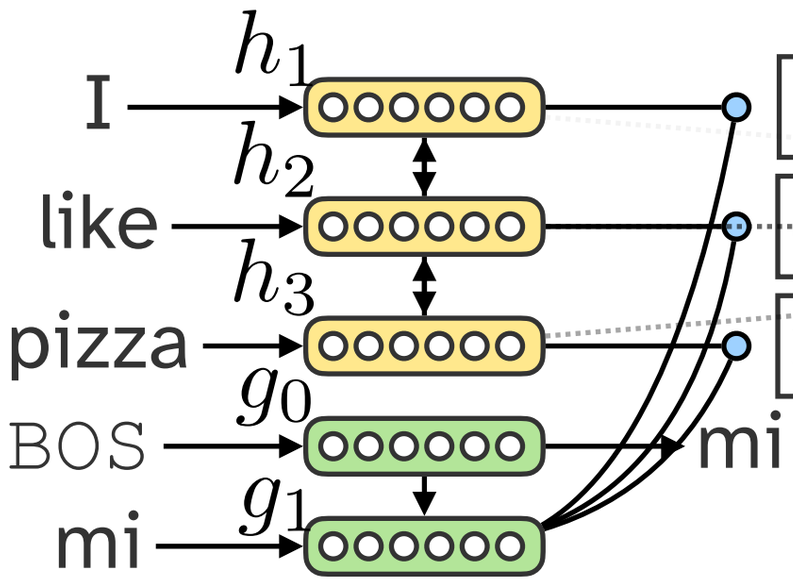
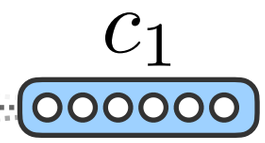
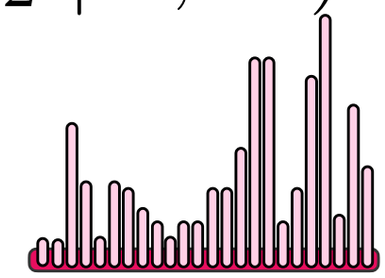


Illustration:
Dot Product Attention





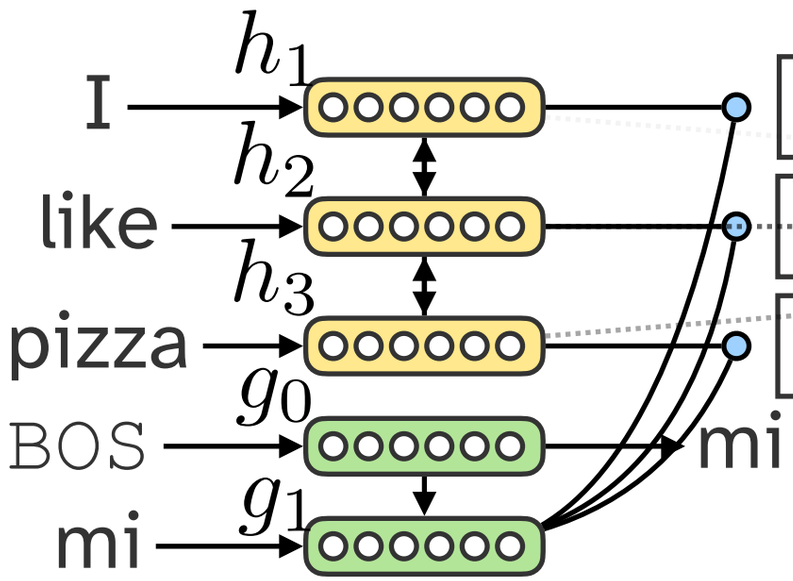
$$p(Y_2 \mid \overline{x}, \text{mi}) = \text{softmax}(f(c_1, g_1))$$

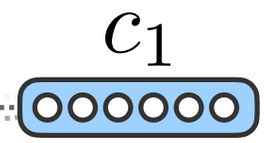


sample → piace

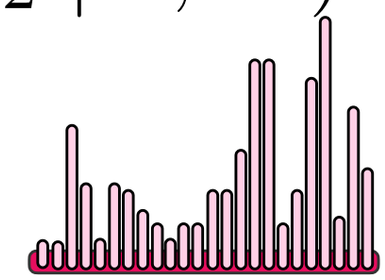
	mi			
I				
like				
pizza				

Illustration:
Dot Product Attention





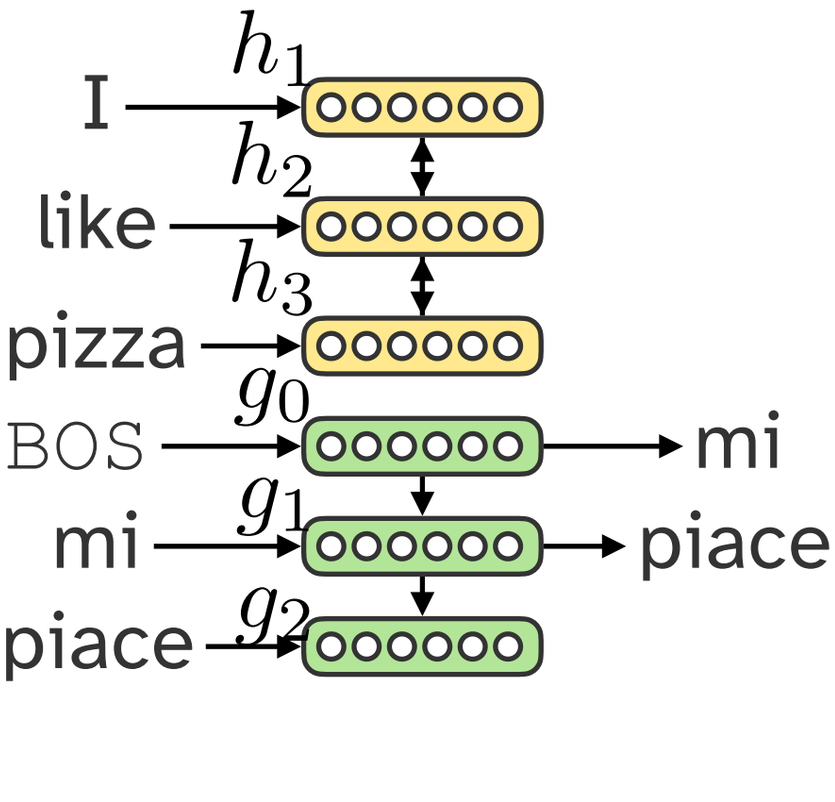
$$p(Y_2 \mid \overline{x}, \text{mi}) = \text{softmax}(f(c_1, g_1))$$



sample → piace

	mi	piace		
I				
like				
pizza				

Illustration:
Dot Product Attention

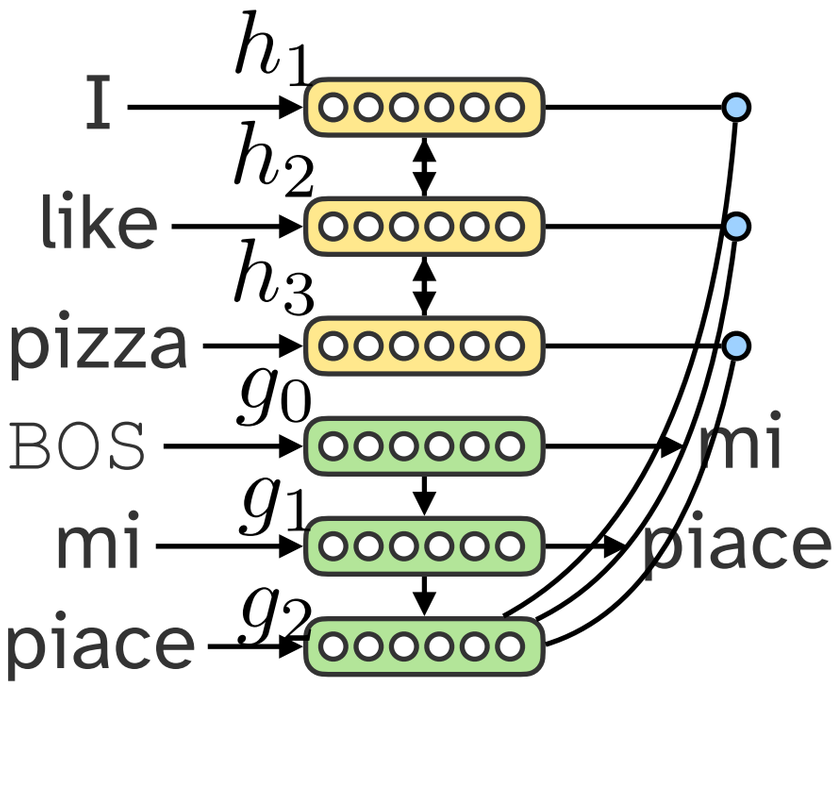


General hidden state update rule:

$$g_i = g(g_{i-1}, c_{i-1}, y_i)$$

	mi	piace		
I				
like				
pizza				

Illustration:
Dot Product Attention



General scoring function:

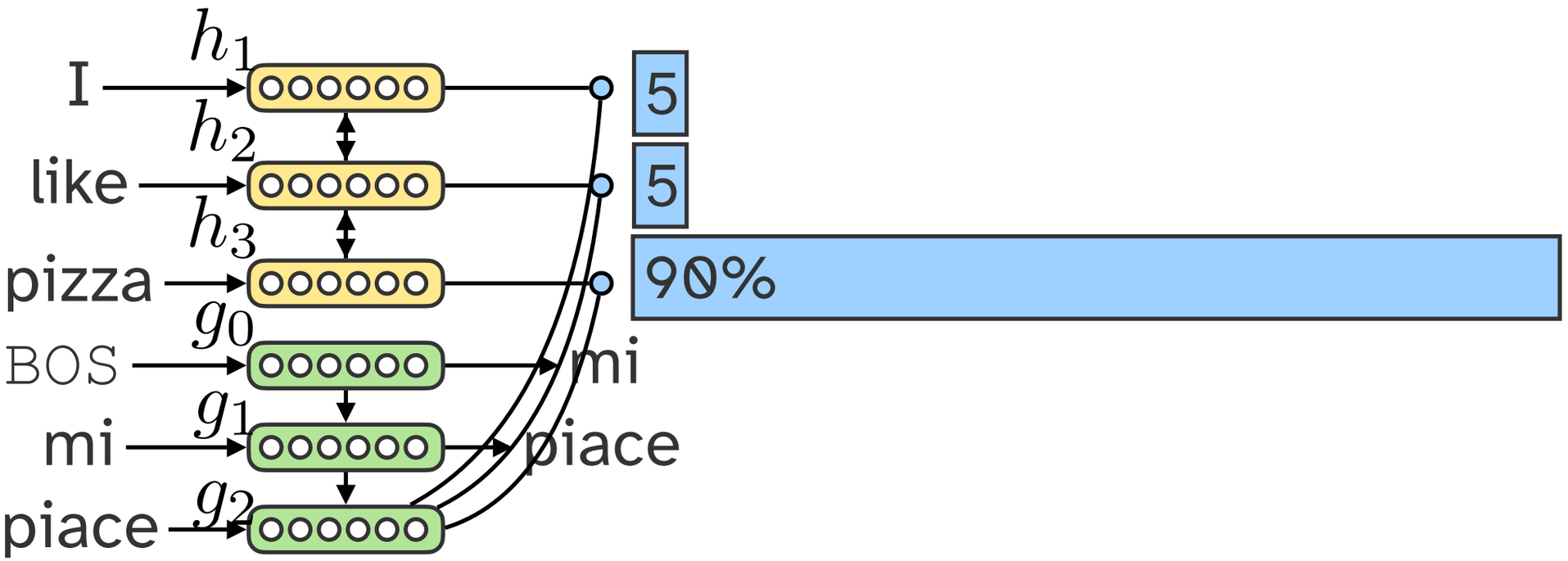
$$s_i = a(g_i, \mathbf{h}) \in \mathbb{R}^{|\overline{x}|}$$

E.g., dot product attention:

$$a(g_i, \mathbf{h}) = g_i^\intercal \mathbf{h}$$

	mi	piace		
I				
like				
pizza				

Illustration:
Dot Product Attention



Attention weights:

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

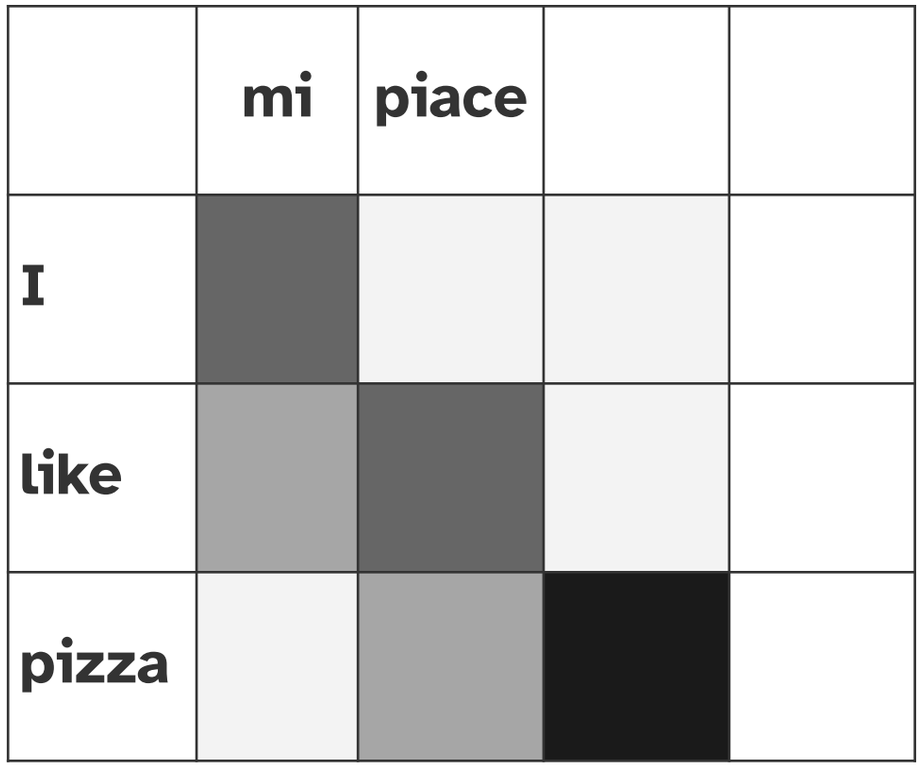
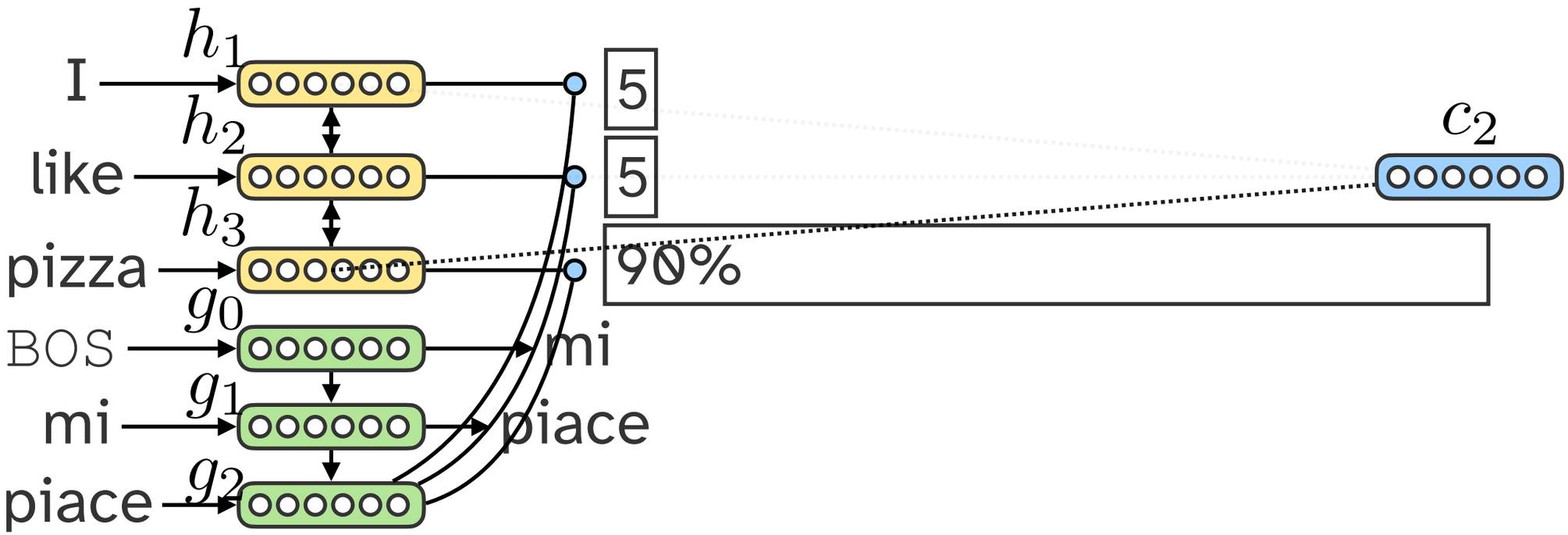


Illustration: Dot Product Attention

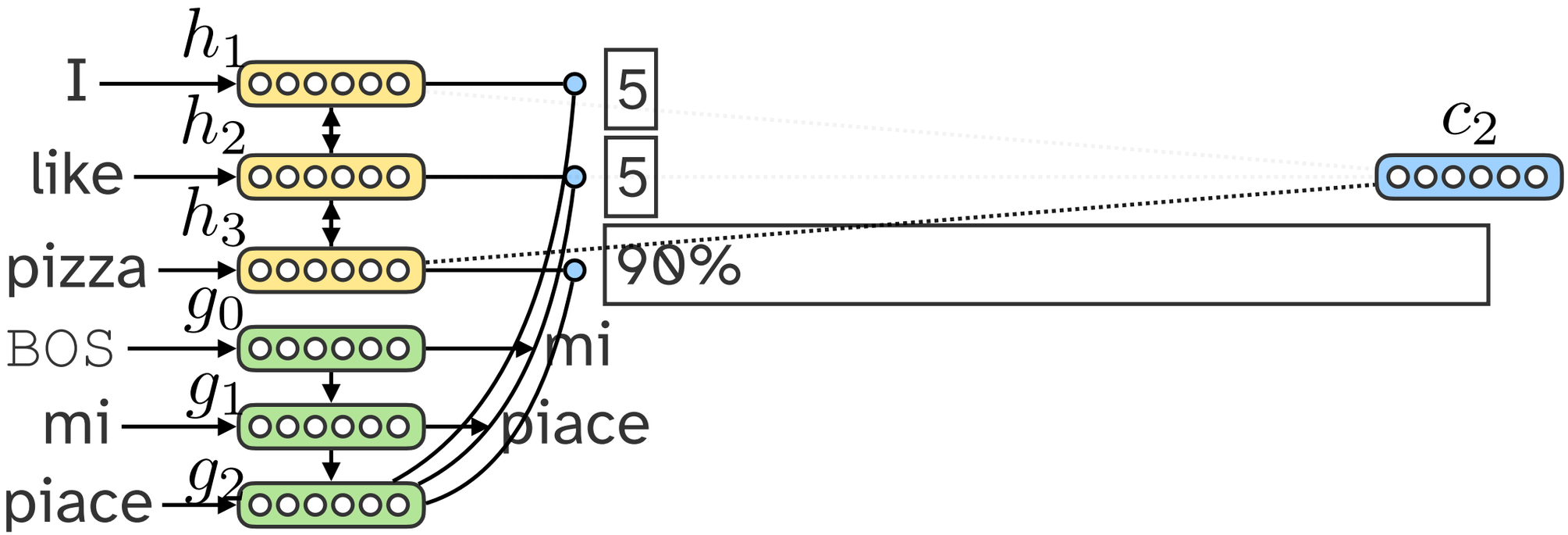


Weighted sum of values:

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

	mi	piace		
I				
like				
pizza				

Illustration: Dot Product Attention



General function for distribution over next token:

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

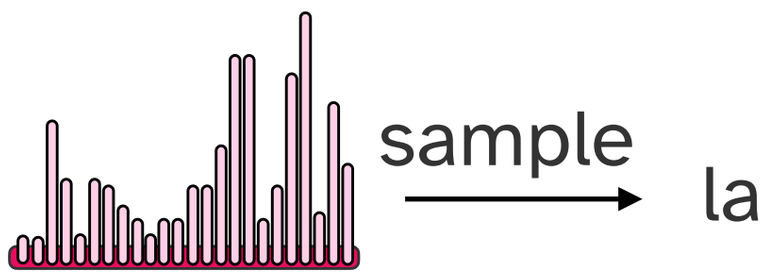
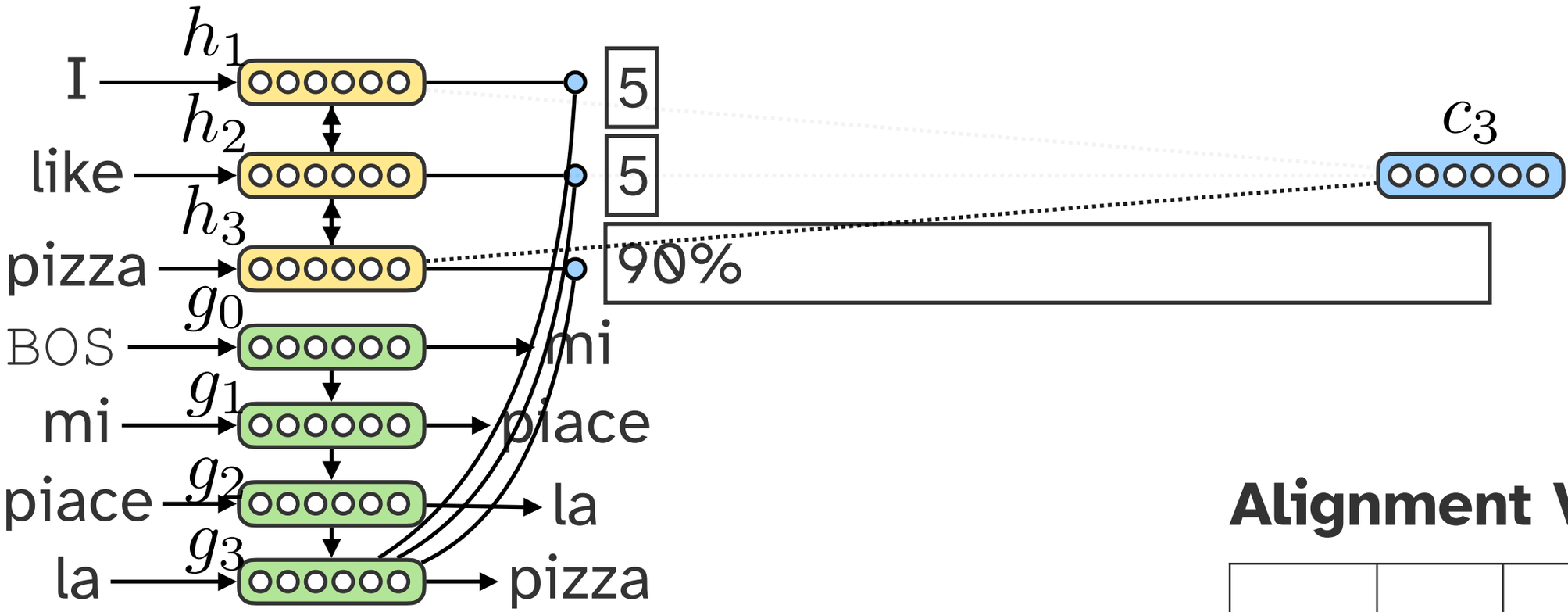


Illustration:
Dot Product Attention



Alignment Visualization

	mi	piace	la	pizza
I				
like				
pizza				

One Last Overview

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

One Last Overview

- 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}

One Last Overview

- 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

One Last Overview

- 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 ($\mathbf{h}$): $s_i = a(g_i, \mathbf{h}) \in \mathbb{R}^{|\overline{x}|}$

One Last Overview

- 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:

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

One Last Overview

- 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

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

One Last Overview

- 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:

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

One Last Overview

- 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: $g_i = g(g_{i-1}, c_{i-1}, y_i)$

Complexity

$$h_1, \ldots, h_{|\overline{x}|} = \mathbf{h} \in \mathbb{R}^{d \times |\overline{x}|}$$

O(n) $$s_i = a(g_i, \mathbf{h}) \in \mathbb{R}^{|\overline{x}|}$$

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

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

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

$$g_i = g(g_{i-1}, c_{i-1}, y_i)$$

O(n m)

Back to Problem 2

- **Main problem: fixed-size representation (“bottleneck”) of input sequence at each stage of decoding**
- Attention fixes this by allowing access to different parts of the input sequence based on where we are in decoding
- While decoding token-by-token, we can “focus” on different input parts
- Looks like alignment in MT!

	mi	piace	la	pizza
I				
like				
pizza				

	macja	ime	është	e	zezë
my					
cat					
is					
black					

Other Benefits

- Helps with vanishing gradient problem via shortcut to far-away states

- Provides some kind of interpretability (“soft” alignment) learned without any supervision

	mi	piace	la	pizza
I				
like				
pizza				

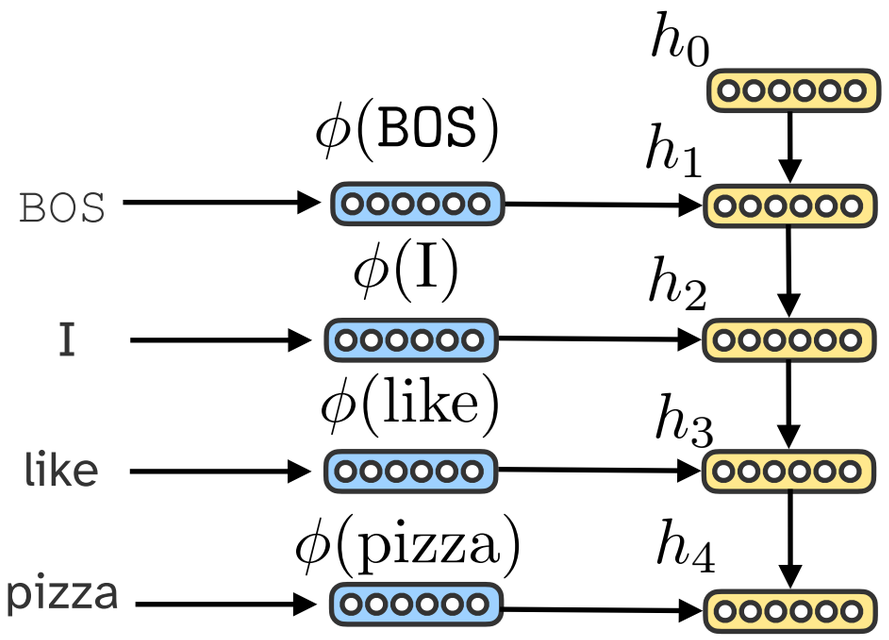
	macja	ime	është	e	zezë
my					
cat					
is					
black					

Self-Attention

- Attention was initially developed for sequence-to-sequence problems

- But we can also use it during single-sequence generation to solve analogous problems:
  - Problem 1: long-distance dependencies
  - Problem 2: fixed-size representation of sequence so far

When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her …



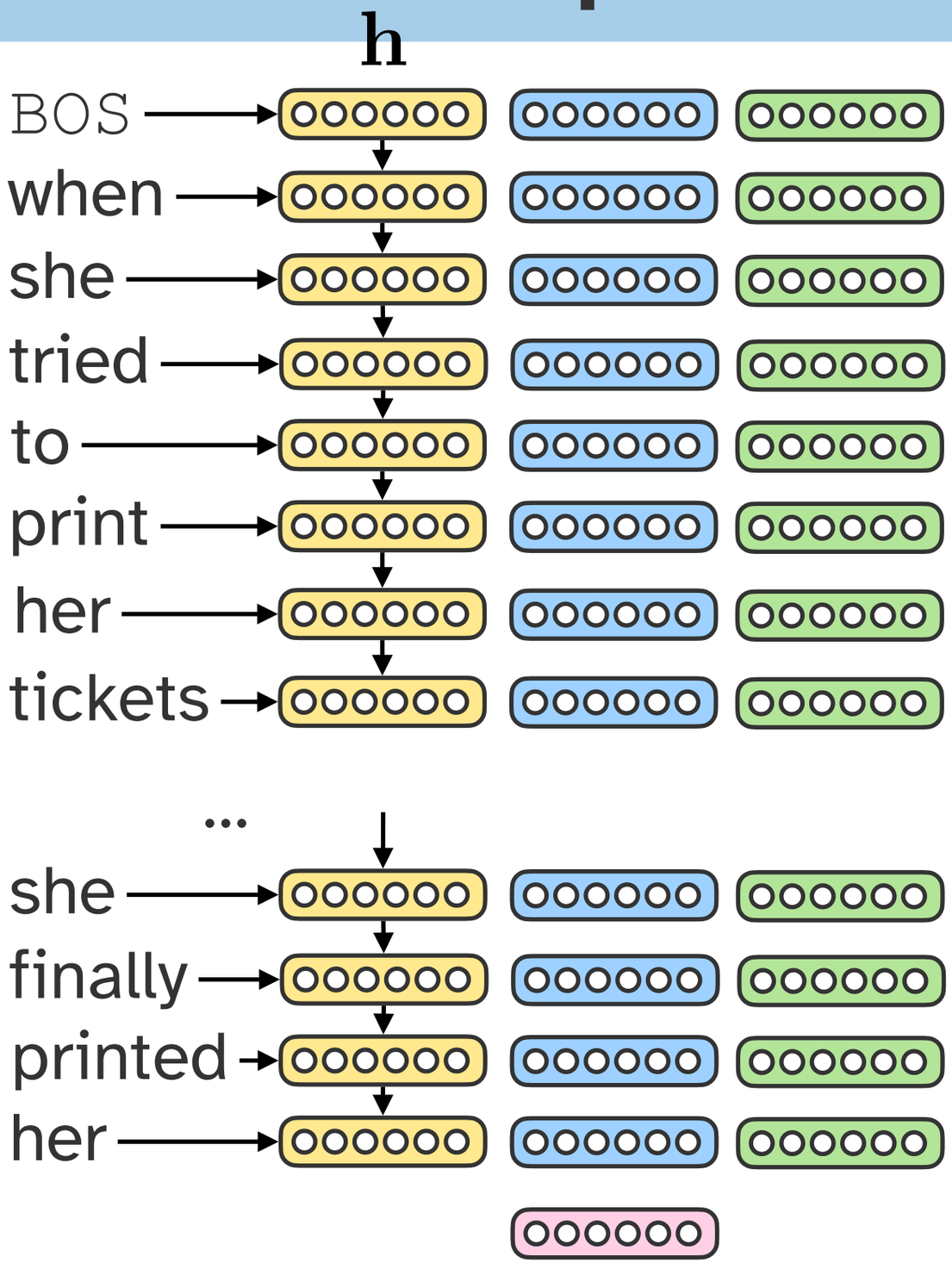
Self-Attention

- Keys, values, and queries are transformations of the same underlying representations

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

- When trying to generate our next token at index *j*, we need a query: $q_j = Qh_j \in \mathbb{R}^d \qquad Q \in \mathbb{R}^{d \times d}$
  - 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: $c_j = \sum_{i=1}^{j} \alpha_{ji} v_i$

One Step of Self-Attention

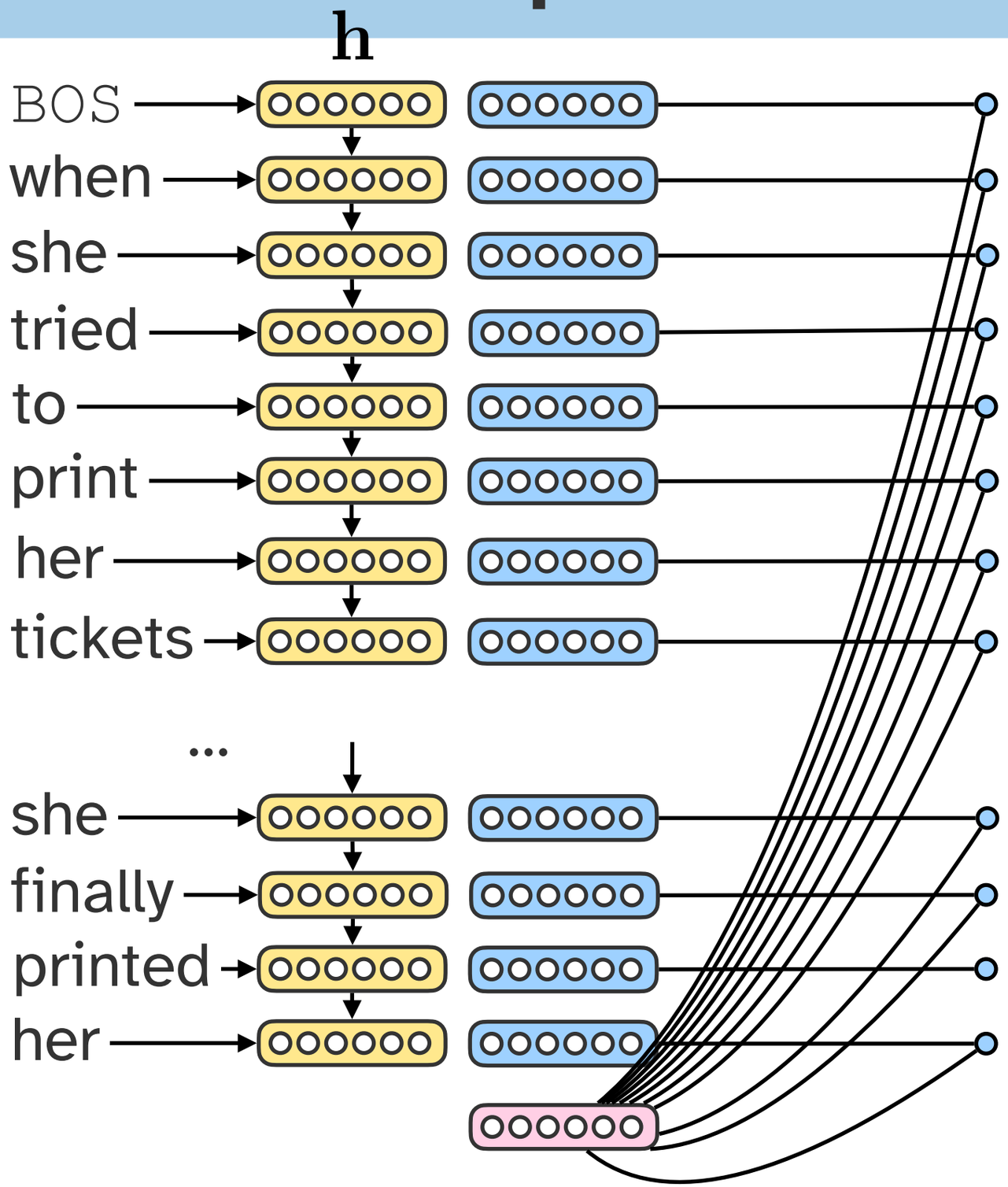


$$\mathbf{k} = K\mathbf{h} \in \mathbb{R}^{d \times j}$$

$$\mathbf{v} = V\mathbf{h} \in \mathbb{R}^{d \times j}$$

$$q_j = Qh_j \in \mathbb{R}^{d}$$

One Step of Self-Attention

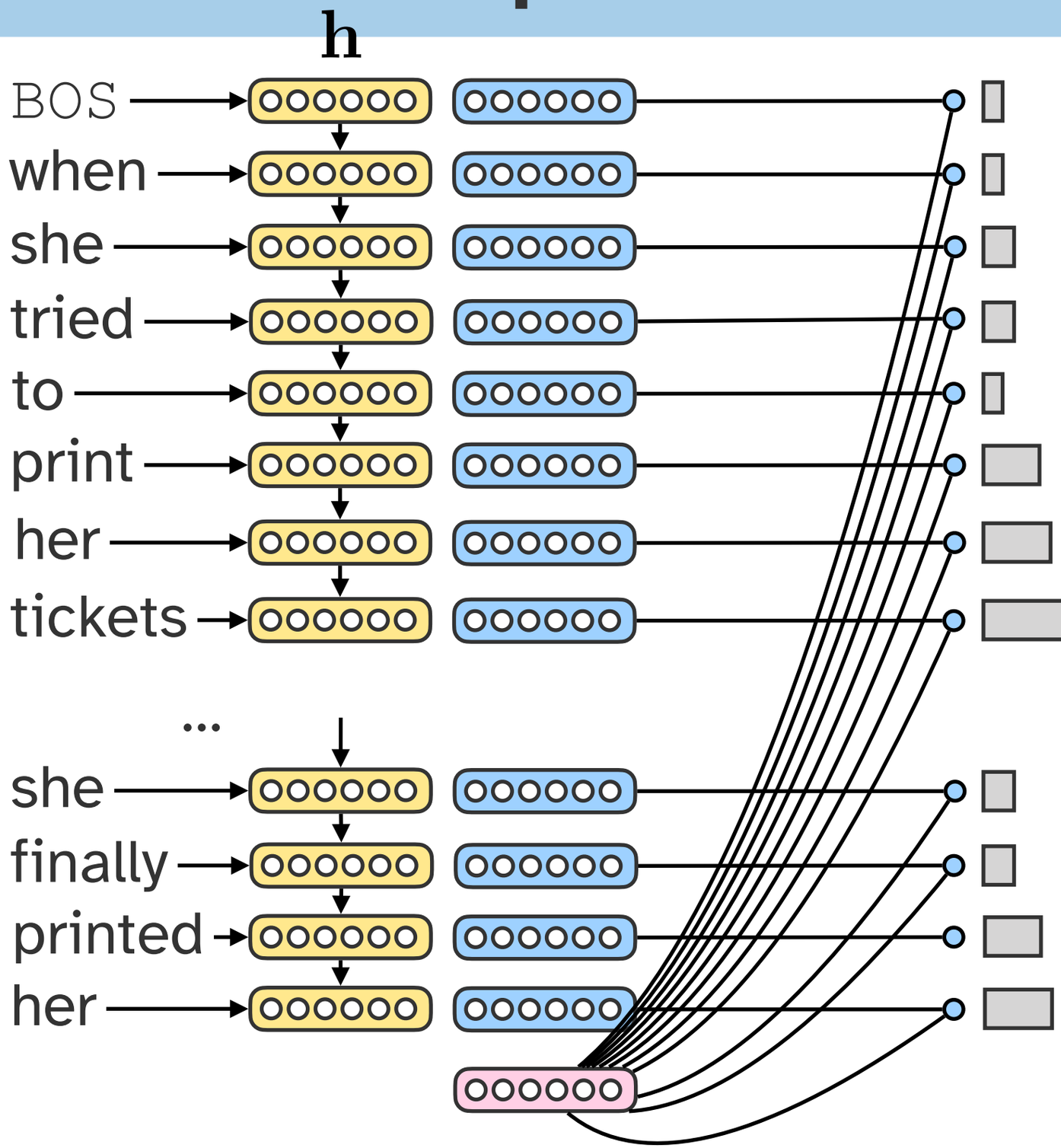


$$\mathbf{k} = K\mathbf{h} \in \mathbb{R}^{d \times j}$$

$$q_j = Qh_j \in \mathbb{R}^d$$

$$\mathbf{s}_j = q_j^\intercal \mathbf{k} \in \mathbb{R}^j$$

One Step of Self-Attention



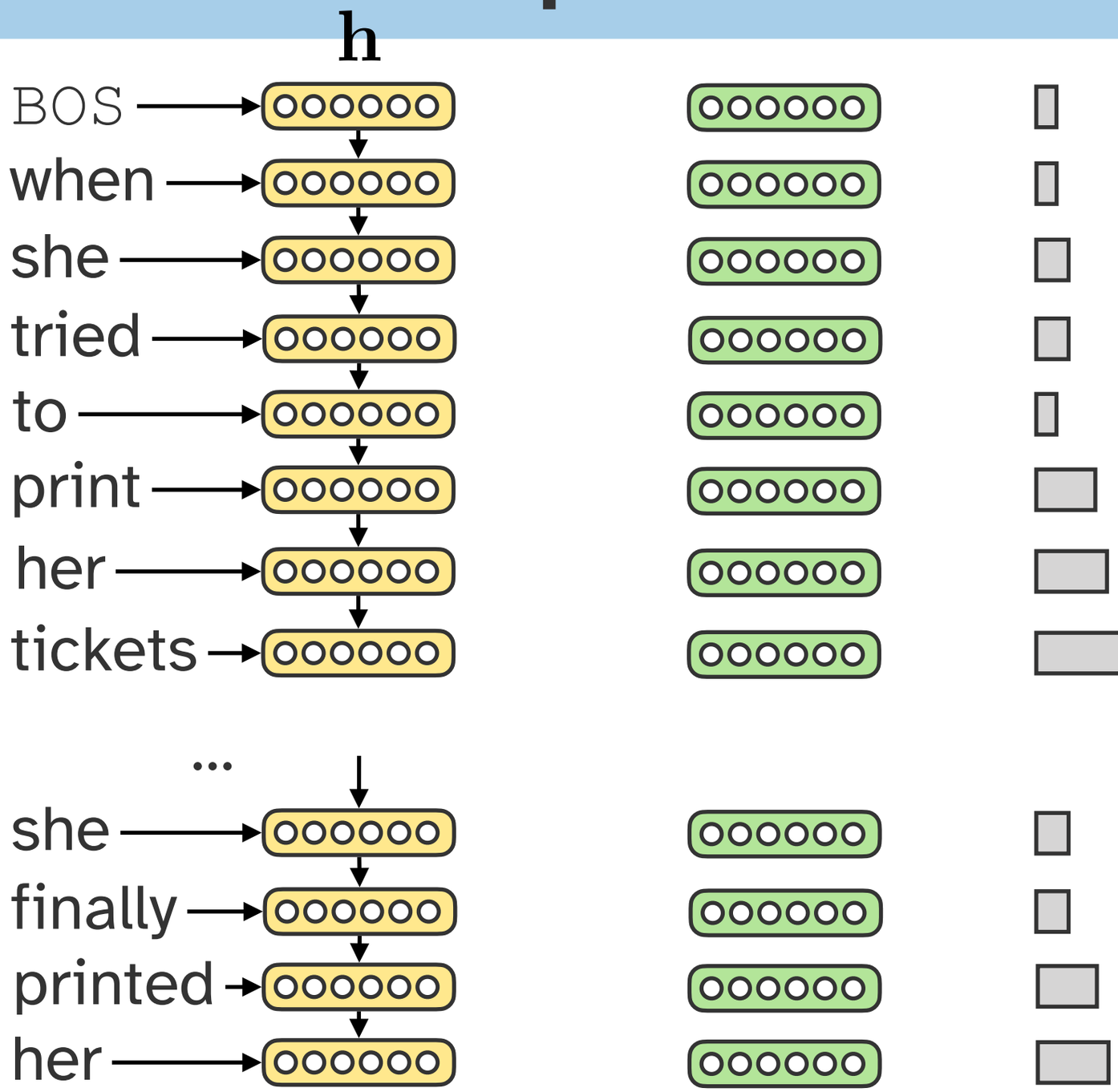
$$\mathbf{k} = K\mathbf{h} \in \mathbb{R}^{d \times j}$$

$$q_j = Qh_j \in \mathbb{R}^d$$

$$\mathbf{s}_j = q_j^\intercal \mathbf{k} \in \mathbb{R}^j$$

$$\alpha_j = \mathrm{softmax}(\mathbf{s}_j)$$

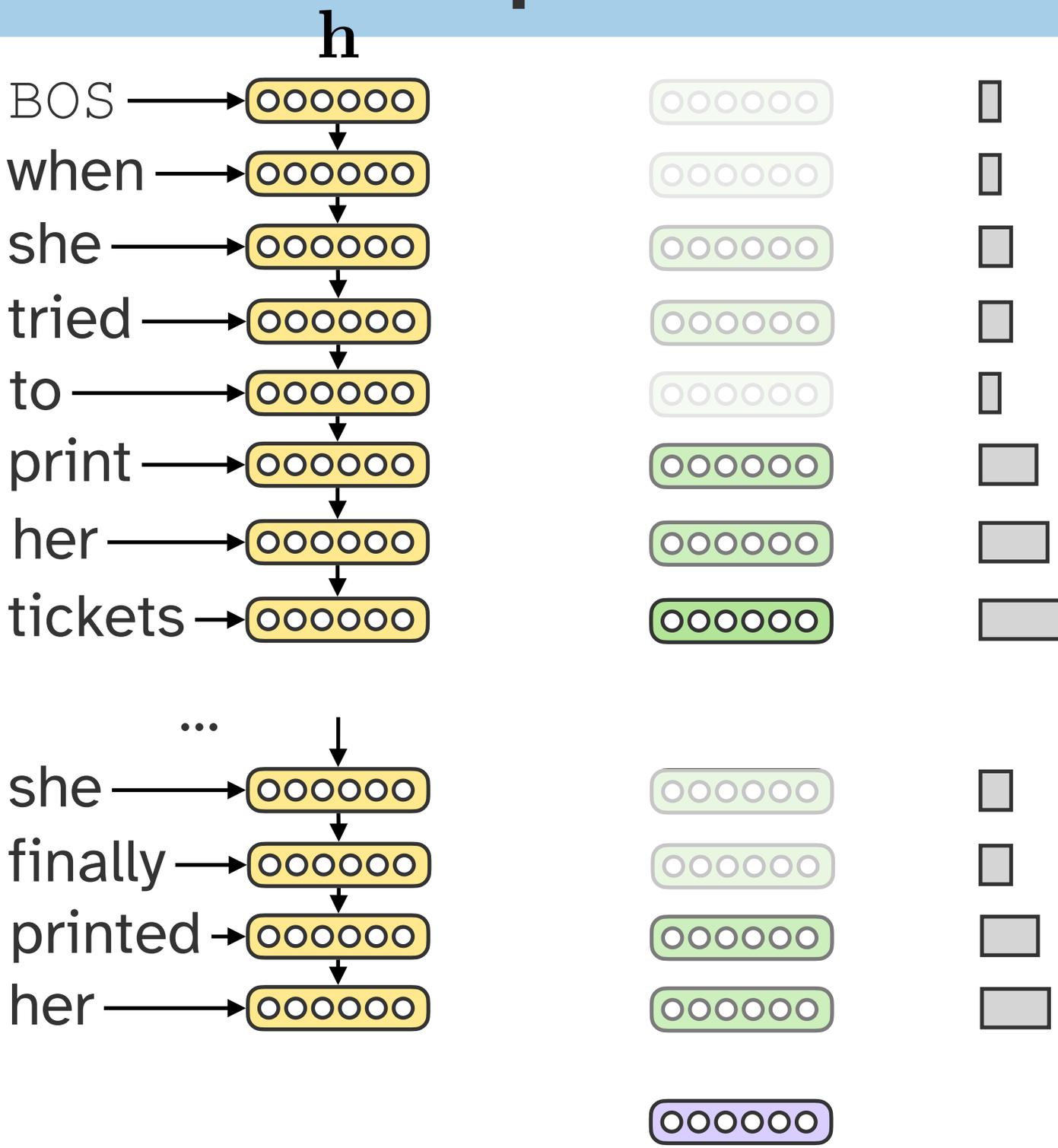
One Step of Self-Attention



$$\mathbf{v} = V\mathbf{h} \in \mathbb{R}^{d \times j}$$

$$\alpha_j = \mathrm{softmax}(\mathbf{s}_j)$$

One Step of Self-Attention

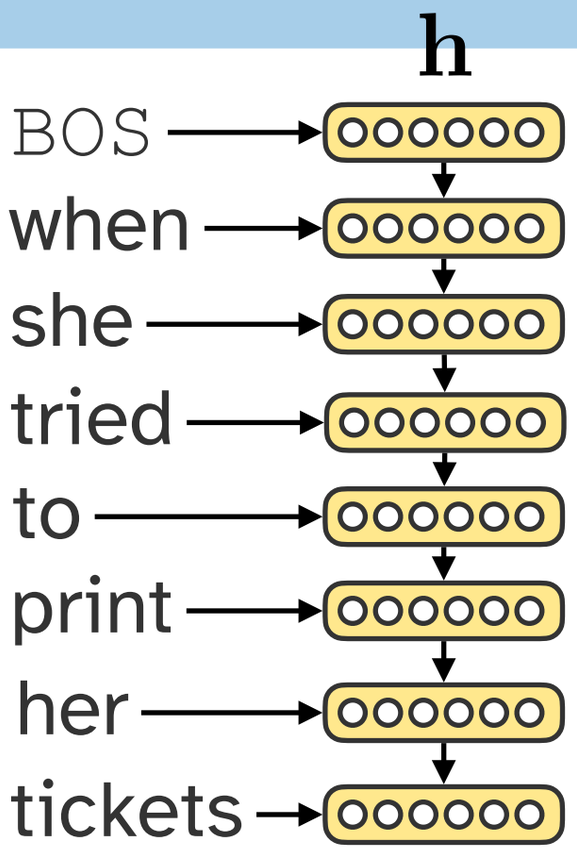


$$\mathbf{v} = V\mathbf{h} \in \mathbb{R}^{d \times j}$$

$$\alpha_j = \text{softmax}(\mathbf{s}_j)$$

$$c_j = \sum_{i=1}^{j} \alpha_{ji} v_i$$

One Step of Self-Attention

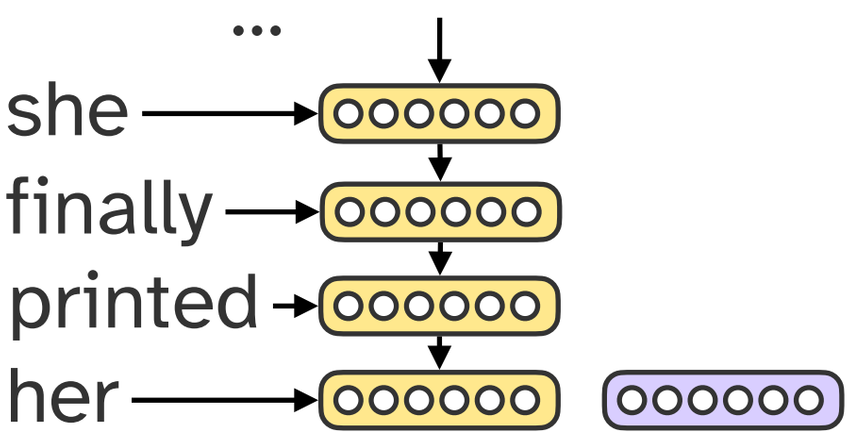


$$x_{j+1} \sim p(X_{j+1} \mid x_1, \ldots, x_j) = \text{softmax}(f(c_j, h_j))$$

tickets

$$h_{j+1} = g(h_j, c_j, x_{j+1})$$

where g is some RNN



$$c_j = \sum_{i=1}^{j} \alpha_{ji} v_i$$

Is Attention All You Need?

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

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

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

~~$h_{j+1} = g(h_j, c_j, x_{j+1})$~~ $\qquad h_{j+1} = \phi(x_{j+1})$

Is Attention All You Need?

- 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: $v_i = V\phi(x_i) \in \mathbb{R}^d \quad V \in \mathbb{R}^{d \times d}$

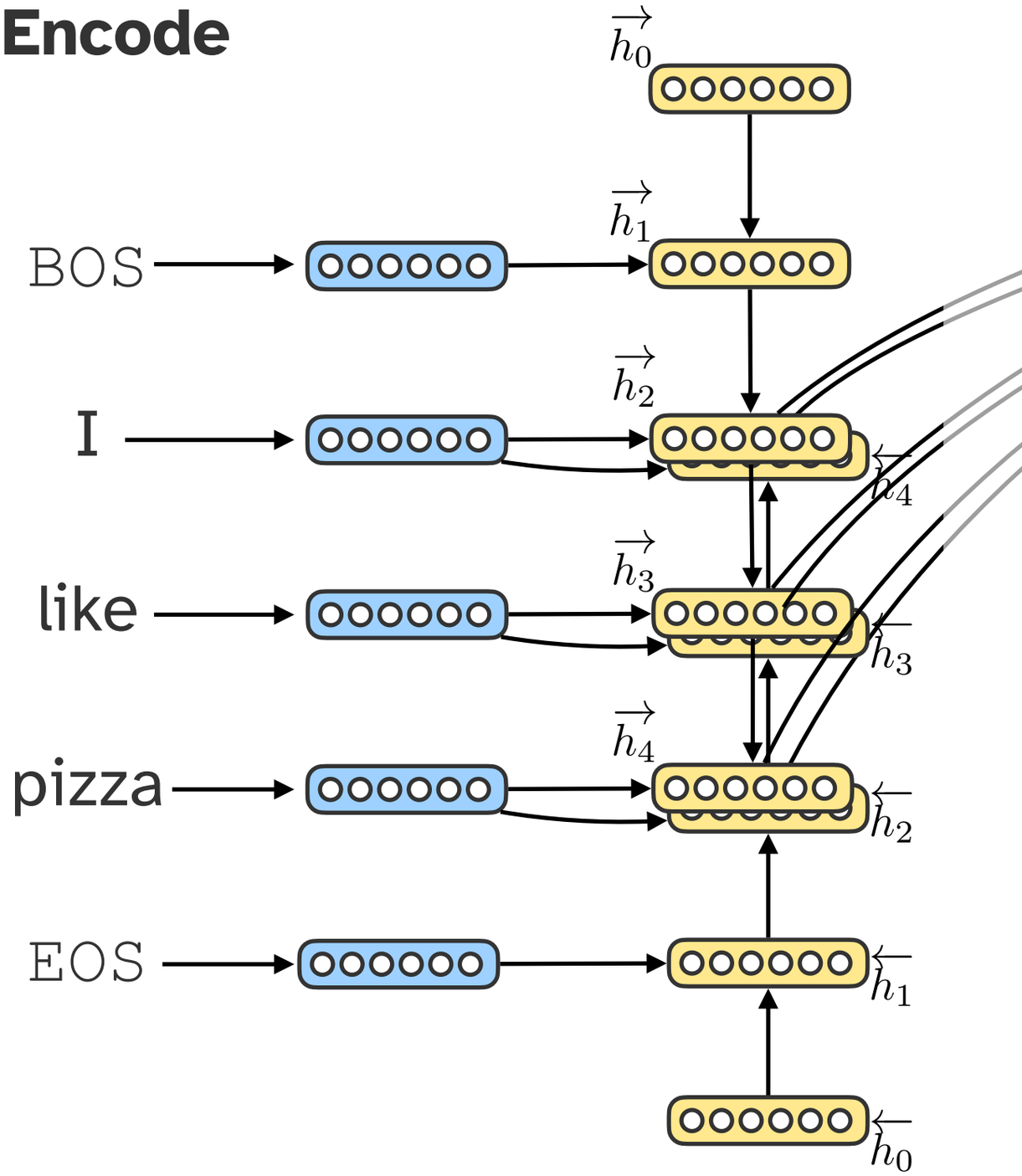
- When trying to generate our next token at index *j*, we need a query: $q_j = Q\phi(x_j) \in \mathbb{R}^d \;\; Q \in \mathbb{R}^{d \times d}$
  - Scores over keys: $s_{ji} = q_j^\intercal k_i$
  - Attention distribution over keys: $\alpha_{ji} = \dfrac{\exp(s_{ji})}{\sum_{i'=1}^{j} \exp(s_{ji'})}$
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

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

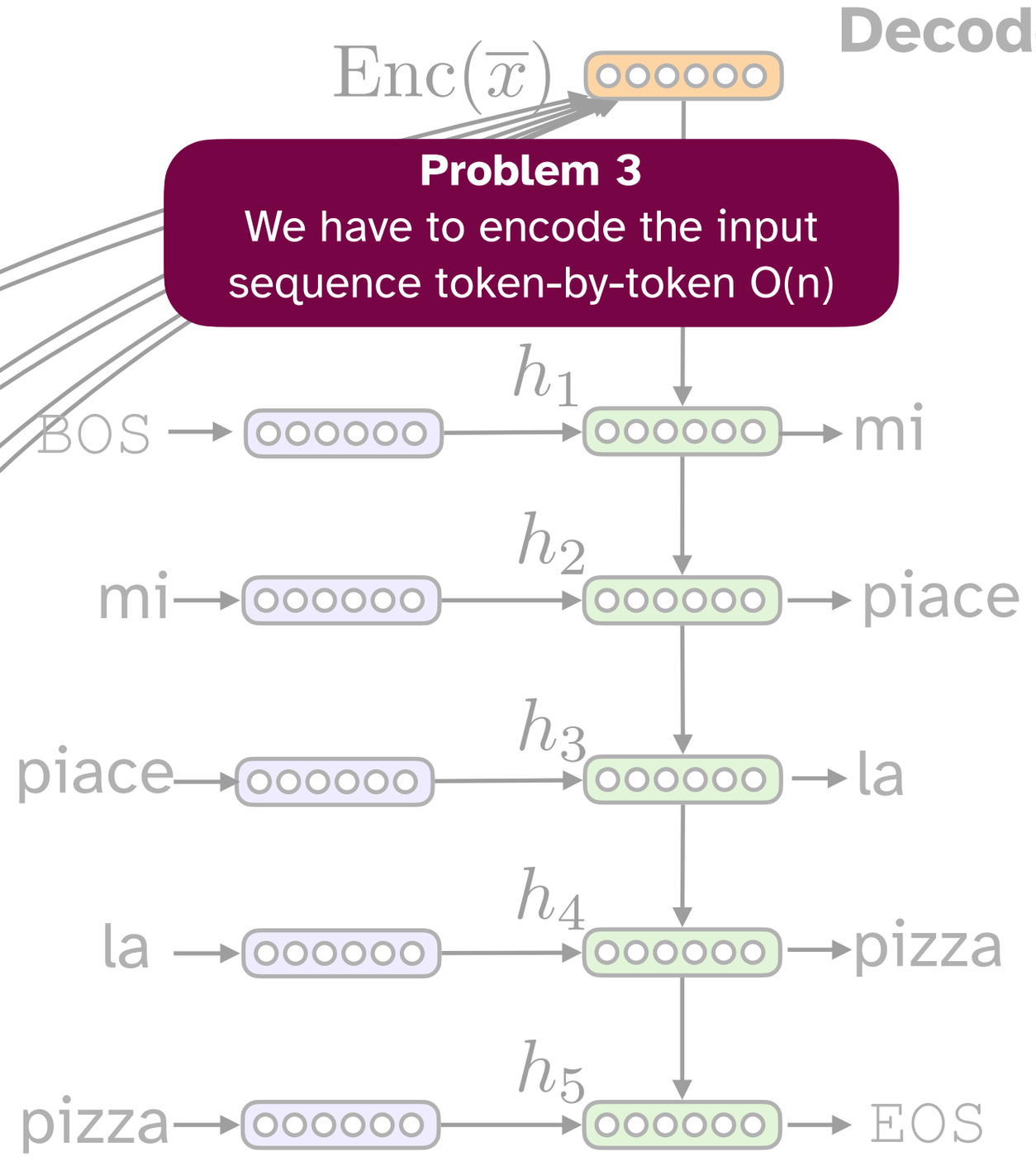
~~$h_{j+1} = g(h_j, c_j, x_{j+1})$~~ $\quad h_{j+1} = \phi(x_{j+1})$

Bidirectional Encoders

Encode



Decode

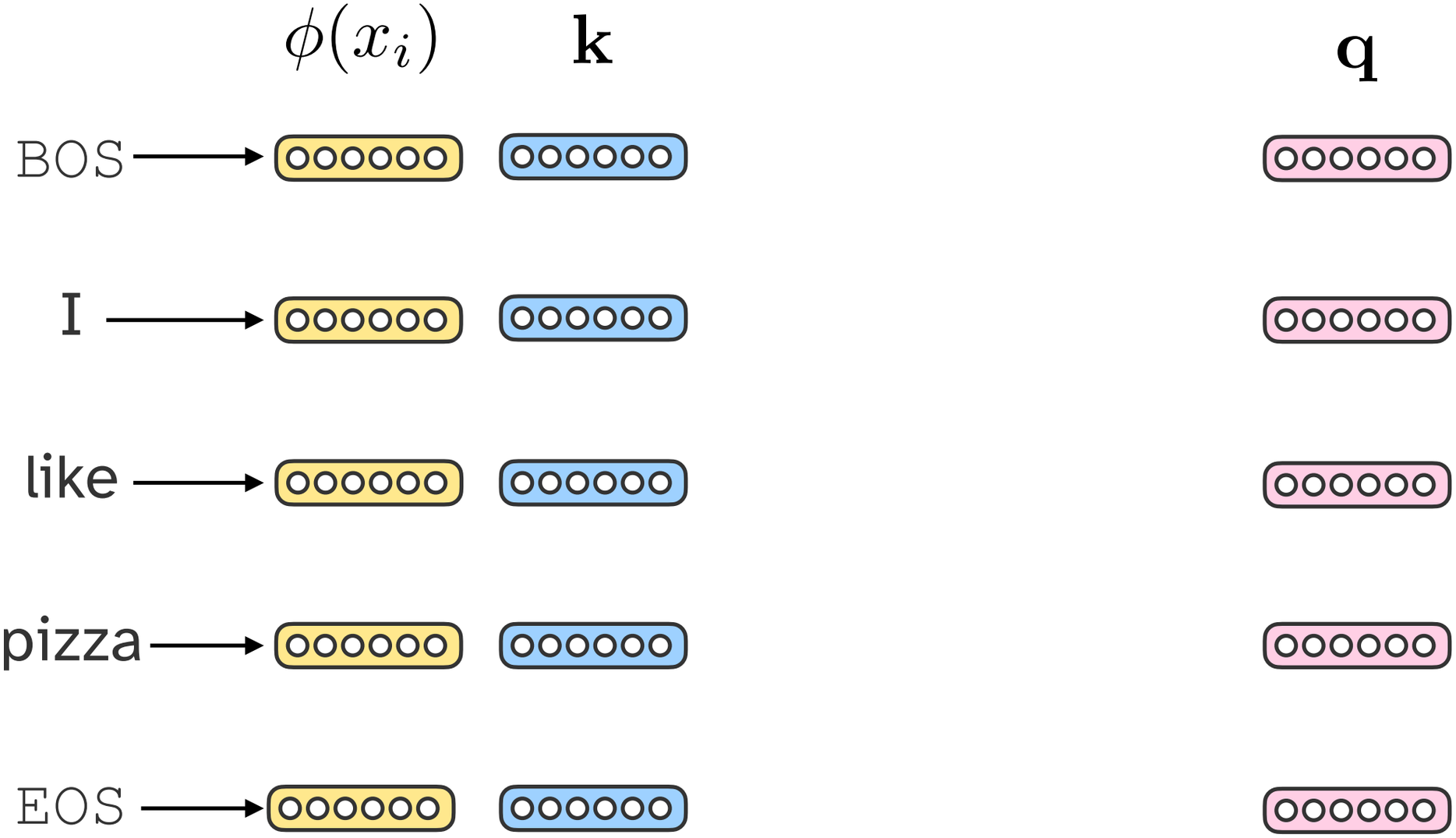


Problem 3

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

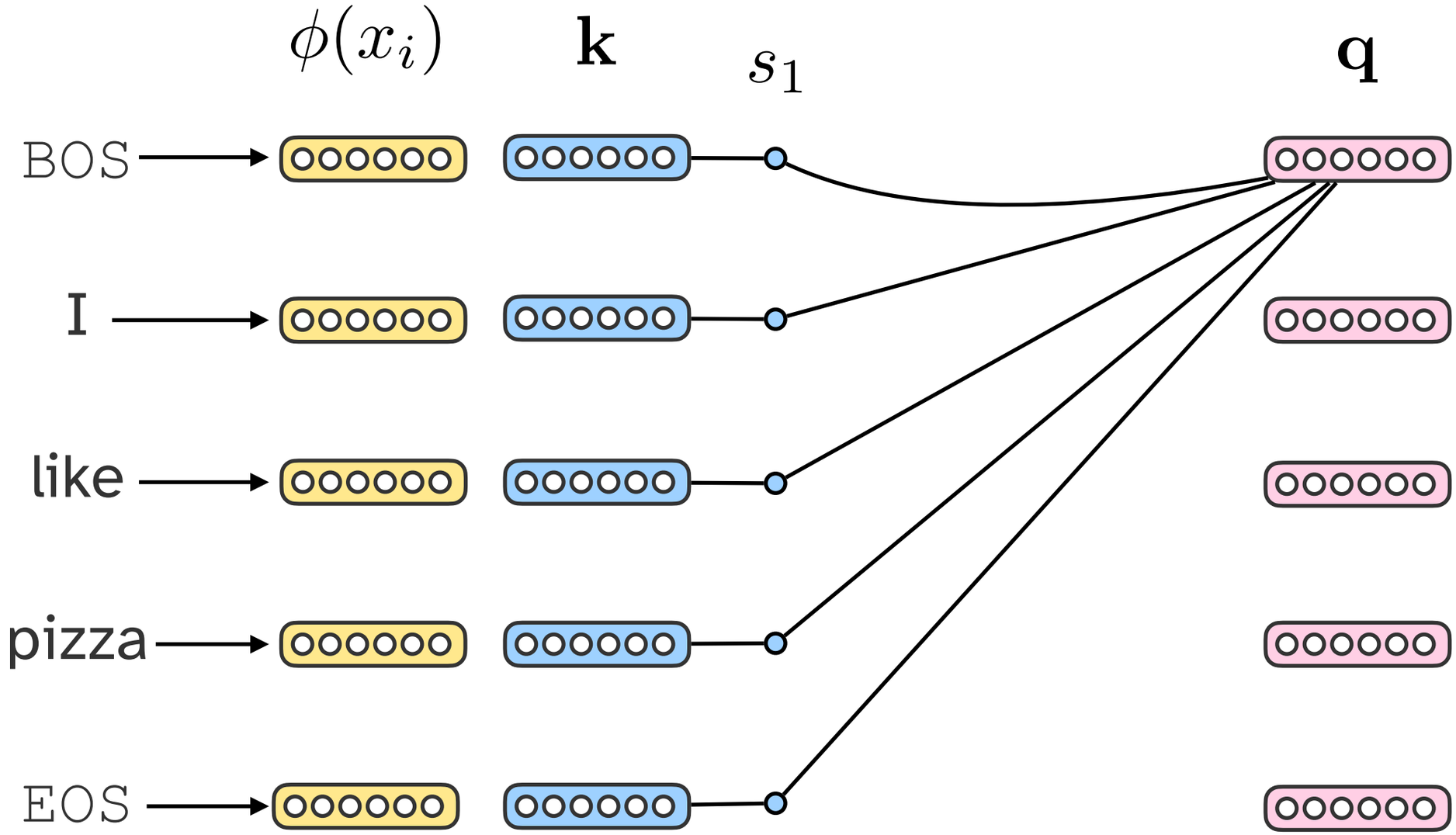
Self-Attention in Encoders

Encode



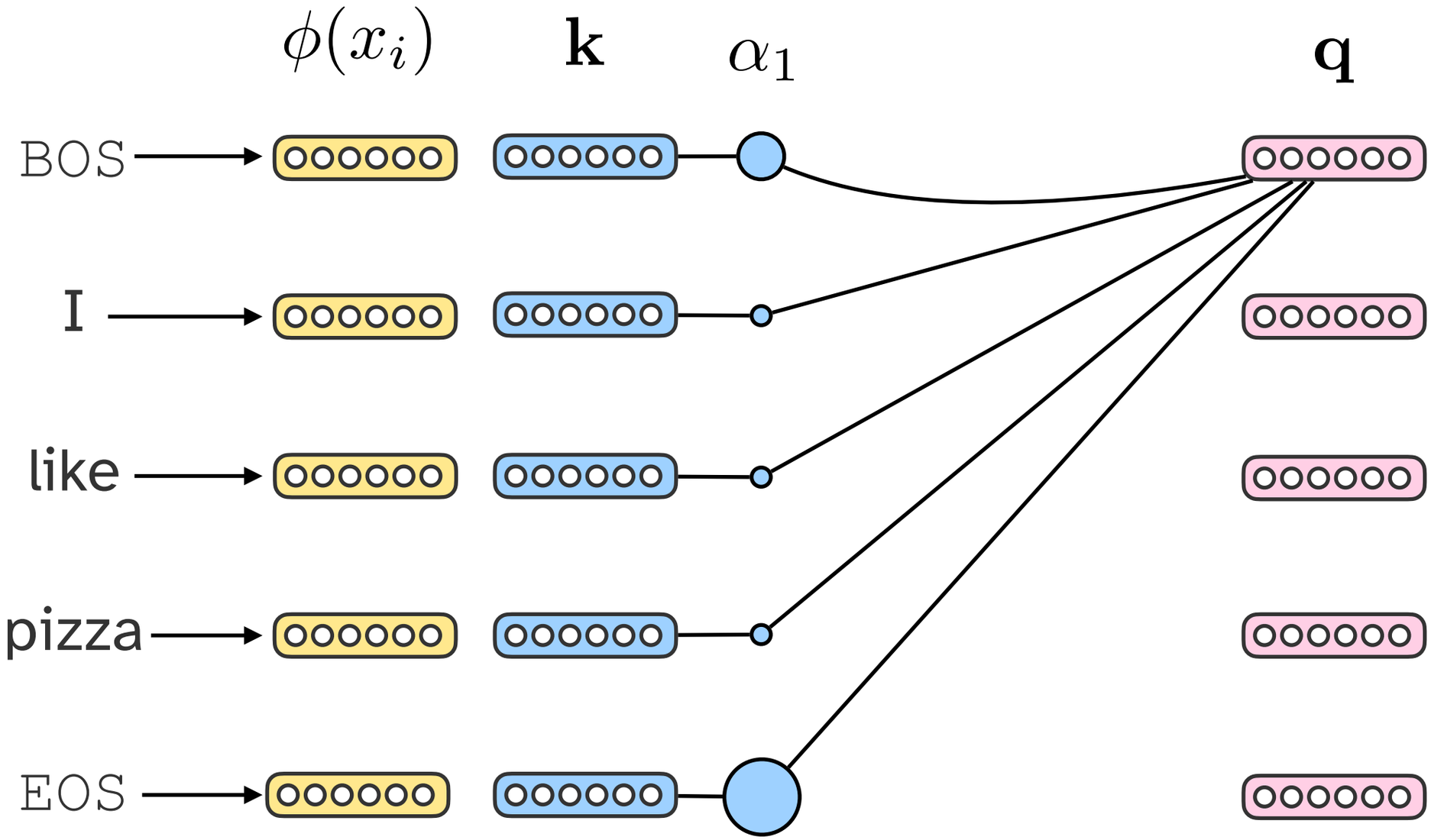
Self-Attention in Encoders

Encode



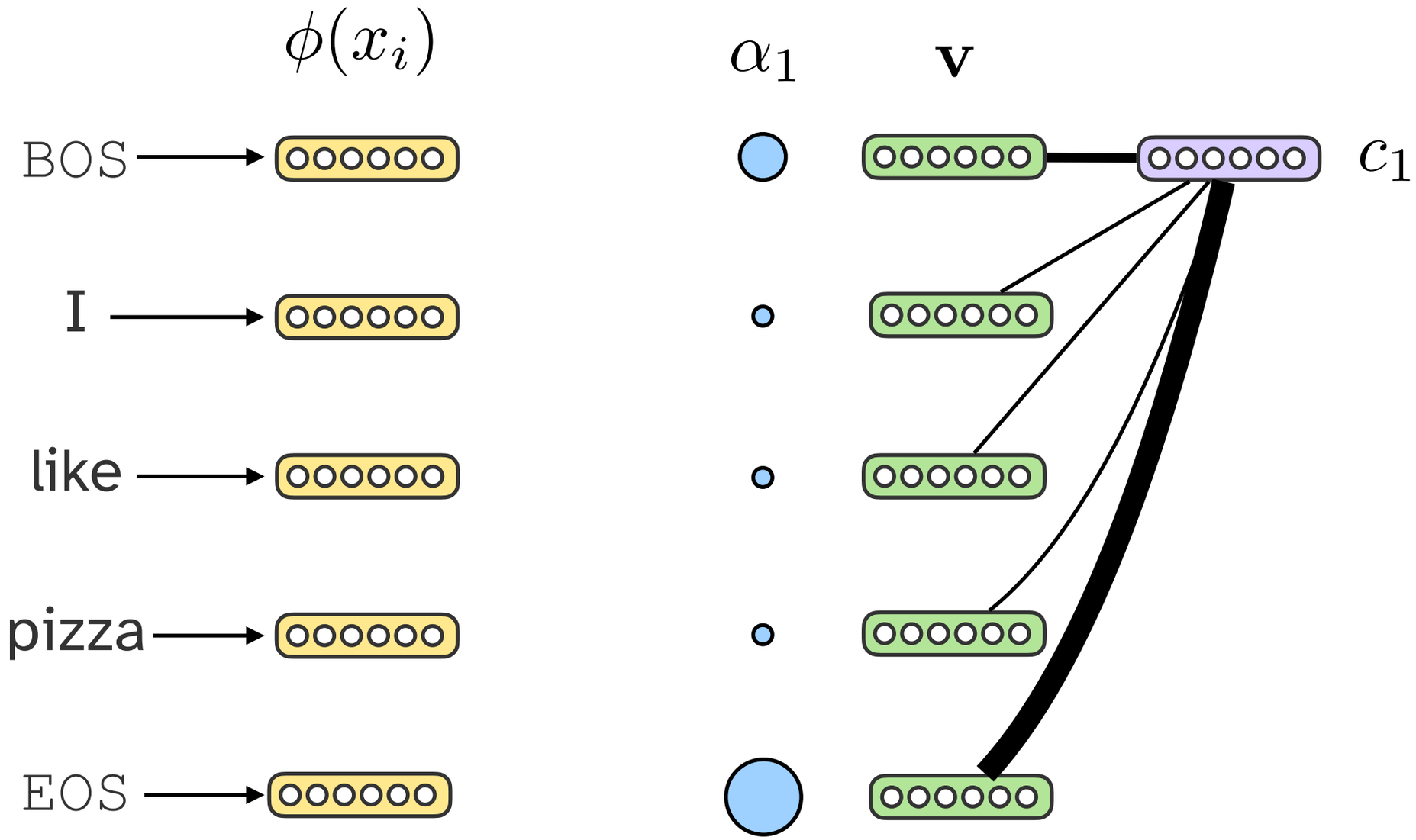
Self-Attention in Encoders

Encode



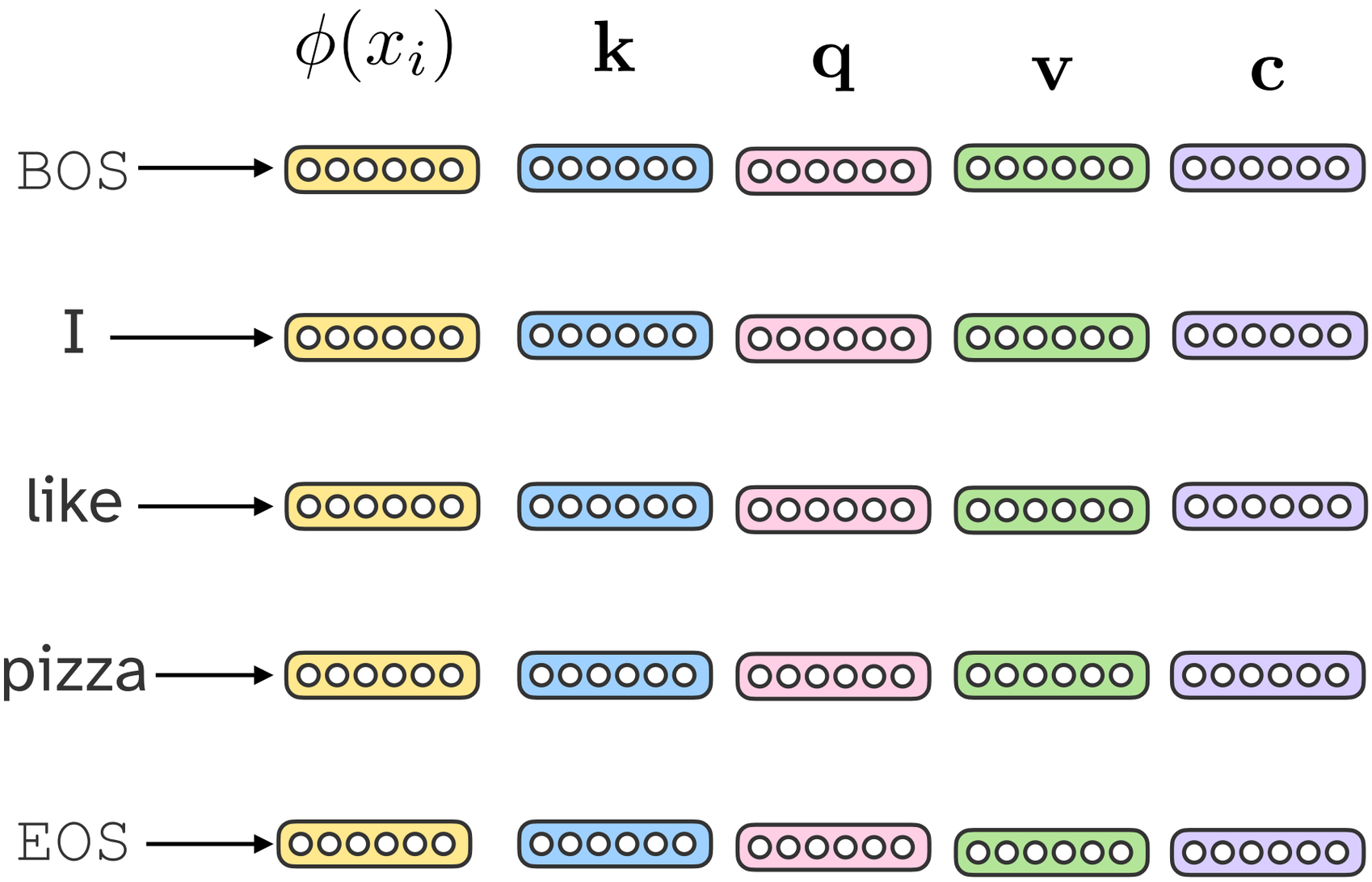
Self-Attention in Encoders

Encode



Self-Attention in Encoders

Encode



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

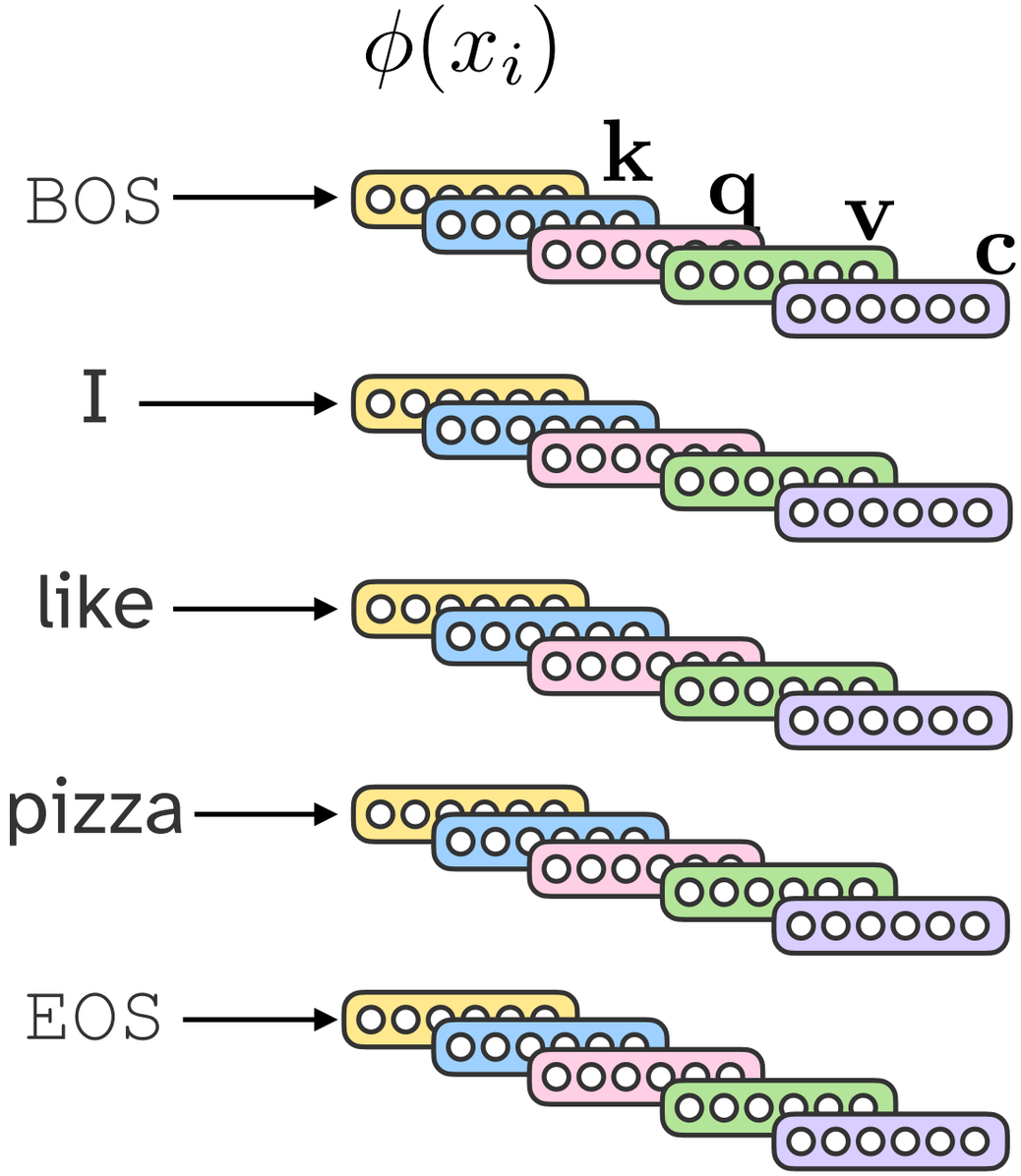
$$\alpha = \mathrm{softmax}_{\mathrm{dim}=1}(\mathbf{s}) \in \{\Delta^{\mathbb{N}_{1:|\overline{x}|}}\}^{|\overline{x}|}$$

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

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

What Do We Lose?

Encode



Problem 1
No position information!

$$k_i = K\phi(x_i) \in \mathbb{R}^d$$

$$v_i = V\phi(x_i) \in \mathbb{R}^d$$

$$q_j = Q\phi(x_j) \in \mathbb{R}^d$$

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

$$\alpha = \mathrm{softmax}_{\mathrm{dim}=1}(\mathbf{s})$$

$$\in \left\{\Delta^{\mathbb{N}_{1:|\overline{x}|}}\right\}^{|\overline{x}|}$$

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

Problem 2
No nonlinearities!

Position Embeddings

- Currently, our model is operating only on embedding of individual wordtypes $\phi(x_i) = \phi(x)$
- Let's also embed their indices: $\phi(x_i) = \phi(x) + \phi(i)$

Position Embeddings

- Currently, our model is operating only on embedding of individual wordtypes $\phi(x_i) = \phi(x)$

- Let's also embed their indices: $\phi(x_i) = \phi(x) + \phi(i)$

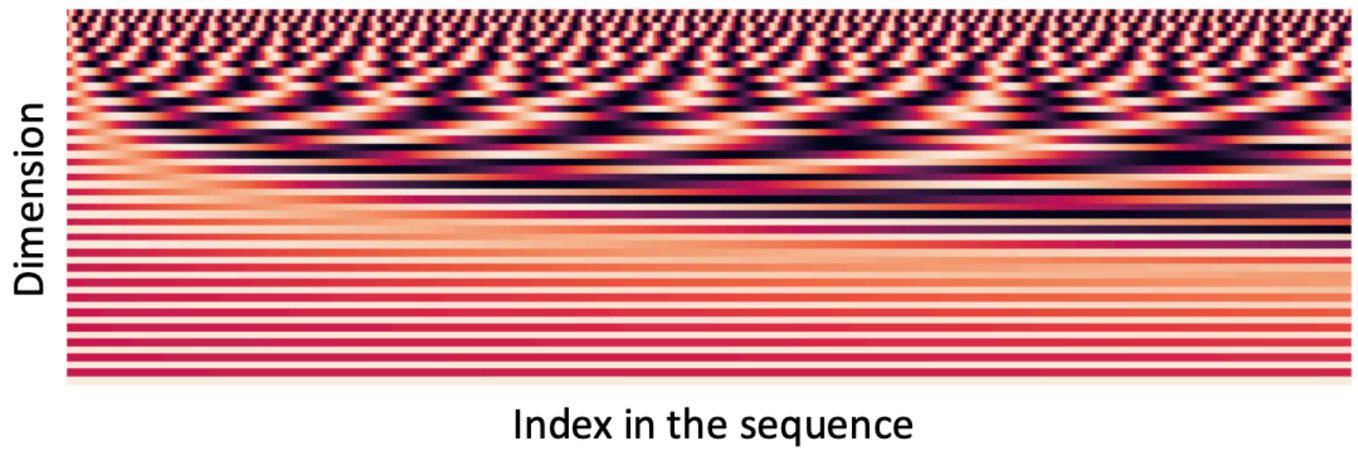
- First option: just learn absolute representations

- But unlike vocabularies of fixed sizes, sequences can be of arbitrary lengths...

Position Embeddings

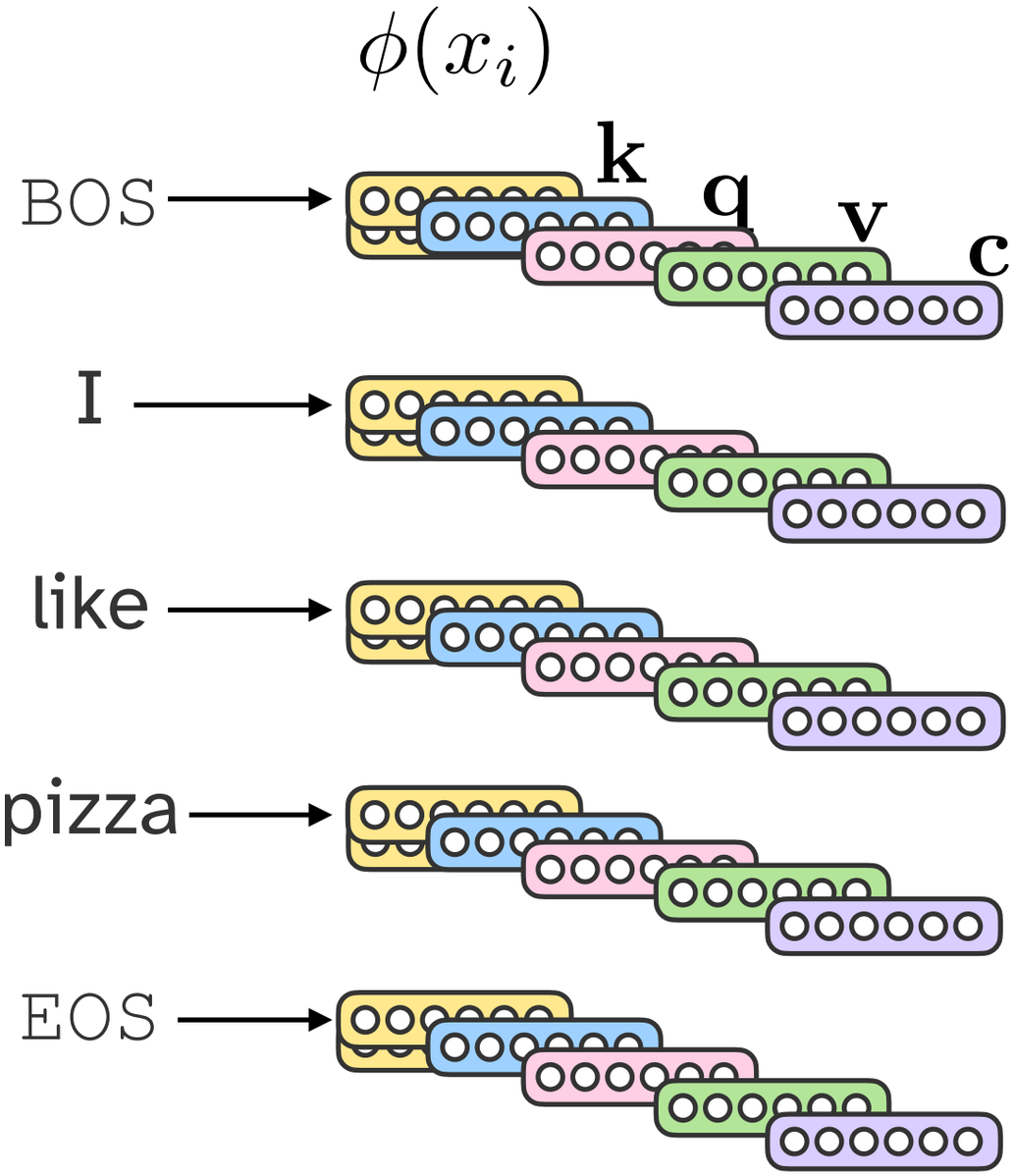
- Currently, our model is operating only on embedding of individual wordtypes $\phi(x_i) = \phi(x)$
- Let's also embed their indices: $\phi(x_i) = \phi(x) + \phi(i)$
- Sinusoidal embeddings

$$\boldsymbol{p}_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



- (in practice, most methods use learned embeddings)

Position Embeddings



$$k_i = K\phi(x_i) \in \mathbb{R}^d$$

$$v_i = V\phi(x_i) \in \mathbb{R}^d$$

$$q_j = Q\phi(x_j) \in \mathbb{R}^d$$

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

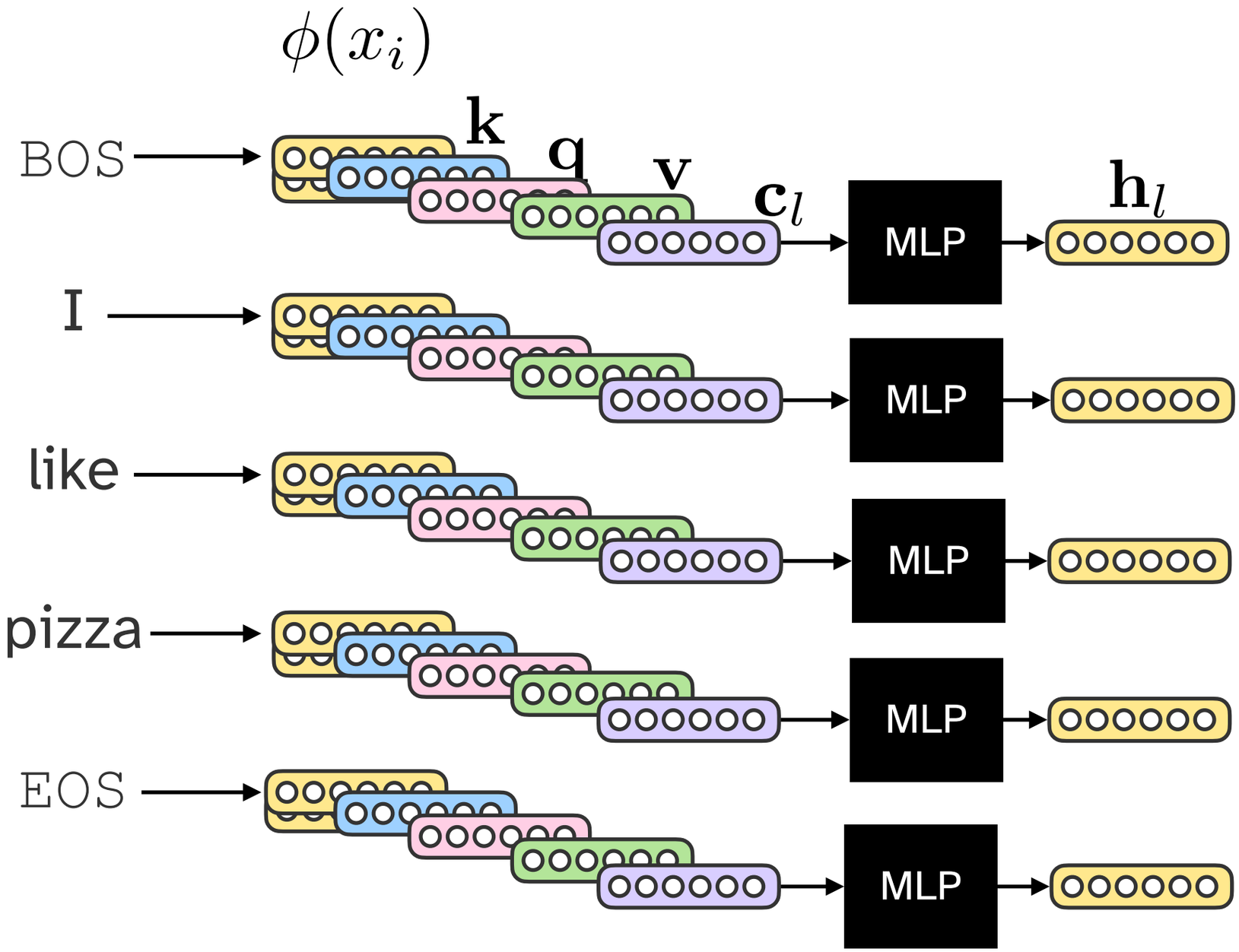
$$\alpha = \mathrm{softmax}_{\mathrm{dim}=1}(\mathbf{s})$$

$$\in \left\{\Delta^{\mathbb{N}_{1:|\overline{x}|}}\right\}^{|\overline{x}|}$$

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

Problem 2
No nonlinearities!

Nonlinearities



$$k_i = K\phi(x_i) \in \mathbb{R}^d$$

$$v_i = V\phi(x_i) \in \mathbb{R}^d$$

$$q_j = Q\phi(x_j) \in \mathbb{R}^d$$

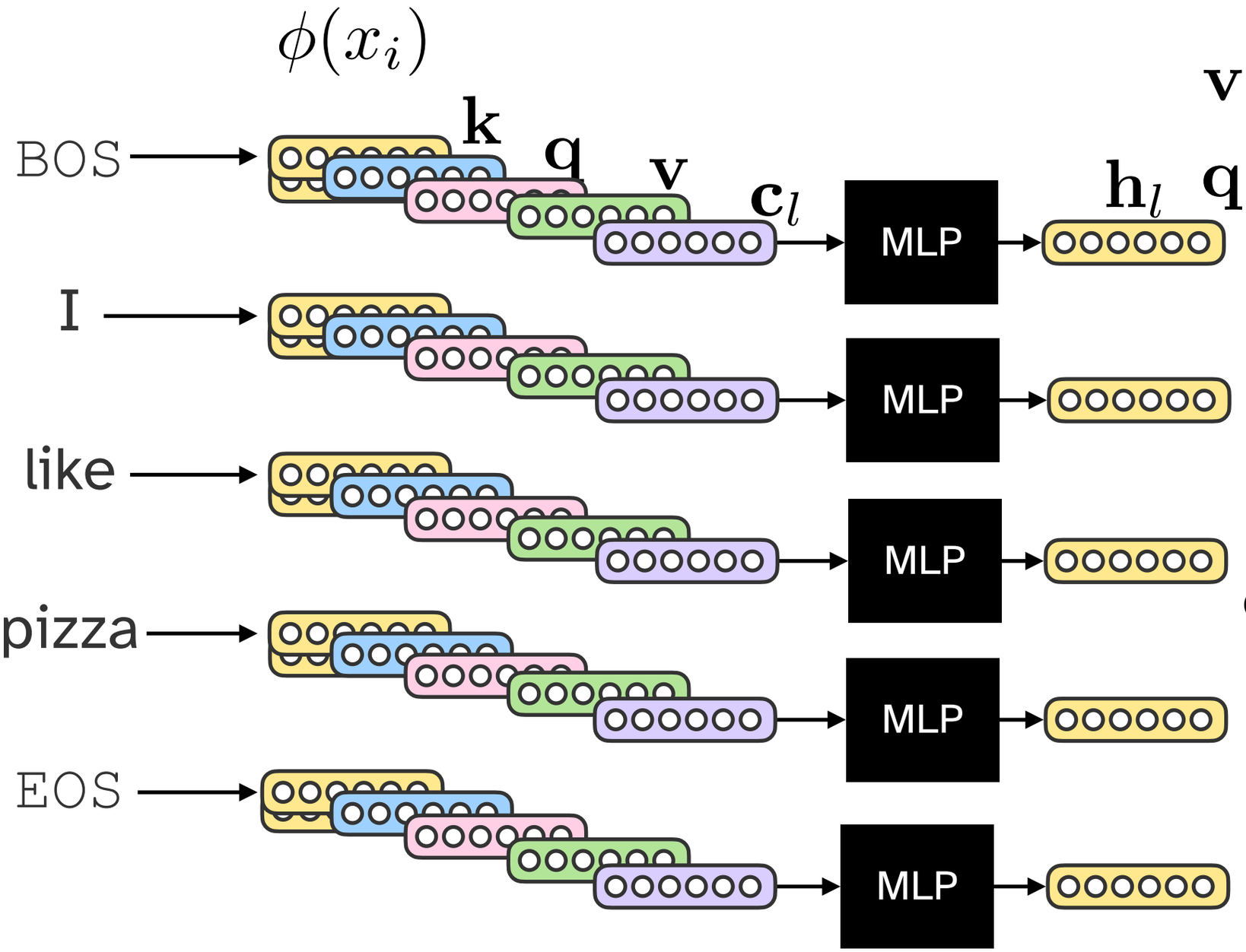
$$\mathbf{s} = \mathbf{q}^\intercal \mathbf{k} \in \mathbb{R}^{|\overline{x}| \times |\overline{x}|}$$

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

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

$$\mathbf{h}_l = W_2^{(l)}\mathrm{ReLU}(W_1^{(l)})\mathbf{c}_l + b_1^{(l)}) + b_2^{(l)}$$

Nonlinearities



$$\mathbf{k}^{(l)} = K^{(l)}\mathbf{h}_l$$

$$\mathbf{v}^{(l)} = V^{(l)}\mathbf{h}_l$$

$$\mathbf{q}^{(l)} = Q^{(l)}\mathbf{h}_l$$

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

$$\alpha = \mathrm{softmax}_{\mathrm{dim}=1}(\mathbf{s})$$

$$\in \{\Delta^{\mathbb{N}_{1:|\overline{x}|}}\}^{|\overline{x}|}$$

$$\mathbf{c}_l = \alpha\mathbf{v}^{(l)}$$

$$\mathbf{h}_l = W_2^{(l)}\mathrm{ReLU}(W_1^{(l)})\mathbf{c}_l + b_1^{(l)}) + b_2^{(l)}$$