Neural Sequence Modeling

EECS 183/283a: Natural Language Processing

N-Gram Approximation

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

Autoregressive approximation

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

N-gram approximation

$$\approx \prod_{i=1}^{|\overline{x}|} \frac{C(x_{i-n+1}, \ldots, x_{i-1}, x_i)}{C(i_{i-n+1}, \ldots, x_{i-1})}$$

Count-based approximation

N-Gram Approximation

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

Autoregressive approximation

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

N-gram approximation

$$\approx \prod_{i=1}^{|\overline{x}|} \frac{C(x_{i-n+1}, \ldots, x_{i-1}, x_i)}{C(i_{i-n+1}, \ldots, x_{i-1})}$$

Count-based approximation

$$\approx \prod_{i=1}^{|\overline{x}|} p(x_i \mid x_{i-n+1}, \ldots, x_i; \theta)$$

Neural n-grams

Neural N-Grams

$$p(x_i \mid x_{i-n+1}, \ldots, x_i; \theta)$$

$$= \frac{\exp(s(x_i | x_{i-n+1}, \ldots, x_{i-1}; \theta))}{\sum_{x' \in \mathcal{V}} \exp(s(x' | x_{i-n+1}, \ldots, x_{i-1}; \theta))}$$

Softmax over wordtype scores

Neural N-Grams

$$p(x_i \mid x_{i-n+1}, \ldots, x_i; \theta)$$

$$= \frac{\exp(s(x_i | x_{i-n+1}, \ldots, x_{i-1}; \theta))}{\sum_{x' \in \mathcal{V}} \exp(s(x' | x_{i-n+1}, \ldots, x_{i-1}; \theta))}$$

Softmax over **wordtype scores**

$$s(x \mid x_{i-n+1}, \ldots, x_{i-1}; \theta) = f_\theta(\phi(x_{i-n+1}), \ldots, \phi(x_{i-1}))[x]$$

Neural model

takes *n*-1 inputs:

embeddings of each prefix wordtype

The Feedforward N-Gram Model

Input tokens (prefix)

x_{i-n+1}

x_{i-n+2}

x_{i-1}

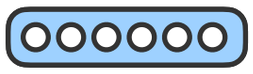
Bengio et al. 2003

The Feedforward N-Gram Model

Input tokens (prefix)

Word embeddings

$x_{i-n+1} \longrightarrow \phi(x_{i-n+1})$



$x_{i-n+2} \longrightarrow \phi(x_{i-n+2})$

...

$x_{i-1} \longrightarrow \phi(x_{i-1})$

Bengio et al. 2003

The Feedforward N-Gram Model

Input tokens (prefix)

Word embeddings

Concatenate word representations

Mix features

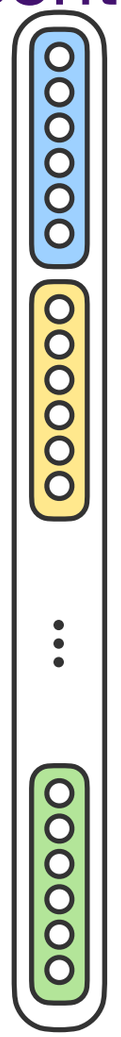
Scores over vocabulary

$x_{i-n+1} \longrightarrow \phi(x_{i-n+1})$

$x_{i-n+2} \longrightarrow \phi(x_{i-n+2})$

...

$x_{i-1} \longrightarrow \phi(x_{i-1})$



$v \in \mathbb{R}^{(n-1)\cdot d}$

Multi-layer perceptron (MLP) with nonlinearities

E.g., in Bengio et al:

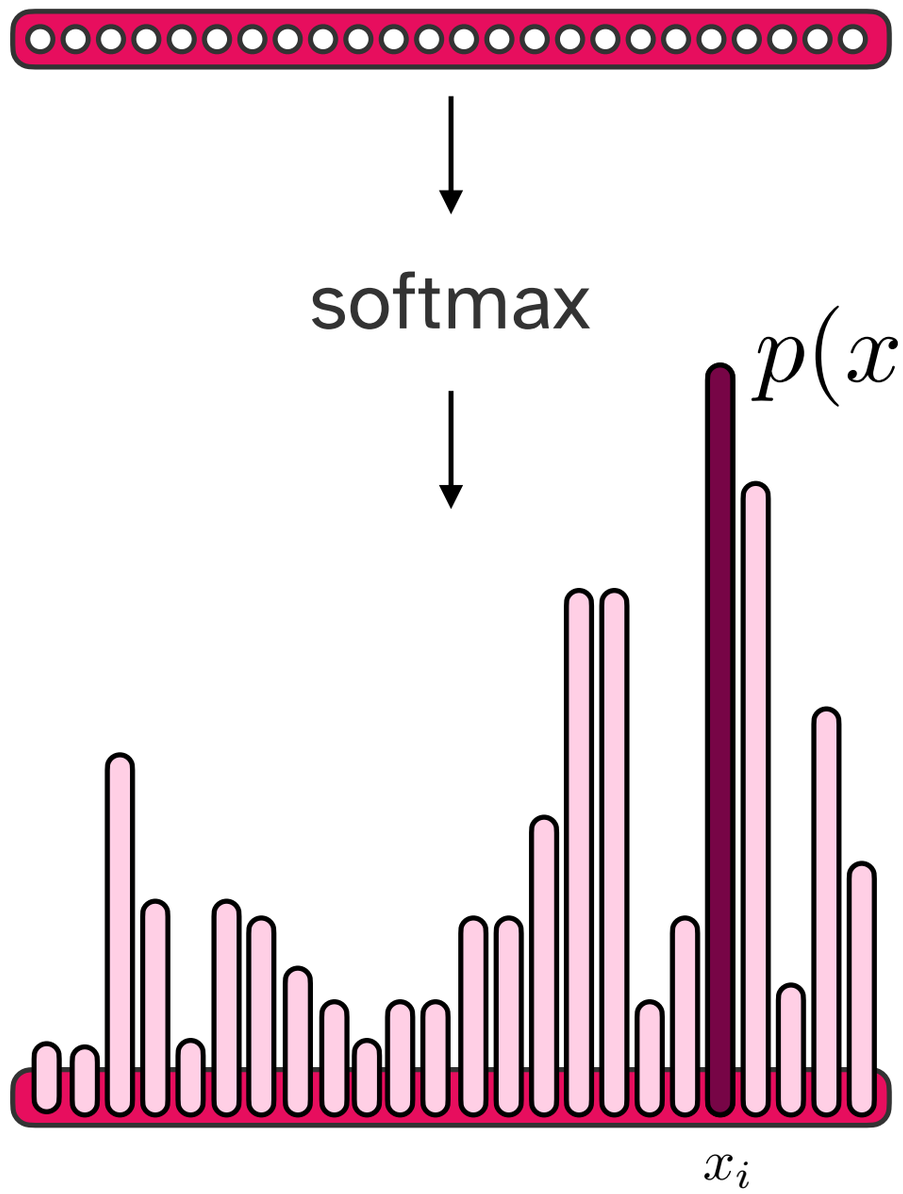
$$s = b + Wv + U \tanh(c + Hv)$$

$s \in \mathbb{R}^{|\mathcal{V}|}$

Bengio et al. 2003

The Feedforward N-Gram Model

Scores over vocabulary



Probabilities over vocabulary

Model Parameters θ

Input tokens (prefix)

x_{i-n+1}

x_{i-n+2}

...

x_{i-1}

Word embeddings

$\phi(x_{i-n+1})$

$\phi(x_{i-n+2})$

$\phi(x_{i-1})$

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

Concatenate word representations

$v \in \mathbb{R}^{(n-1) \cdot d}$

Mix features

Multi-layer perceptron (MLP) with nonlinearities

E.g., in Bengio et al:

$$s = b + Wv + U \tanh(c + Hv)$$

$$H \in \mathbb{R}^{d(n-1) \times h}$$
$$c \in \mathbb{R}^{h}$$
$$U \in \mathbb{R}^{h \times |\mathcal{V}|}$$
$$W \in \mathbb{R}^{d(n-1) \times |\mathcal{V}|}$$
$$b \in \mathbb{R}^{|\mathcal{V}|}$$

Scores over vocabulary

$s \in \mathbb{R}^{|\mathcal{V}|}$

Bengio et al. 2003

Model Parameters θ

$$\phi \in \mathbb{R}^{|\mathcal{V}| \times d}$$
$$H \in \mathbb{R}^{d(n-1) \times h}$$
$$c \in \mathbb{R}^{h}$$
$$U \in \mathbb{R}^{h \times |\mathcal{V}|}$$
$$W \in \mathbb{R}^{d(n-1) \times |\mathcal{V}|}$$
$$b \in \mathbb{R}^{|\mathcal{V}|}$$

$$C_n \in \mathbb{N}^{|\mathcal{V}|^n}$$
$$C_{n-1} \in \mathbb{N}^{|\mathcal{V}|^{n-1}}$$

Neural N-grams

```
n = 6
|V| = 20,000
d = 100
h = 60
```

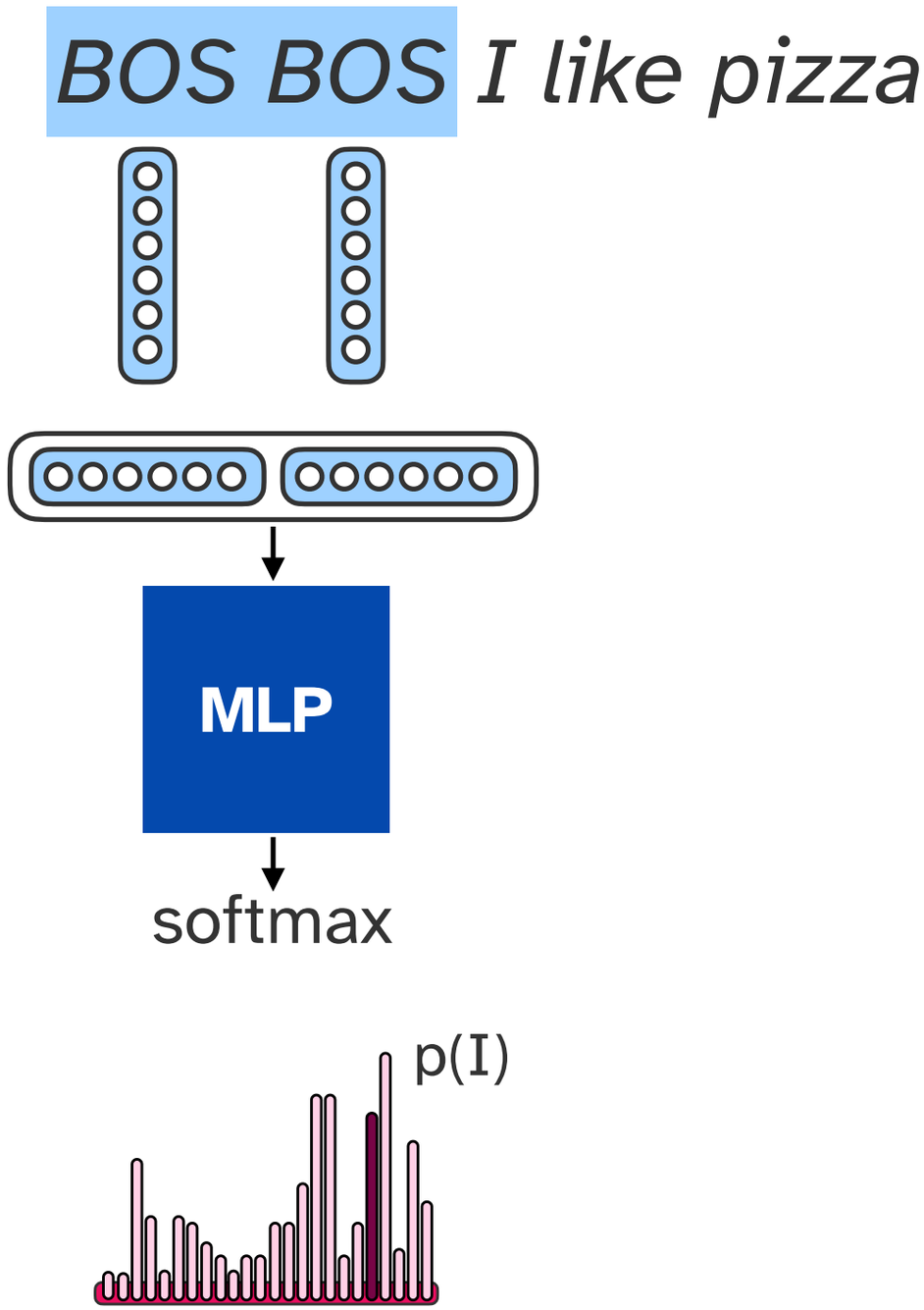
→ ~13M parameters

Count-based N-grams

6.4×10^{25}...

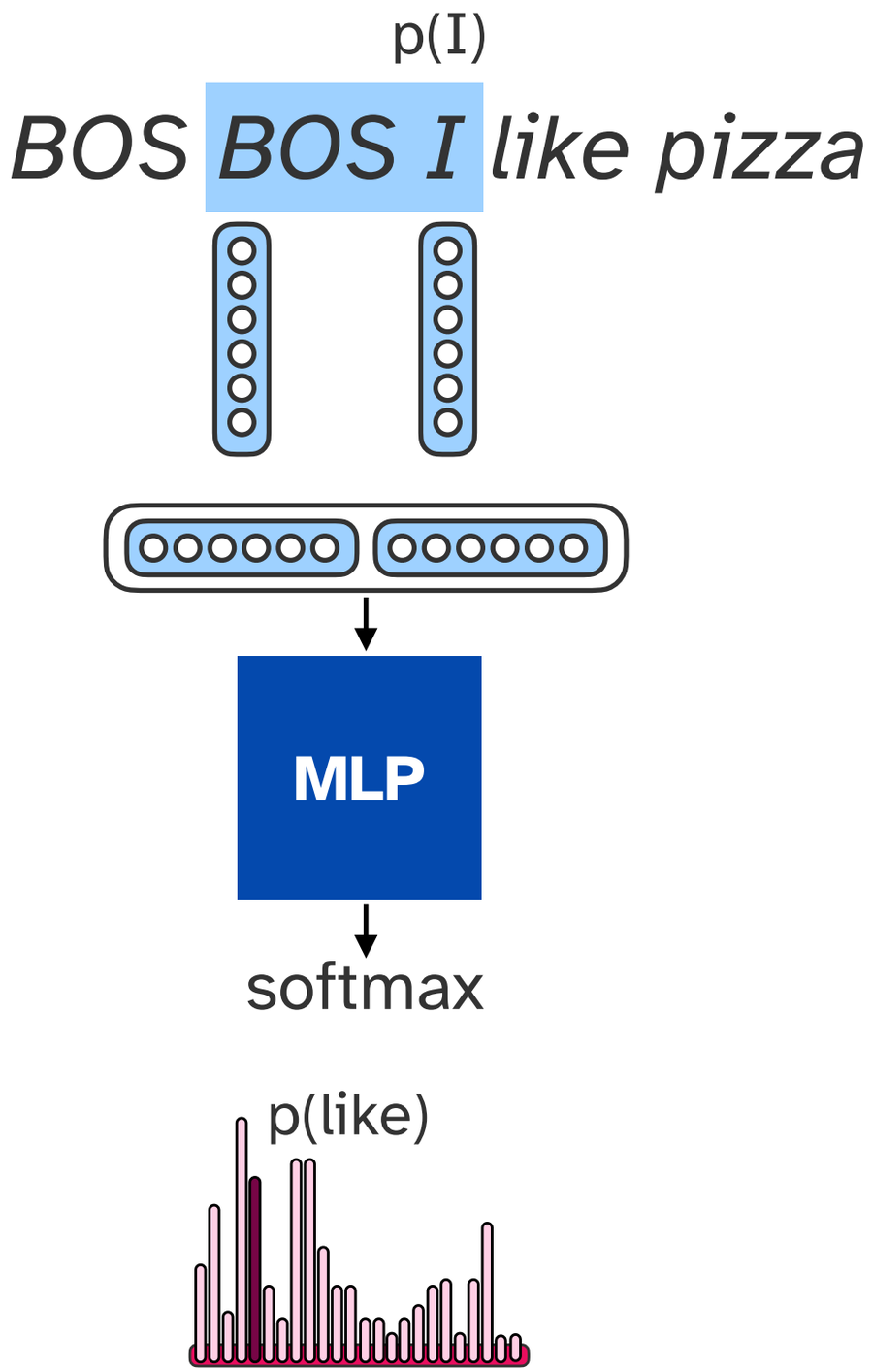
Feedforward N-Grams: Scoring Sequences

n=3



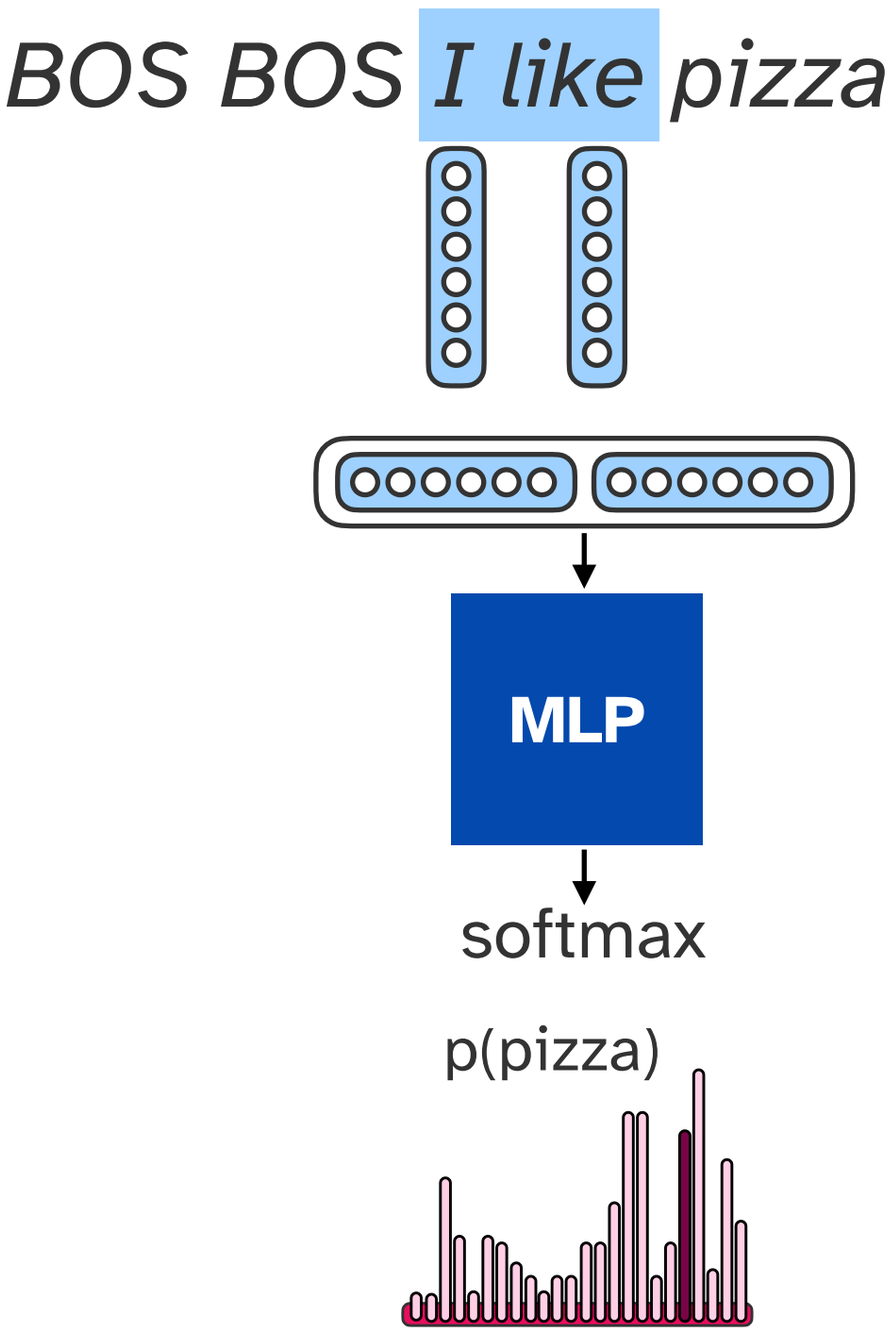
Feedforward N-Grams: Scoring Sequences

n=3



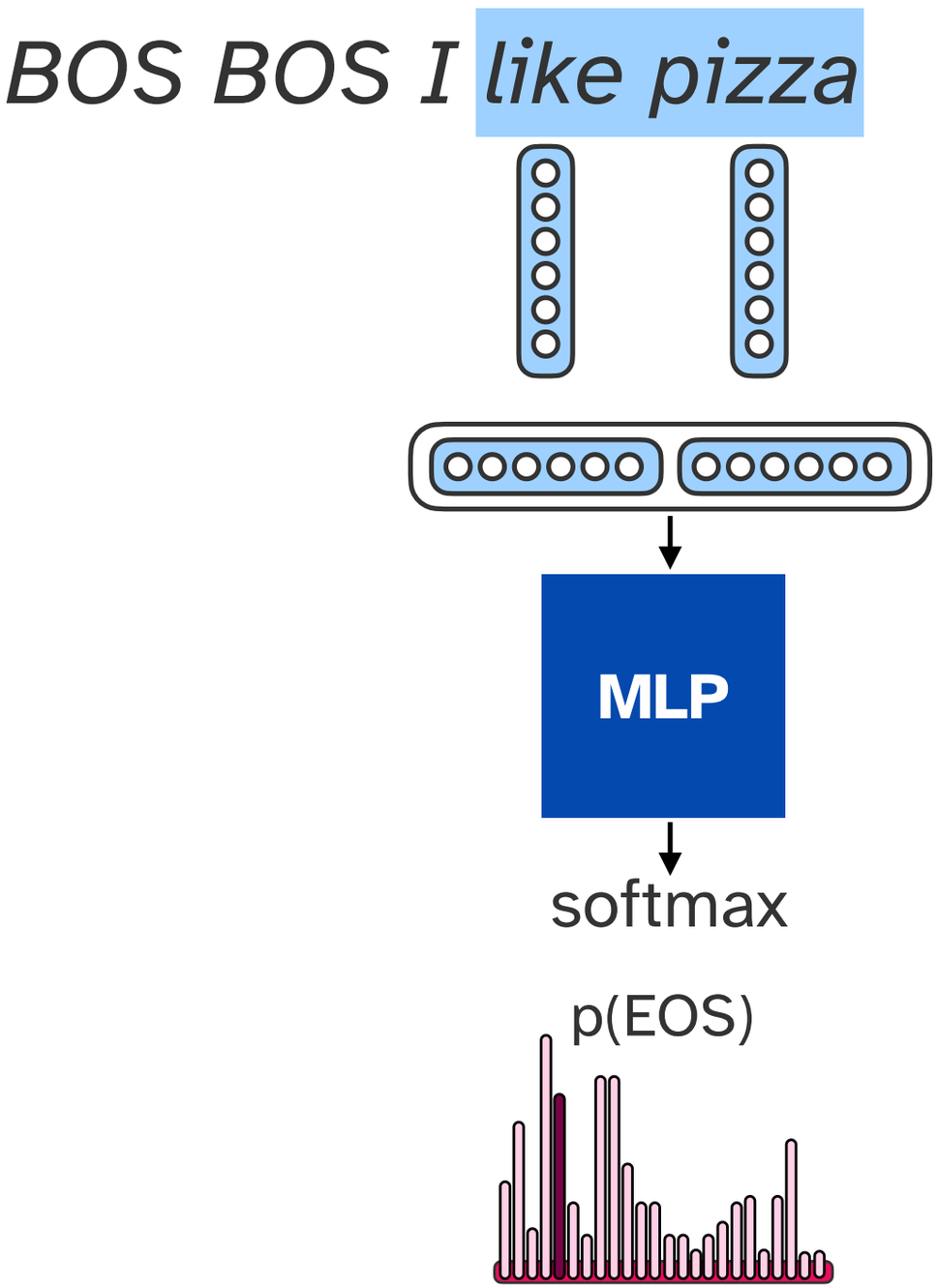
Feedforward N-Grams: Scoring Sequences

n=3



Feedforward N-Grams: Scoring Sequences

n=3



Feedforward N-Grams: Scoring Sequences

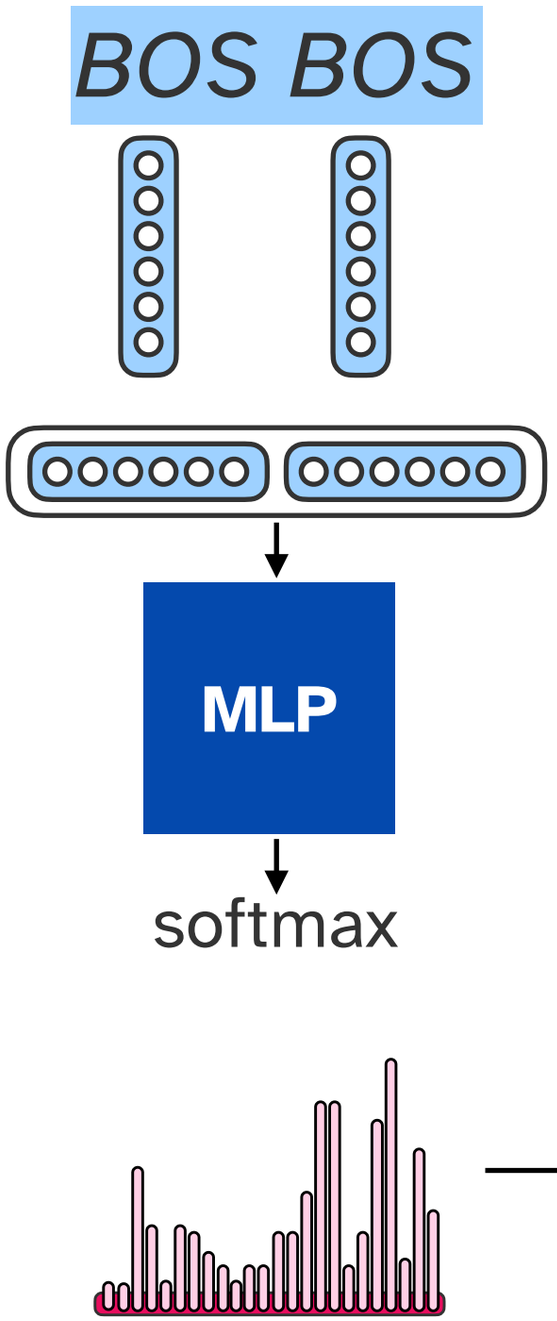
n=3

p(I) p(like) p(pizza) p(EOS)

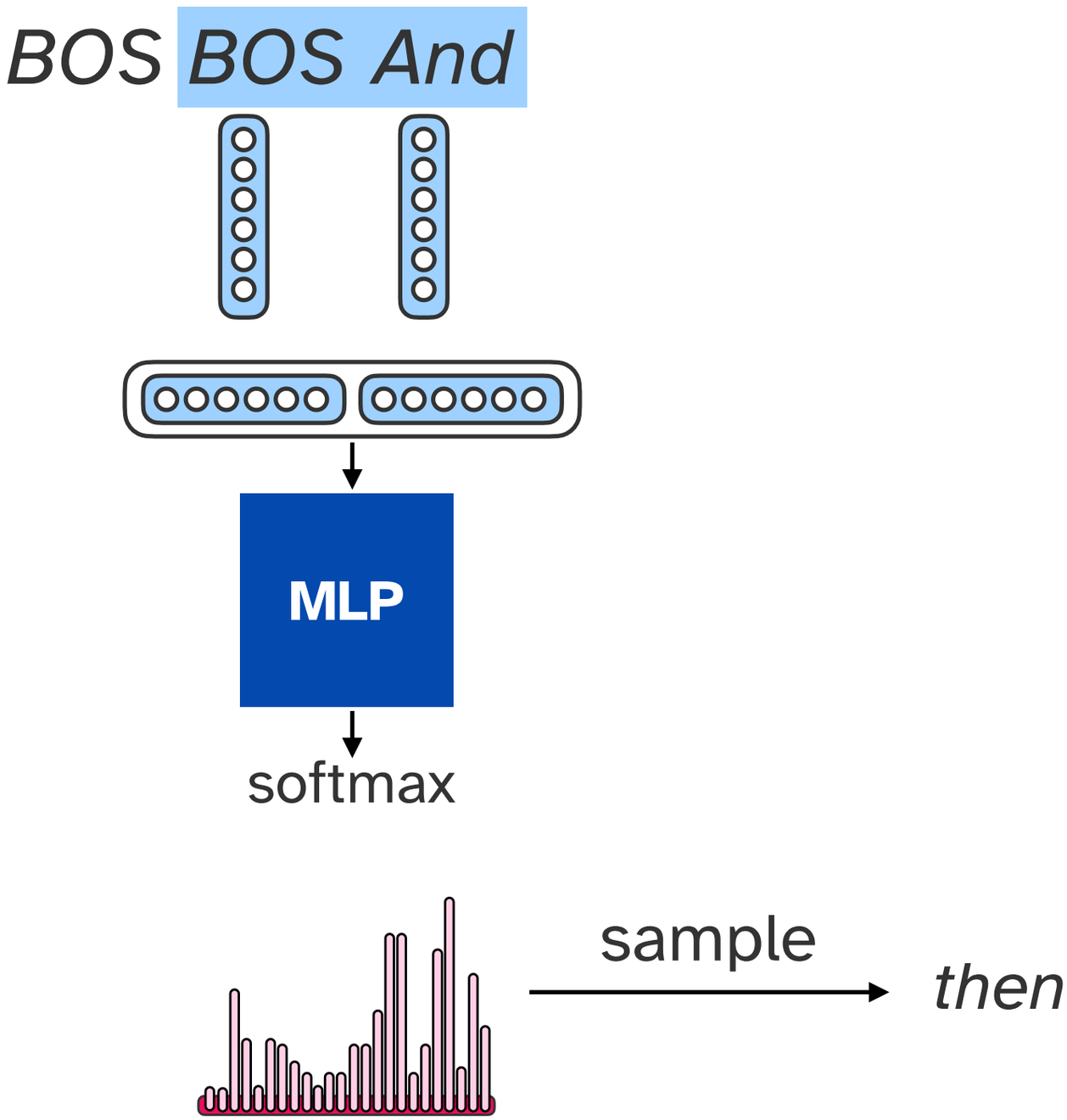
BOS BOS I like pizza

$$\exp\left(-\frac{1}{3}\left(\log p(\mathrm{I}) + \log p(\mathrm{like}) + \log p(\mathrm{pizza}) + p(\texttt{EOS})\right)\right)$$

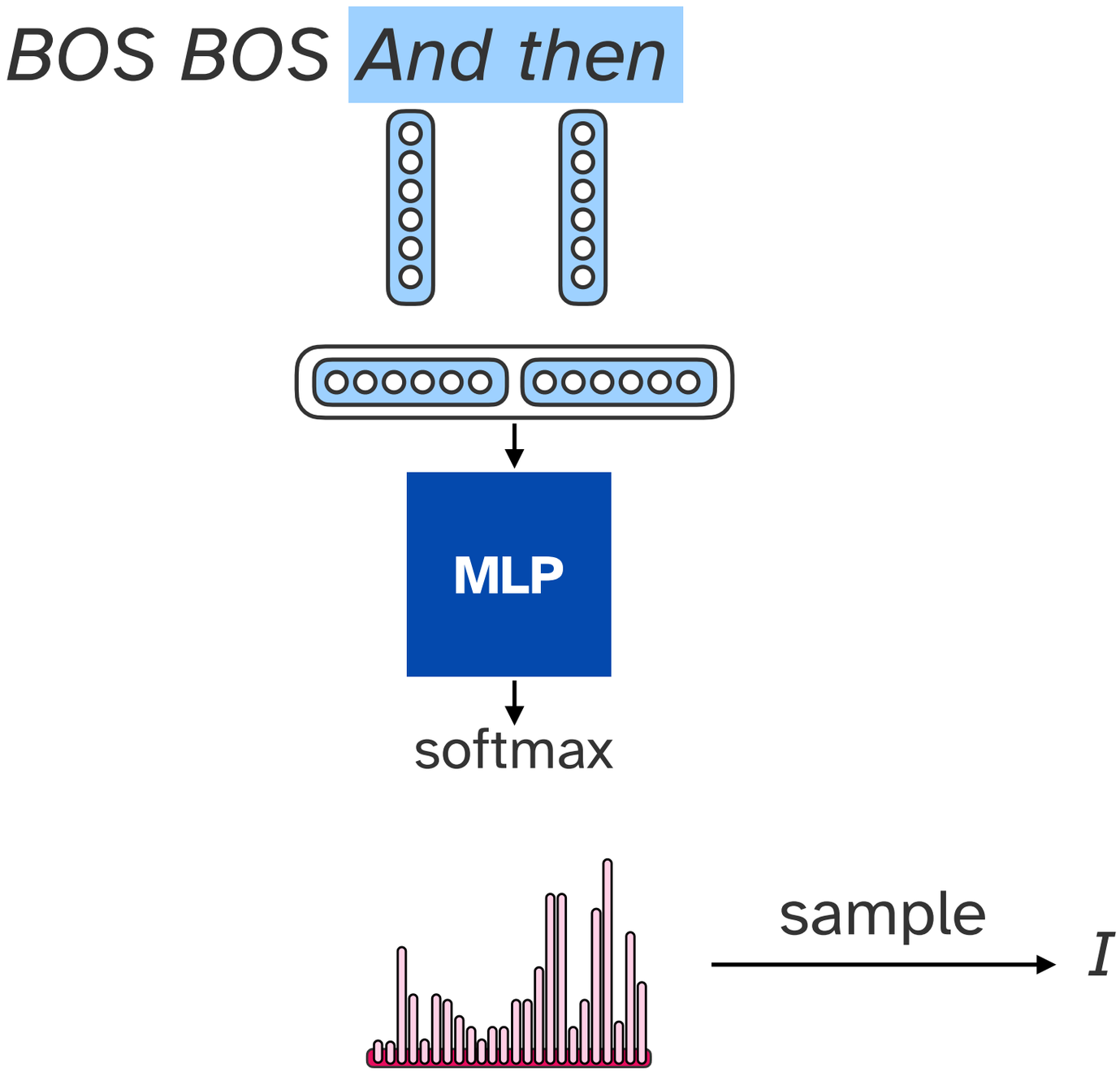
Feedforward N-Grams: Generation



Feedforward N-Grams: Generation



Feedforward N-Grams: Generation



Feedforward N-Grams: Generation

BOS BOS And then I …

Training Neural N-Grams

- We have some training dataset $\mathcal{D} = \{\overline{x}\}_{i=1}^{M}$
- We want to find the parameters of our neural model θ that maximize the likelihood of this dataset
- Train using gradient descent in log-probability space

$$\theta^* = \arg\min_{\theta} \left(-\sum_{\overline{x} \in \mathcal{D}} \sum_{i=1}^{|\overline{x}|} \log p(x_i \mid x_{i-n+1}, \ldots, x_{i-1}; \theta) \right)$$

$$\begin{aligned}
\phi &\in \mathbb{R}^{|\mathcal{V}| \times d} \\
H &\in \mathbb{R}^{d(n-1) \times h} \\
c &\in \mathbb{R}^{h} \\
U &\in \mathbb{R}^{h \times |\mathcal{V}|} \\
W &\in \mathbb{R}^{d(n-1) \times |\mathcal{V}|} \\
b &\in \mathbb{R}^{|\mathcal{V}|}
\end{aligned}$$

Drawbacks of Count-based N-Gram Models

- What if the count of the target *n*-gram is `0`?

Drawbacks of Count-based N-Gram Models

What if the count of the target *n*-gram is 0?

- What if our *n*-1-gram prefix has a count of 0?

Drawbacks of Count-based N-Gram Models

✔ What if the count of the target *n*-gram is 0?

✔ What if our *n*-1-gram prefix has a count of 0?

- Storage is at worst exponential wrt $|\mathcal{V}|$

Drawbacks of Count-based N-Gram Models

✔ What if the count of the target *n*-gram is 0?

✔ What if our *n*-1-gram prefix has a count of 0?

✔ Storage is at worst exponential wrt $|\mathcal{V}|$

- No notion of similarity

Drawbacks of Count-based N-Gram Models

- ✓ What if the count of the target *n*-gram is 0?
- ✓ What if our *n*-1-gram prefix has a count of 0?
- ✓ Storage is at worst exponential wrt $|\mathcal{V}|$
- ✓ No notion of similarity
- Intervening words

Drawbacks of Count-based N-Gram Models

- ✓ What if the count of the target *n*-gram is 0?
- ✓ What if our *n*-1-gram prefix has a count of 0?
- ✓ Storage is at worst exponential wrt $|\mathcal{V}|$
- ✓ No notion of similarity
- ✓ Intervening words
- ✗ Without a big *n*, cannot handle long-distance dependencies

N-Gram Approximation

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

Autoregressive approximation

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

N-gram approximation

$$\approx \prod_{i=1}^{|\overline{x}|} \frac{C(x_{i-n+1}, \ldots, x_{i-1}, x_i)}{C(i_{i-n+1}, \ldots, x_{i-1})}$$

Count-based approximation

$$\approx \prod_{i=1}^{|\overline{x}|} p(x_i \mid x_{i-n+1}, \ldots, x_i; \theta)$$

Neural n-grams

Still can't represent contexts past *n*

Infinite Context?

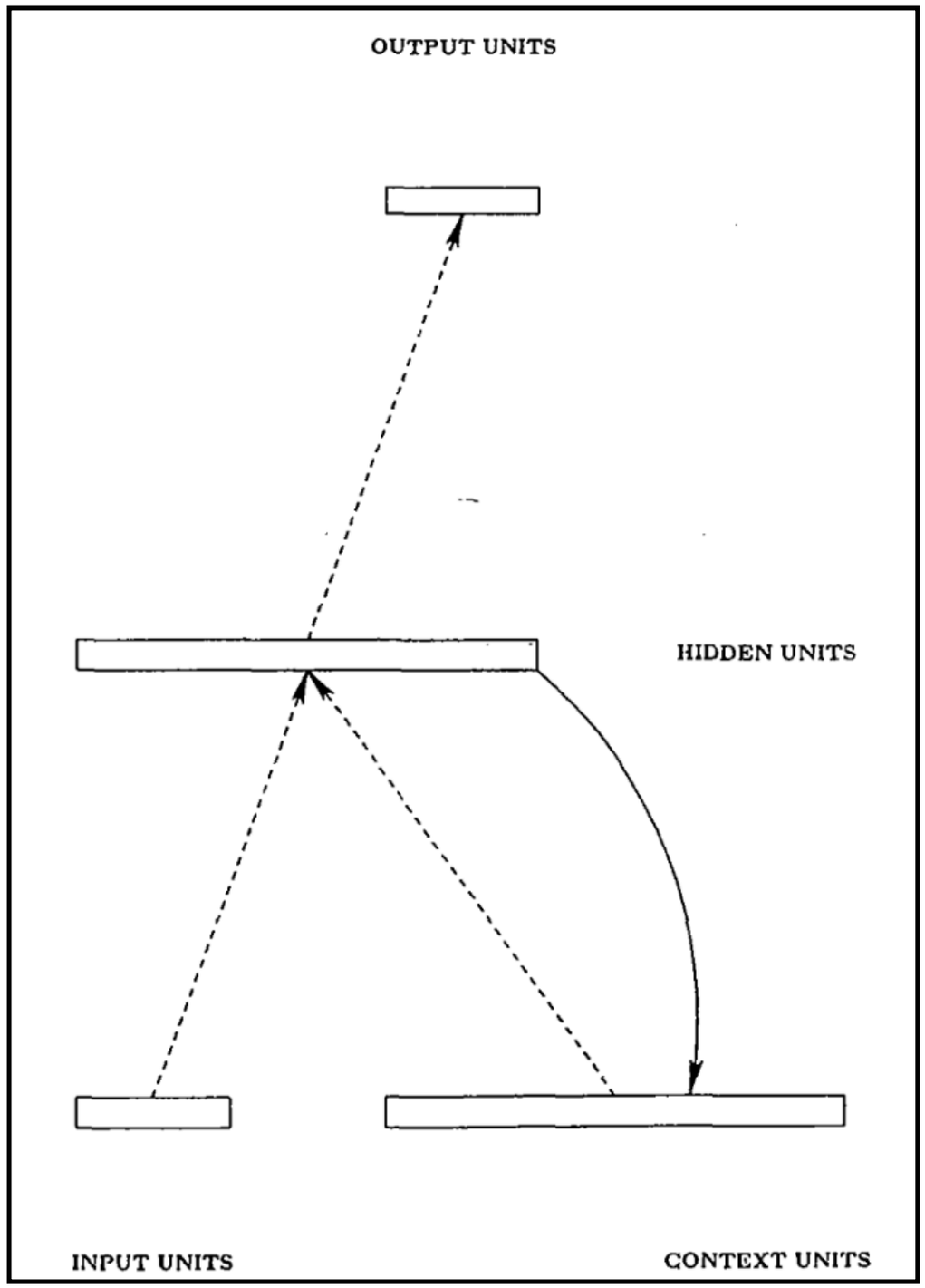
$$p(\overline{x}) = \prod_{i=1}^{|\overline{x}|} p(x_i \mid x_1, \ldots, x_{i-1})$$

- N-gram models (count-based or neural) impose fixed windows of context

- Could we instead truly compute $p(x_i \mid x_1, \ldots, x_{i-1})$ for any *i*?

Recurrent Neural Network

- **Key idea:** add a latent space whose values are both:
  - determined by computations performed during previous “timesteps”, and
  - taken in as input at each timestep
- The value of this latent space is called a “hidden state”



Elman et al. 1990

Recurrent Neural Network

Input tokens

Word embeddings

Hidden states

h_0 (learned or null)

x_1 → $\phi(x_1)$ → h_1

$h_i = \sigma(W_h h_{i-1} + W_e \phi(x_i) + b_1)$

x_2 → $\phi(x_2)$ → h_2

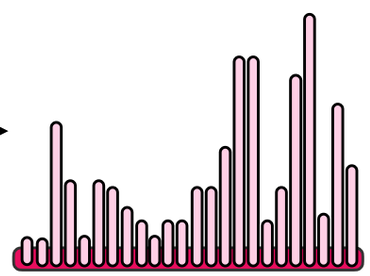
...

...

x_{i-1} → $\phi(x_{i-1})$ → h_{i-1}

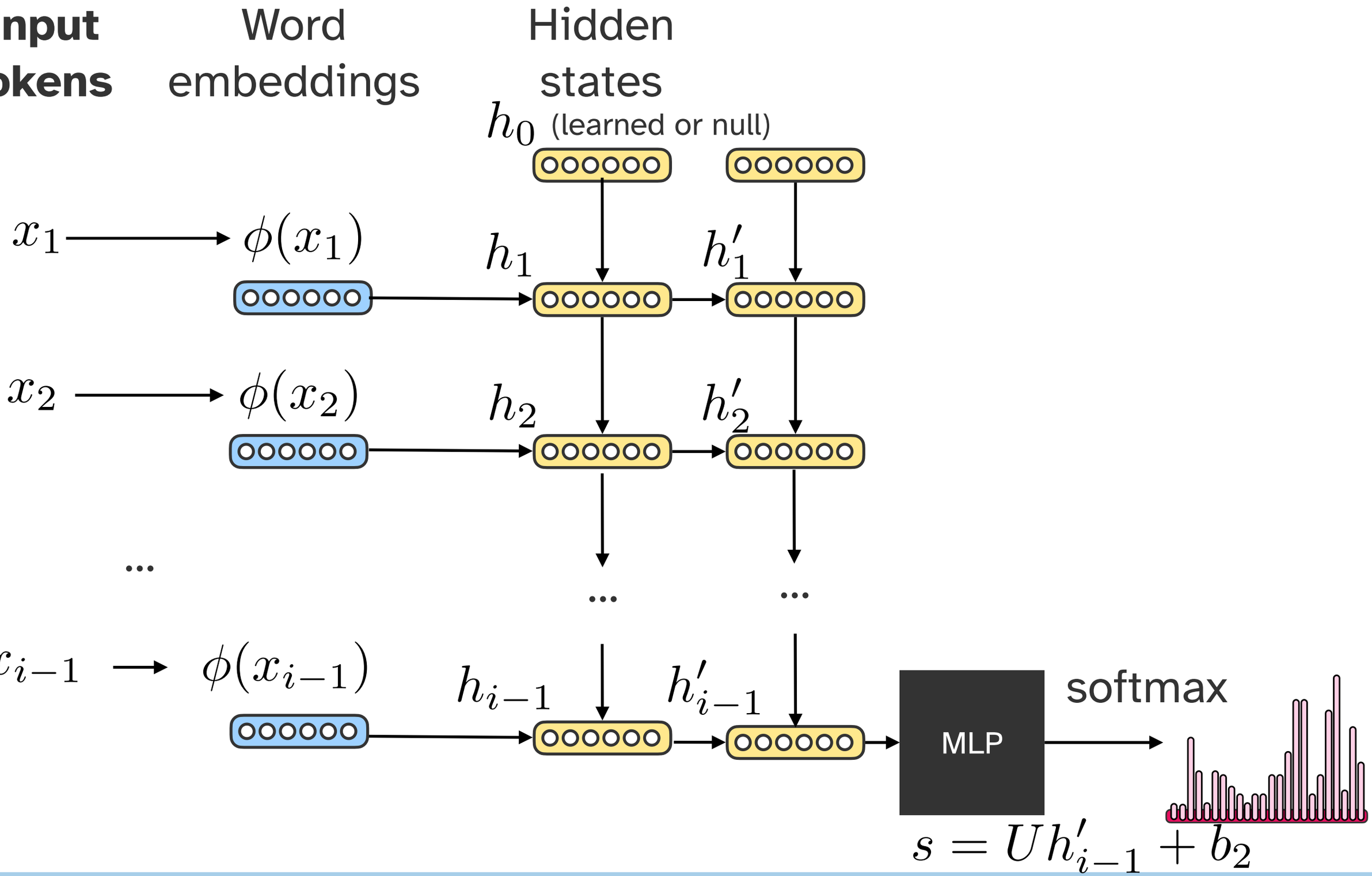
MLP

softmax



$s = U h_{i-1} + b_2$

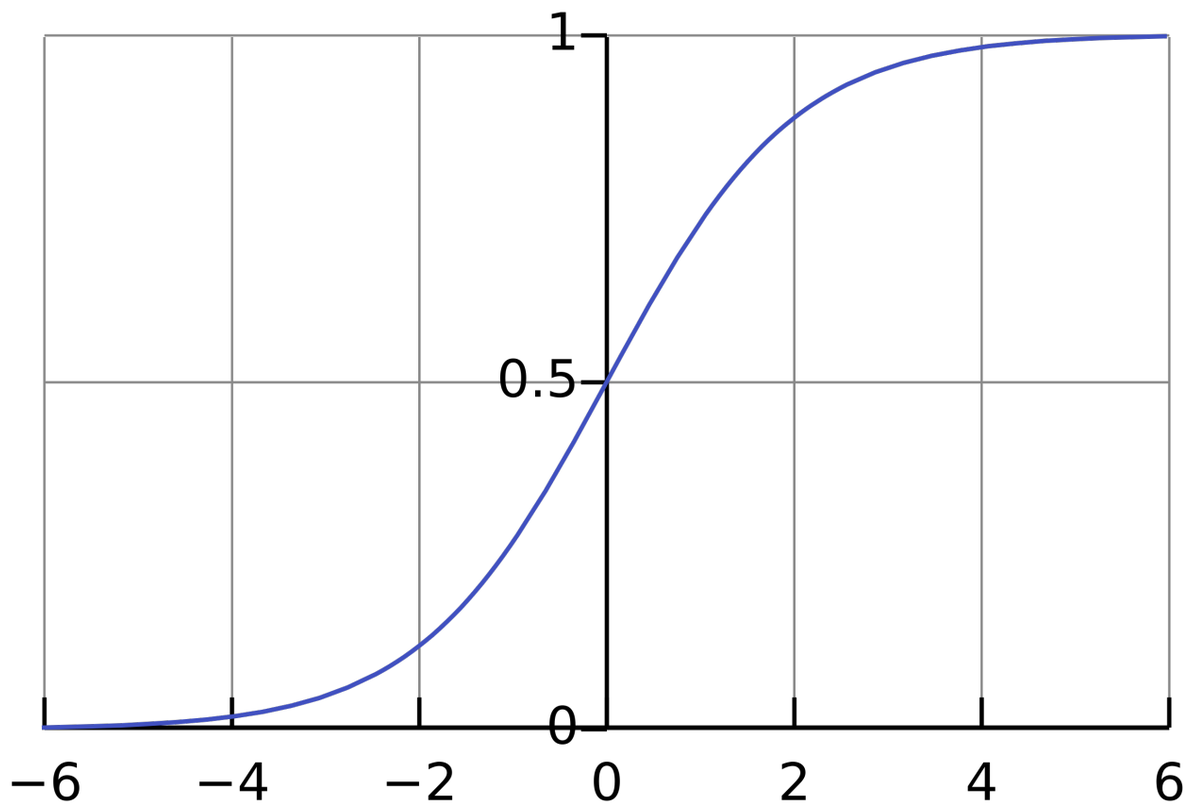
Multi-Layered RNN



RNN Parameters

$$h_i = \sigma(W_h h_{i-1} + W_e \phi(x_i) + b_1)$$
$$s = U h'_{i-1} + b_2$$

$$\phi(\cdot) \in \mathbb{R}^{|\mathcal{V}| \times d}$$
$$W_e \in \mathbb{R}^{d \times d'}$$
$$b_1 \in \mathbb{R}^{d'}$$
$$W_h \in \mathbb{R}^{d' \times d'}$$
$$U \in \mathbb{R}^{d' \times |\mathcal{V}|}$$
$$b_2 \in \mathbb{R}^{|\mathcal{V}|}$$



RNN Parameters

$$\phi \in \mathbb{R}^{|\mathcal{V}| \times d}$$
$$H \in \mathbb{R}^{d(n-1) \times h}$$
$$c \in \mathbb{R}^{h}$$
$$U \in \mathbb{R}^{h \times |\mathcal{V}|}$$
$$W \in \mathbb{R}^{d(n-1) \times |\mathcal{V}|}$$
$$b \in \mathbb{R}^{|\mathcal{V}|}$$

Neural N-grams

```
n = 6
|V| = 20,000
d = 100
h = 60
```

➡ ~13M parameters

$$\phi(\cdot) \in \mathbb{R}^{|\mathcal{V}| \times d}$$
$$W_e \in \mathbb{R}^{d \times d'}$$
$$b_1 \in \mathbb{R}^{d'}$$
$$W_h \in \mathbb{R}^{d' \times d'}$$
$$U \in \mathbb{R}^{d' \times |\mathcal{V}|}$$
$$b_2 \in \mathbb{R}^{|\mathcal{V}|}$$

RNN

```
|V| = 20,000
d = 100
d' = 60
```

➡ ~2.1M parameters

Plus, we can (theoretically) keep information from any point in the past!

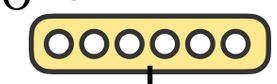
RNN: Scoring Sequences

Input tokens

Word embeddings

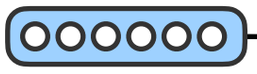
Hidden states

h_0 (learned or null)



$\phi(\texttt{BOS})$

BOS

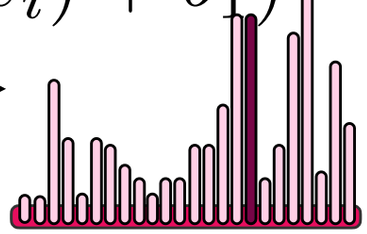


h_1

$h_i = \sigma(W_h h_{i-1} + W_e \phi(x_i) + b_1)$

MLP

softmax



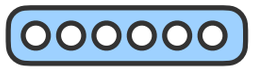
p(I)

$\phi(\mathrm{I})$

I

$\phi(\mathrm{like})$

like

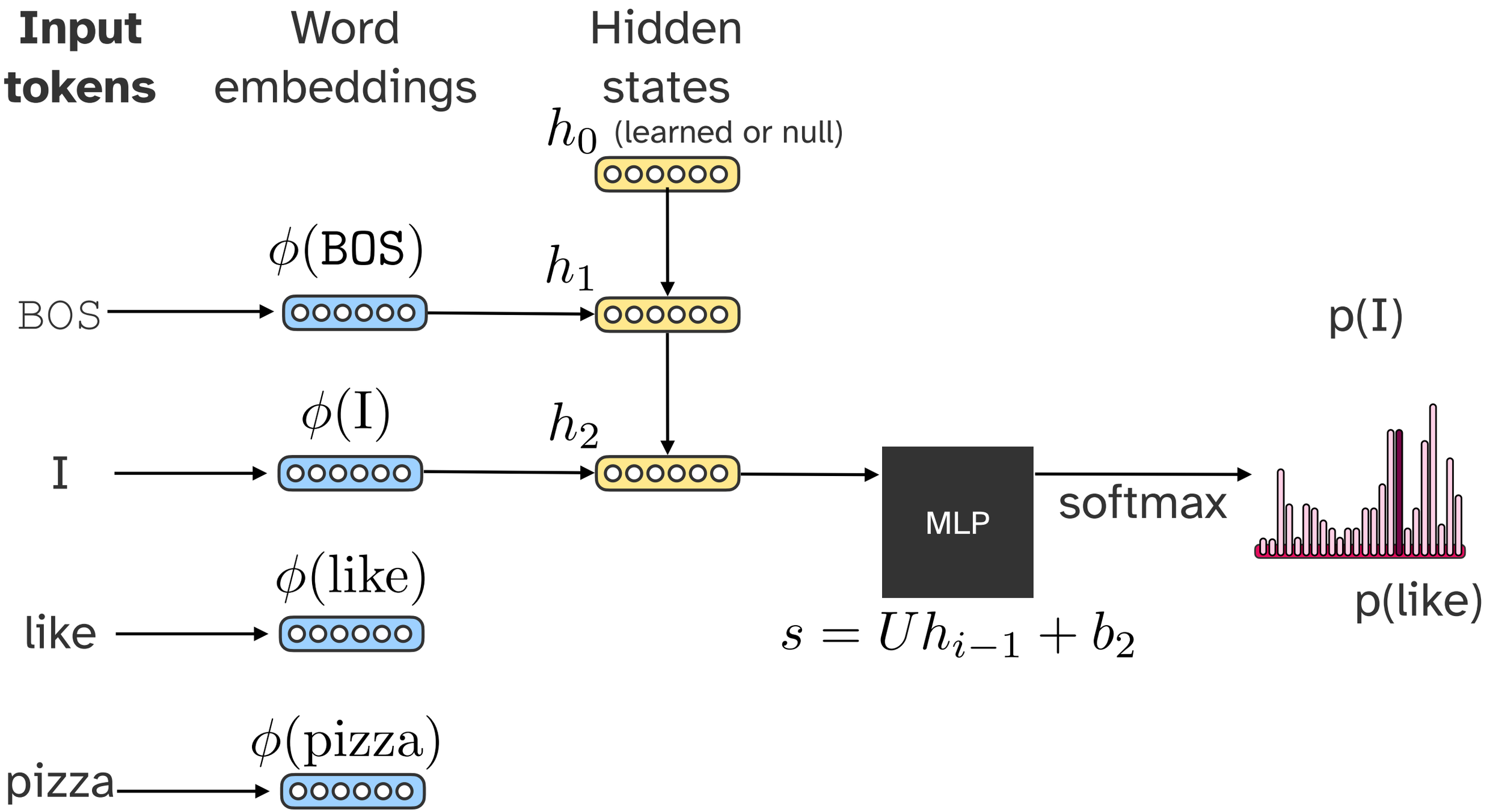


$\phi(\mathrm{pizza})$

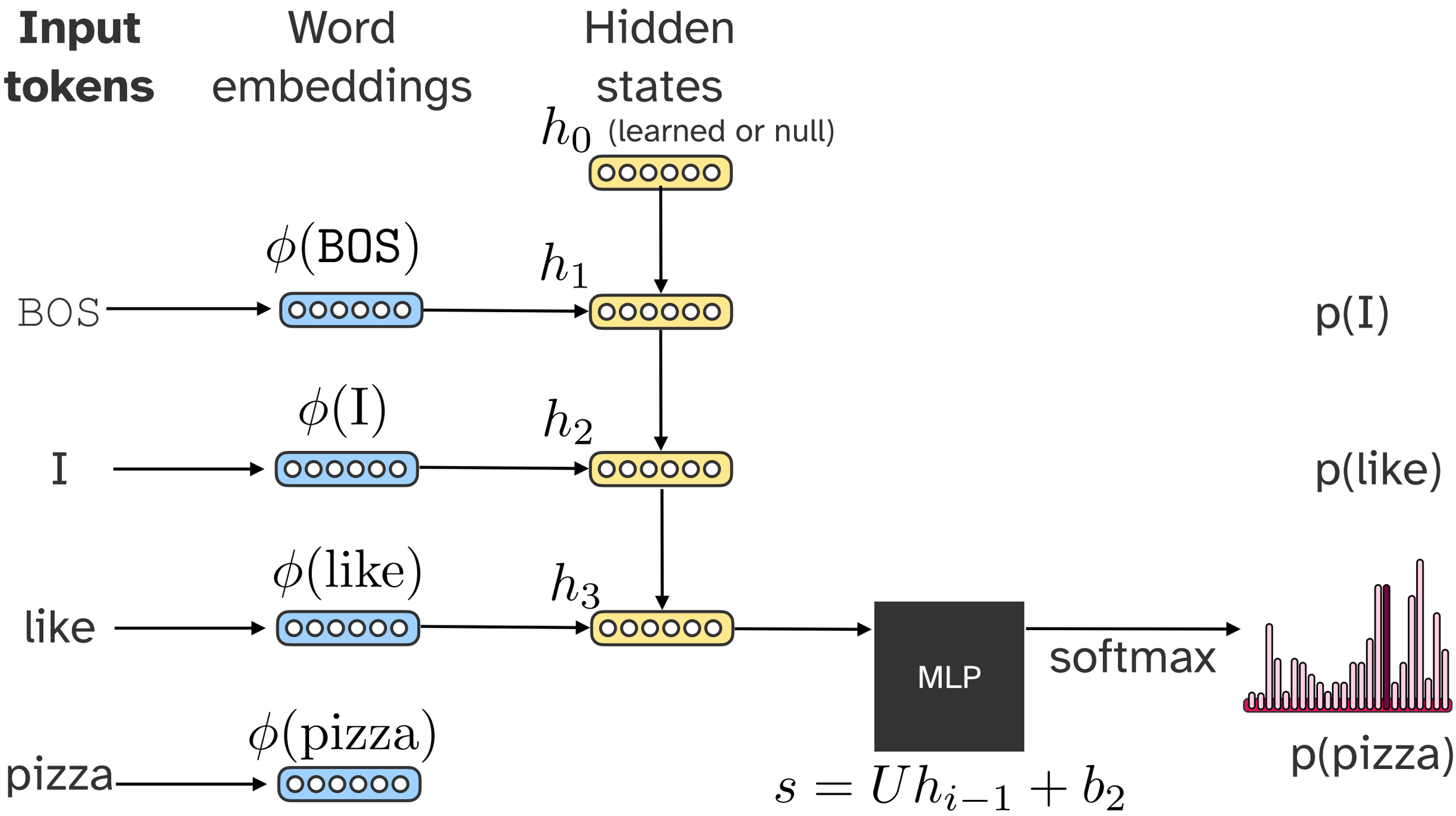
pizza

$s = U h_{i-1} + b_2$

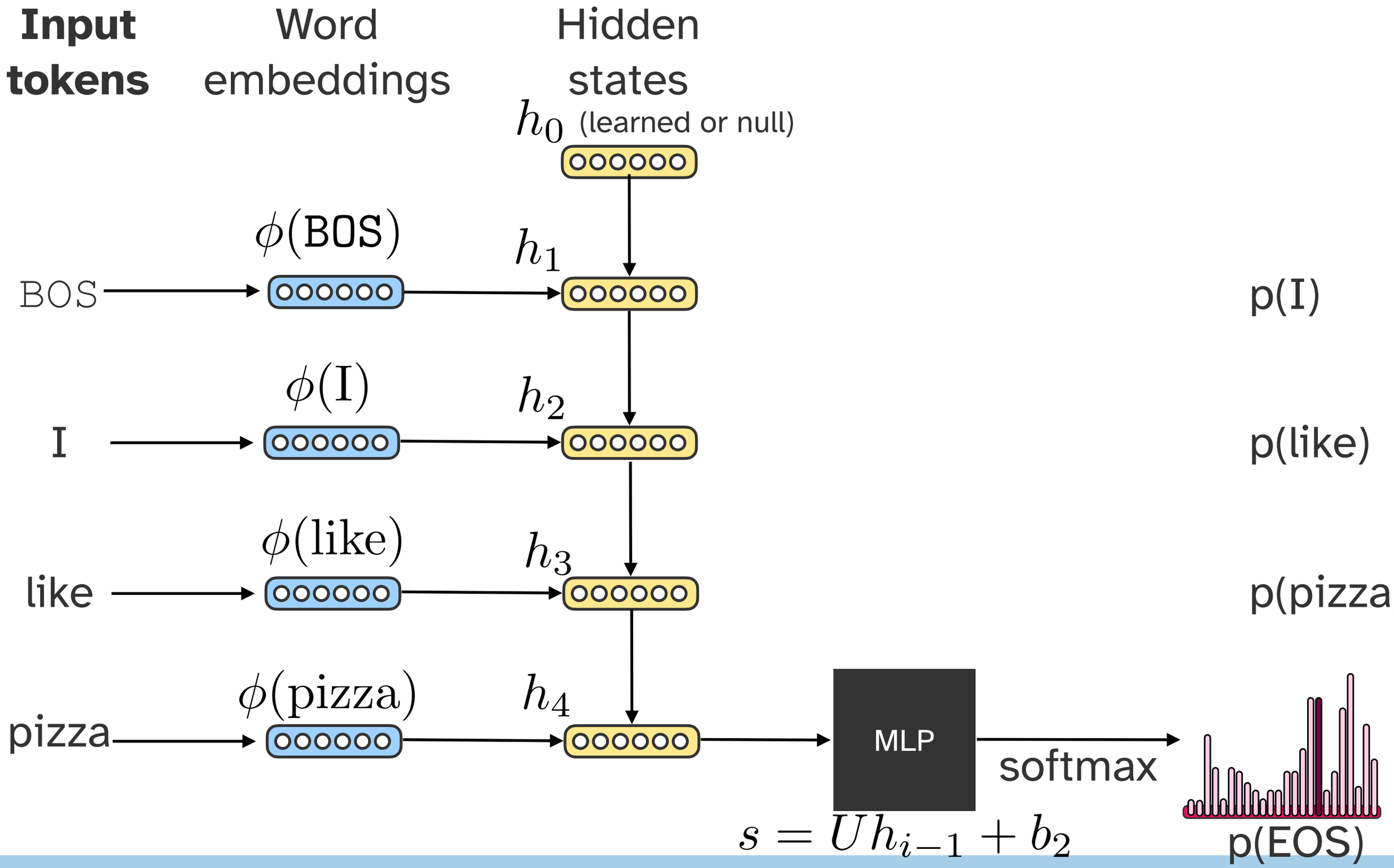
RNN: Scoring Sequences



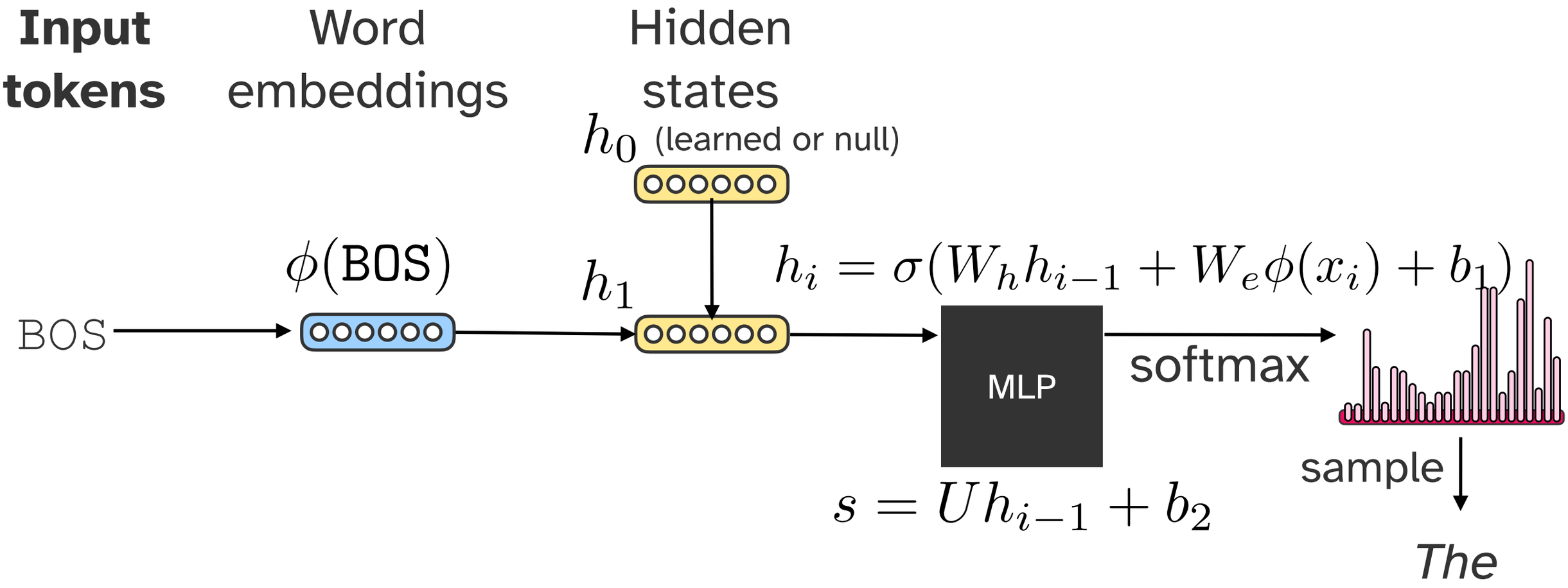
RNN: Scoring Sequences



RNN: Scoring Sequences

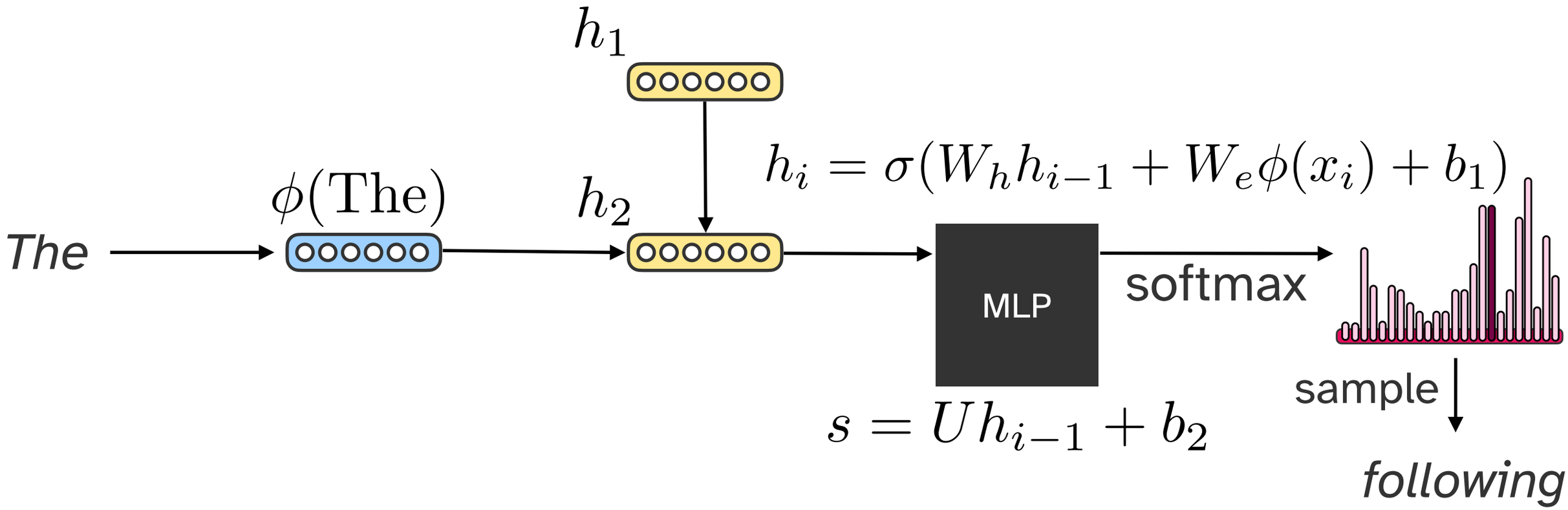


RNN: Generating Sequences



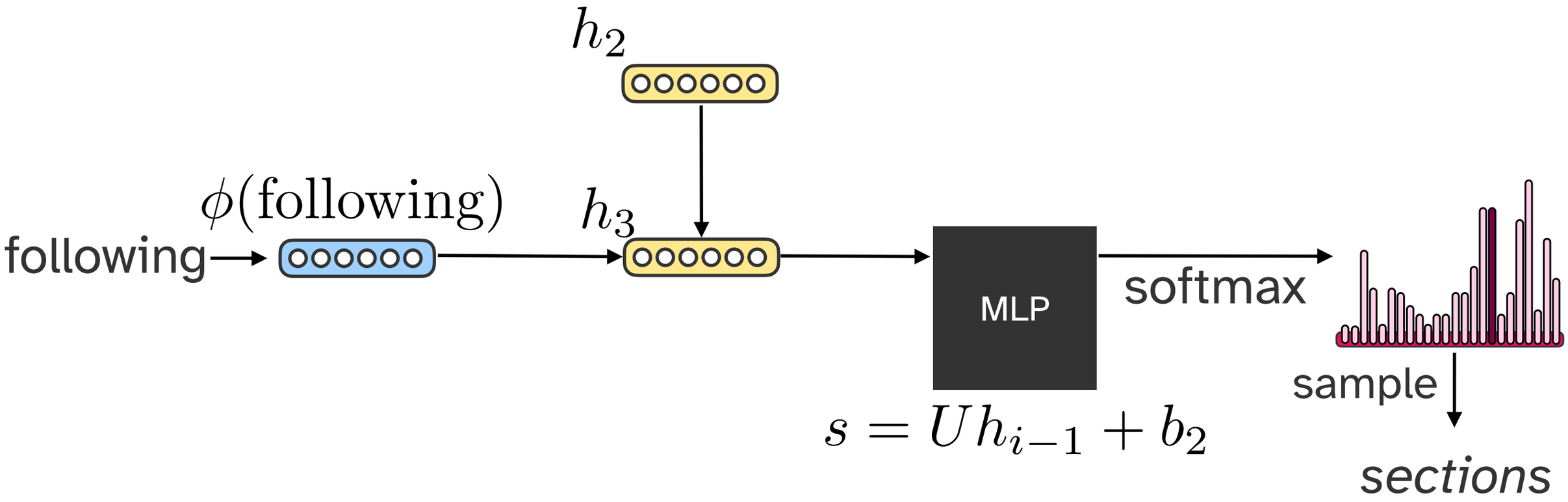
RNN: Generating Sequences

Input tokens | Word embeddings | Hidden states



RNN: Generating Sequences

Input tokens | Word embeddings | Hidden states



Training an RNN

- We have some training dataset $\mathcal{D} = \{\overline{x}\}_{i=1}^{M}$
- We want to find the parameters of our neural model θ that maximize the likelihood of this dataset
- Train using gradient descent in log-probability space

$$\theta^* = \arg\min_{\theta} \left(-\sum_{\overline{x} \in \mathcal{D}} \sum_{i=1}^{|\overline{x}|} \log p(x_i \mid x_1, \ldots, x_{i-1}; \theta) \right)$$

$$\begin{aligned} \phi(\cdot) &\in \mathbb{R}^{|\mathcal{V}| \times d} \\ W_e &\in \mathbb{R}^{d \times d'} \\ b_1 &\in \mathbb{R}^{d'} \\ W_h &\in \mathbb{R}^{d' \times d'} \\ U &\in \mathbb{R}^{d' \times |\mathcal{V}|} \\ b_2 &\in \mathbb{R}^{|\mathcal{V}|} \end{aligned}$$

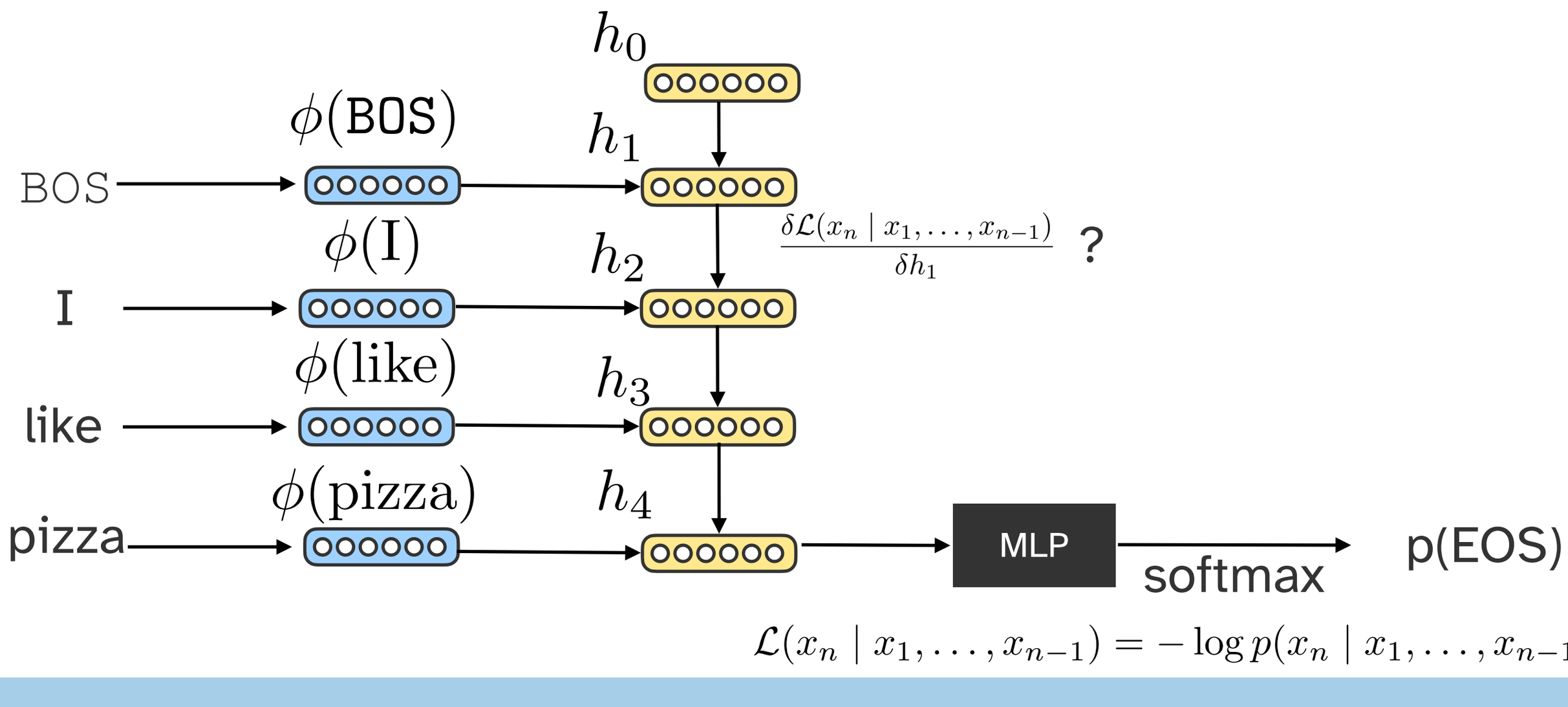
During training, we always compute the probability of each observed token by conditioning on its observed prefix. This is also called “teacher forcing”.

RNNs solve many of our problems!

- ✓ What if we've never seen a sequence before?
- ✓ What if we've never seen its prefix before?
- ✓ Model is tiny compared to count-based or neural n-grams
- ✓ Using word embeddings allows learning from similarity
- ✓ Persistent hidden state makes it robust to intervening words
- ✓ Can theoretically handle long-distance dependencies

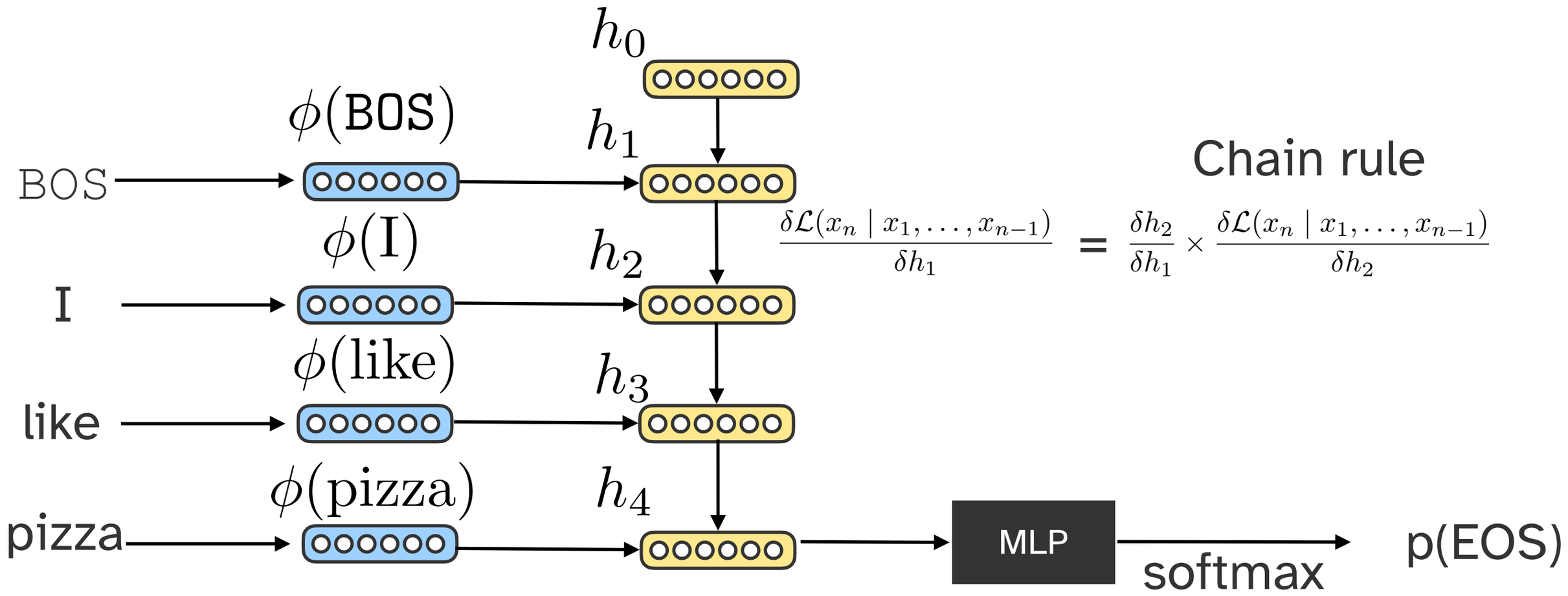
But are they perfect?

- Vanishing gradients



$$\mathcal{L}(x_n \mid x_1, \ldots, x_{n-1}) = -\log p(x_n \mid x_1, \ldots, x_{n-1}; \theta)$$

Vanishing Gradients

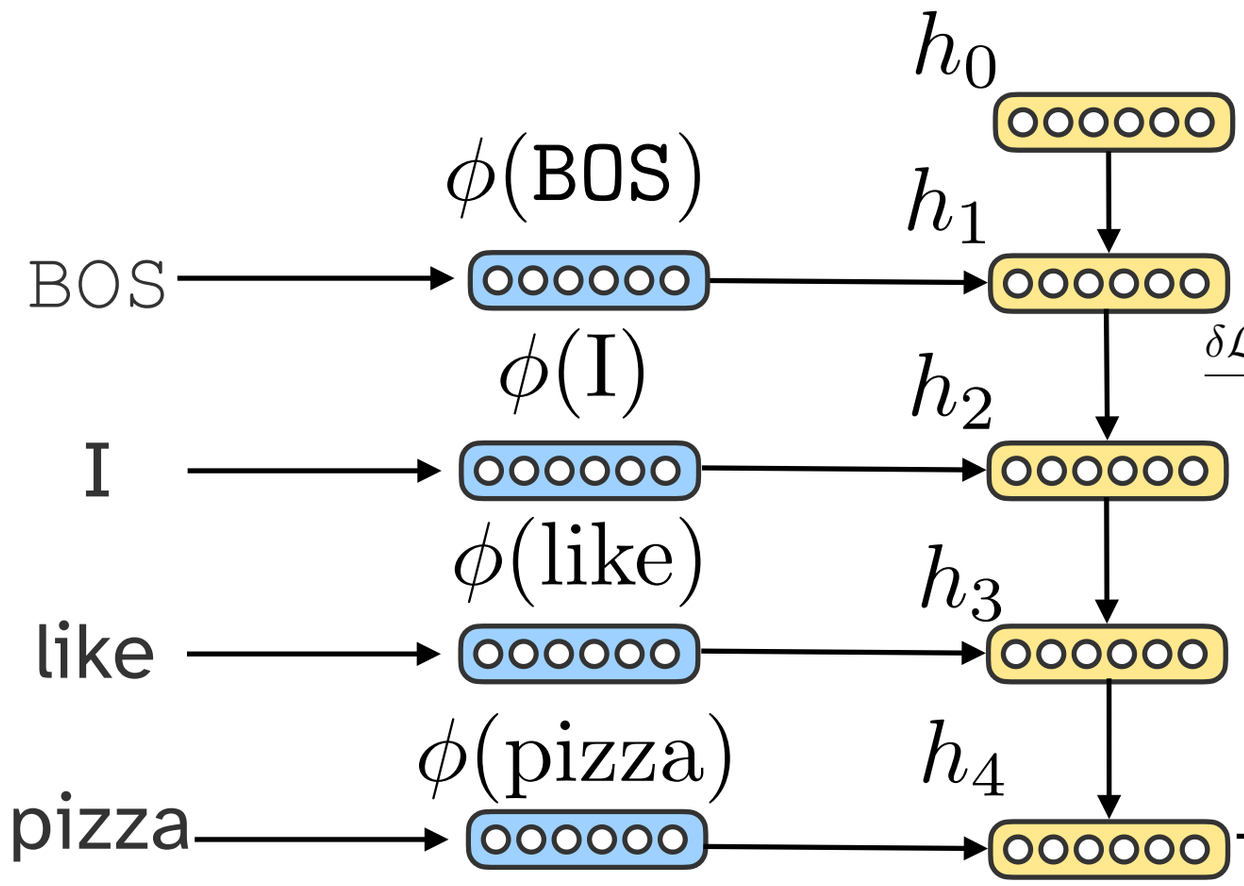


Chain rule

$$\frac{\delta \mathcal{L}(x_n \mid x_1, \ldots, x_{n-1})}{\delta h_1} = \frac{\delta h_2}{\delta h_1} \times \frac{\delta \mathcal{L}(x_n \mid x_1, \ldots, x_{n-1})}{\delta h_2}$$

$$\mathcal{L}(x_n \mid x_1, \ldots, x_{n-1}) = -\log p(x_n \mid x_1, \ldots, x_{n-1}; \theta)$$

Vanishing Gradients

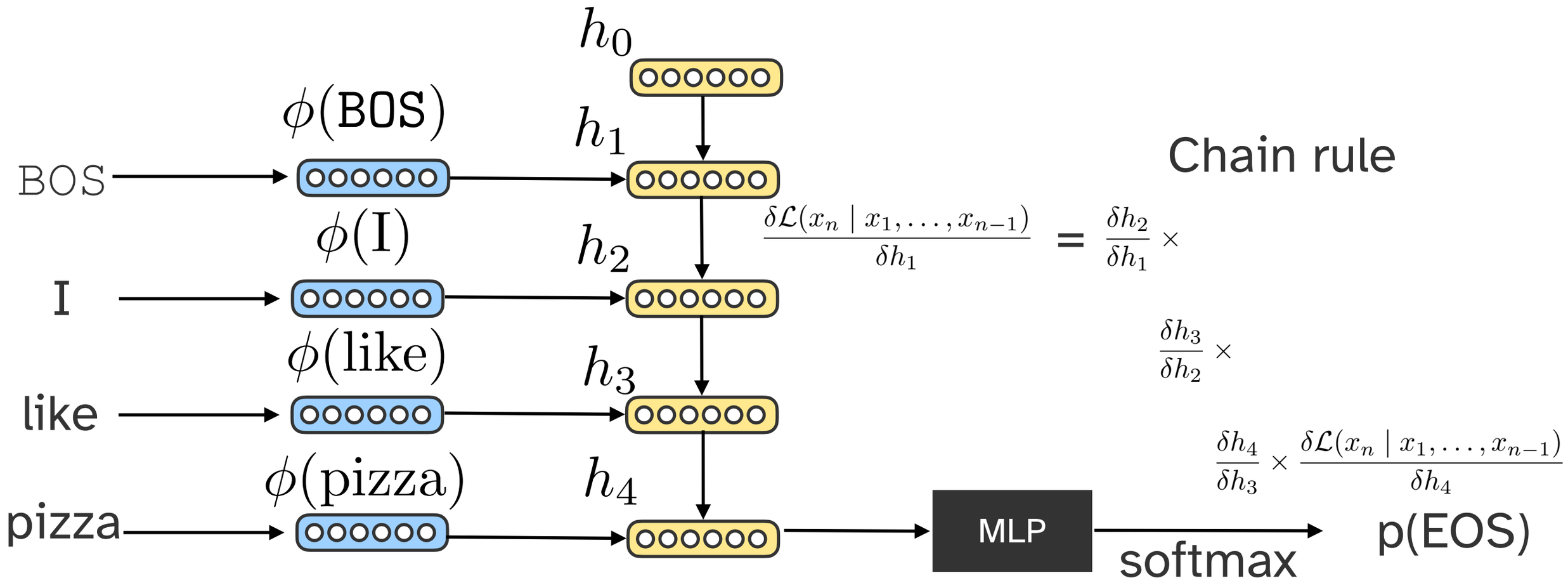


Chain rule

$$\frac{\delta \mathcal{L}(x_n \mid x_1, \ldots, x_{n-1})}{\delta h_1} = \frac{\delta h_2}{\delta h_1} \times \frac{\delta h_3}{\delta h_2} \times \frac{\delta \mathcal{L}(x_n \mid x_1, \ldots, x_{n-1})}{\delta h_3}$$

$$\mathcal{L}(x_n \mid x_1, \ldots, x_{n-1}) = -\log p(x_n \mid x_1, \ldots, x_{n-1}; \theta)$$

Vanishing Gradients



Chain rule

$$\frac{\delta \mathcal{L}(x_n \mid x_1, \ldots, x_{n-1})}{\delta h_1} = \frac{\delta h_2}{\delta h_1} \times \frac{\delta h_3}{\delta h_2} \times \frac{\delta h_4}{\delta h_3} \times \frac{\delta \mathcal{L}(x_n \mid x_1, \ldots, x_{n-1})}{\delta h_4}$$

$$\mathcal{L}(x_n \mid x_1, \ldots, x_{n-1}) = -\log p(x_n \mid x_1, \ldots, x_{n-1}; \theta)$$

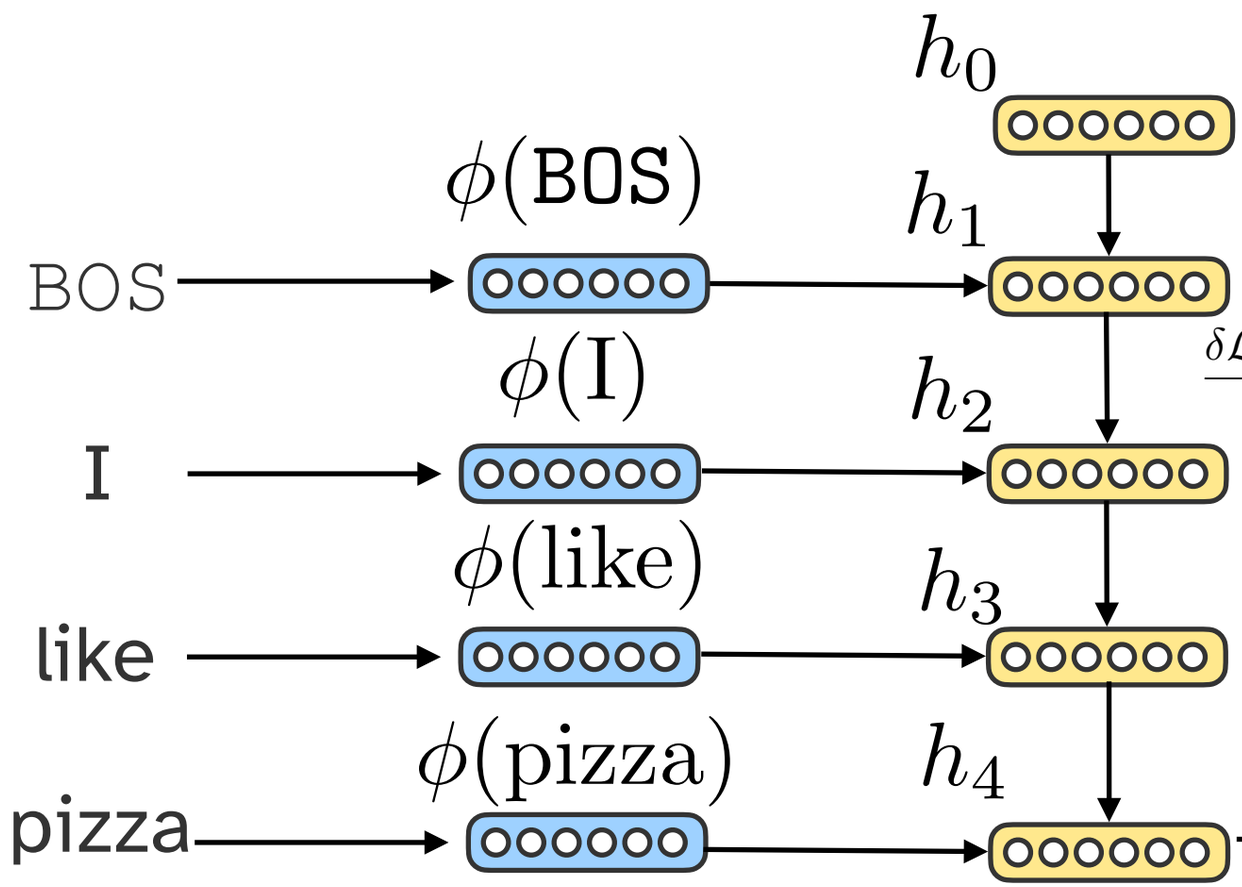
Vanishing Gradients

$$\frac{\delta \mathcal{L}(x_i \mid x_1, \ldots, x_{i-1})}{\delta h_j} = \frac{\delta \mathcal{L}(x_i \mid x_1, \ldots, x_{i-1})}{\delta h_i} \prod_{j<t\leq i} \frac{\delta h_t}{\delta h_{t-1}}$$

$$h_i = \sigma(W_h h_{i-1} + W_e \phi(x_i) + b_1)$$

$$f(h_{i-1}) = W_h h_{i-1} + W_e \phi(x_i) + b_1$$

$$\frac{\delta h_i}{\delta h_{i-1}} = \operatorname{diag}\left(\sigma'(f(h_{i-1}))\right) W_h$$



$$\frac{\delta \mathcal{L}(x_n \mid x_1, \ldots, x_{n-1})}{\delta h_1} = \frac{\delta h_2}{\delta h_1} \times \frac{\delta h_3}{\delta h_2} \times \frac{\delta h_4}{\delta h_3} \times \frac{\delta \mathcal{L}(x_n \mid x_1, \ldots, x_{n-1})}{\delta h_4}$$

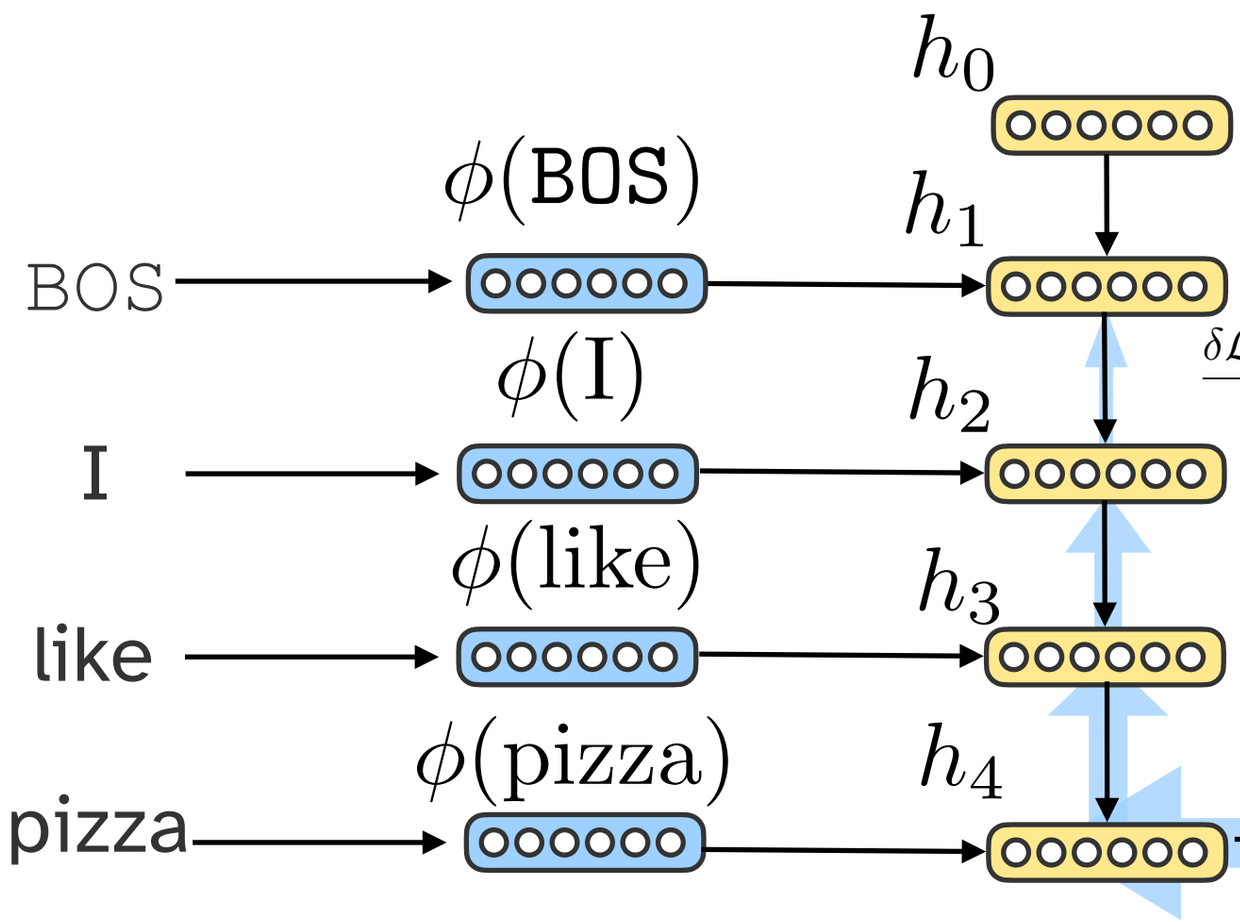
$$\mathcal{L}(x_n \mid x_1, \ldots, x_{n-1}) = -\log p(x_n \mid x_1, \ldots, x_{n-1}; \theta)$$

Vanishing Gradients

$$\frac{\delta \mathcal{L}(x_i \mid x_1, \ldots, x_{i-1})}{\delta h_j} = \frac{\delta \mathcal{L}(x_i \mid x_1, \ldots, x_{i-1})}{\delta h_i} \prod_{j<t\leq i} \text{diag}\left(\sigma'(f(h_{i-1}))\right) W_h$$

$$\prod_{j<t\leq i} W_h \prod_{j<t\leq i} \text{diag}(\sigma'(f(h_{i-1}))$$

$$W_h^{i-j} \prod_{j<t\leq i} \text{diag}(\sigma'(f(h_{i-1}))$$



$$\frac{\delta \mathcal{L}(x_n \mid x_1, \ldots, x_{n-1})}{\delta h_1} = \frac{\delta h_2}{\delta h_1} \times \frac{\delta h_3}{\delta h_2} \times \frac{\delta h_4}{\delta h_3} \times \frac{\delta \mathcal{L}(x_n \mid x_1, \ldots, x_{n-1})}{\delta h_4}$$

If the values of the weight matrix are too small, gradients will get exponentially small

$$\mathcal{L}(x_n \mid x_1, \ldots, x_{n-1}) = -\log p(x_n \mid x_1, \ldots, x_{n-1}; \theta)$$

RNN Problems

- **Vanishing gradients:** gradient signal from many timesteps in the future is negligible compared to gradient signal from nearby tokens

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 …

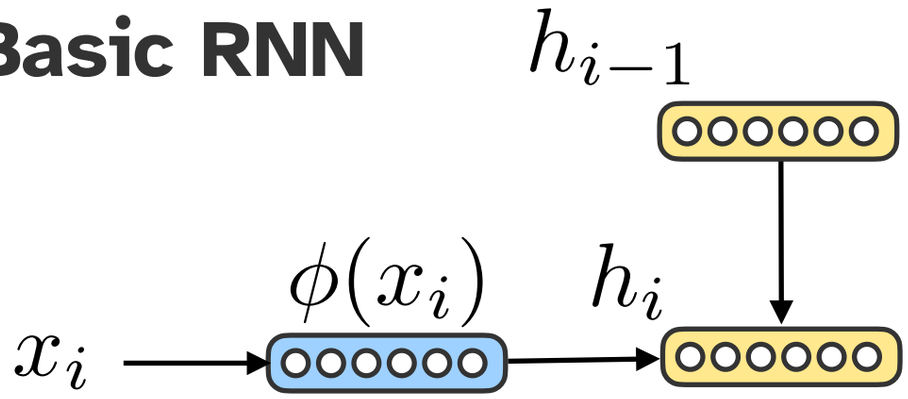
Example from Stanford cs224n

RNN Problems

- **Vanishing gradients:** gradient signal from many timesteps in the future is negligible compared to gradient signal from nearby tokens
- **Exploding gradients:** if the activations are too big, the gradients become too big, and the values of the weights become too big, eventually getting values closer to ∞
    - Can solve using gradient clipping (if the norm of the gradient is greater than a threshold, clamp its value or scale it down)

Long Short-Term Memory

Basic RNN

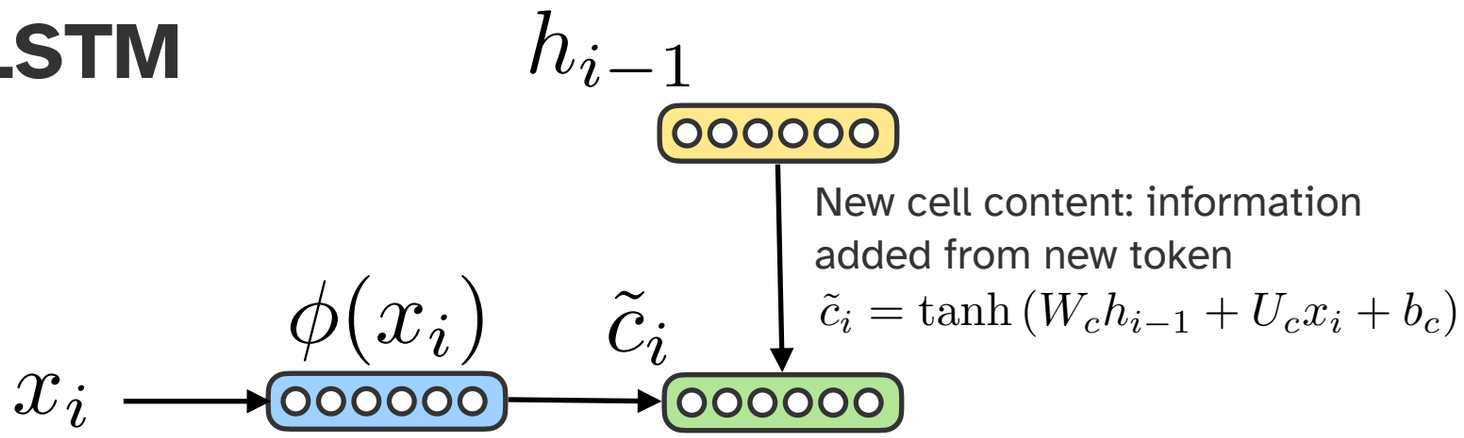


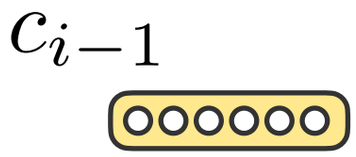
$$h_i = \sigma(W_h h_{i-1} + W_e \phi(x_i) + b_1)$$

Hidden state is updated the same way at every step. The update doesn't depend on the new information coming from x_i

Long Short-Term Memory

LSTM

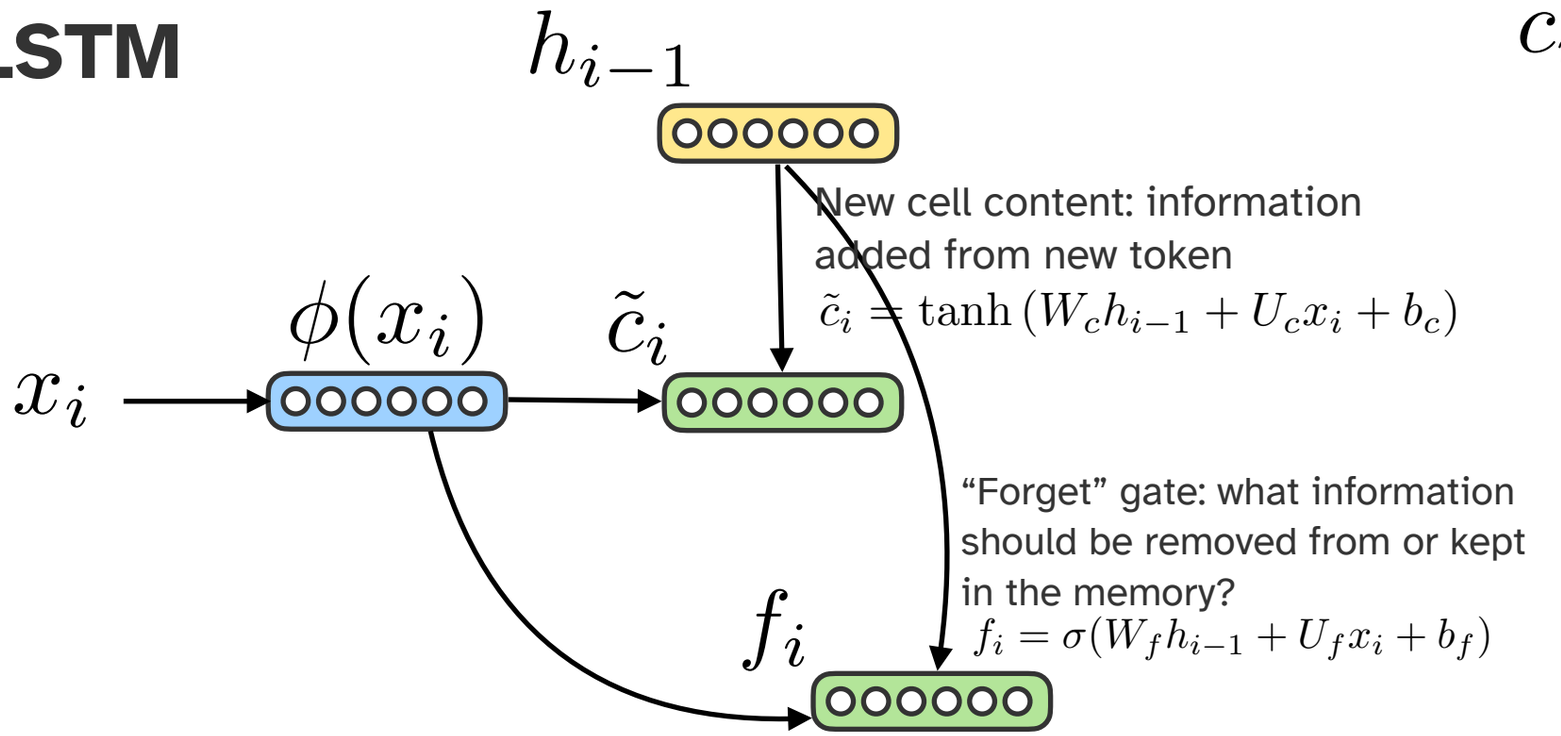




Cell memory: same size as hidden state

Long Short-Term Memory

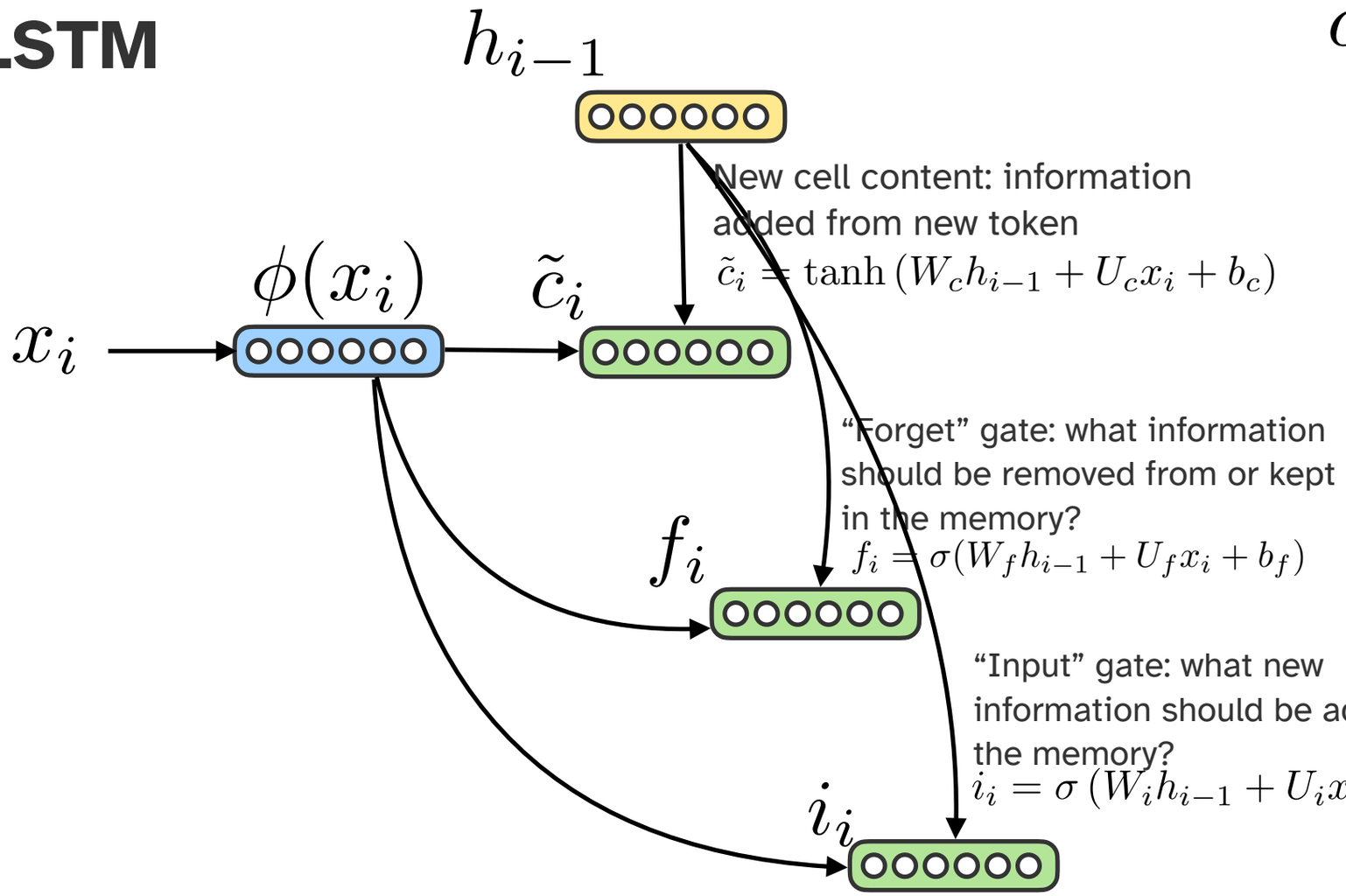
LSTM



Cell memory: same size as hidden state

Long Short-Term Memory

LSTM

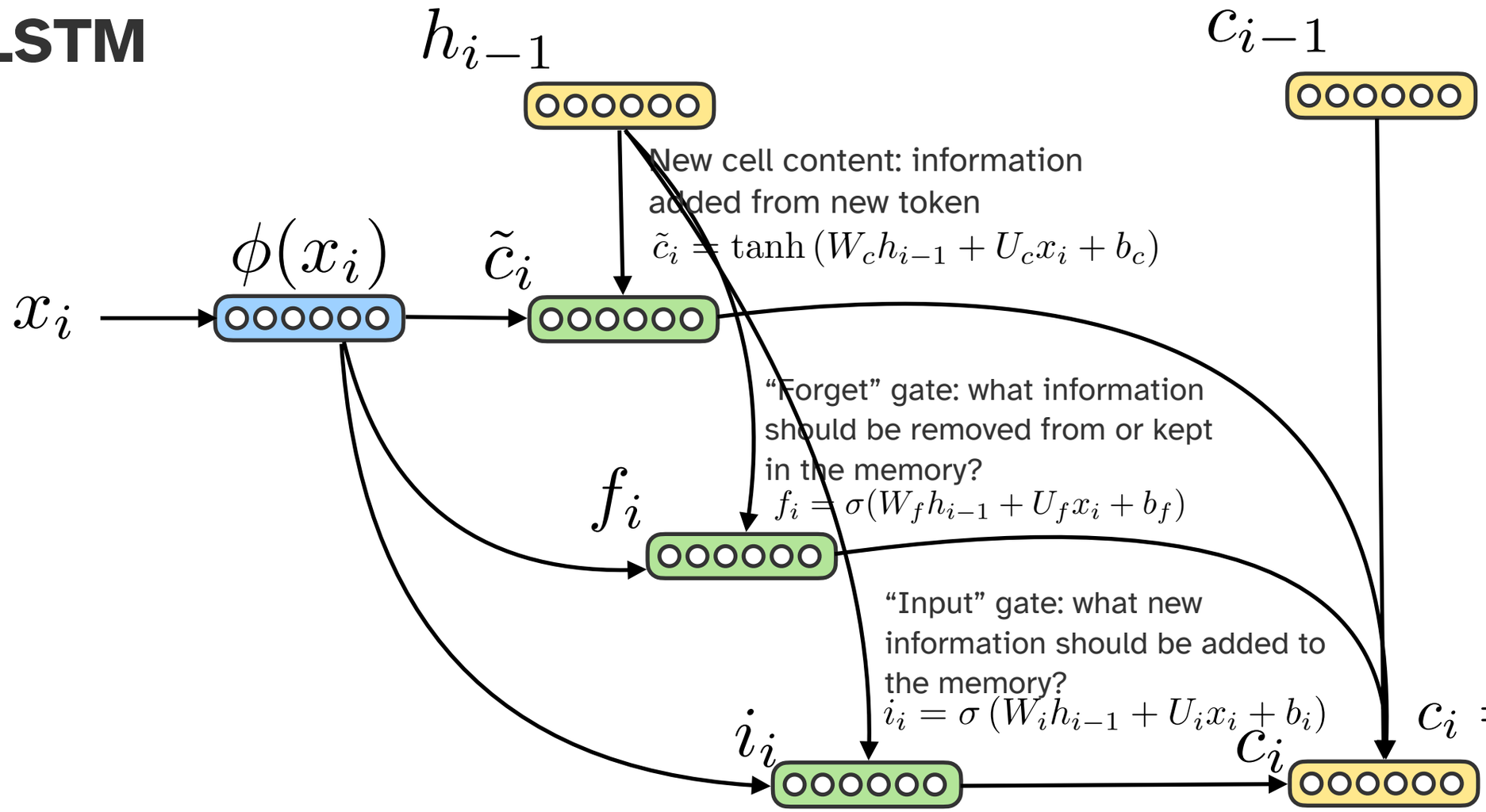


c_{i-1}

Cell memory: same size as hidden state

Long Short-Term Memory

LSTM



Cell memory: same size as hidden state

New information to add

$$c_i = i_i \circ \tilde{c}_i + f_i \circ c_{i-1}$$

Information to forget or keep

Long Short-Term Memory

LSTM

h_{i-1}

c_{i-1}

Cell memory: same size as hidden state

New cell content: information added from new token

$\tilde{c}_i = \tanh\left(W_c h_{i-1} + U_c x_i + b_c\right)$

$\phi(x_i)$

x_i

$\tilde{c}_i$

"Forget" gate: what information should be removed from or kept in the memory?

$f_i = \sigma(W_f h_{i-1} + U_f x_i + b_f)$

f_i

"Input" gate: what new information should be added to the memory?

$i_i = \sigma\left(W_i h_{i-1} + U_i x_i + b_i\right)$

i_i

c_i

$c_i = i_i \circ \tilde{c}_i + f_i \circ c_{i-1}$

New information to add

Information to forget or keep

"Output" gate: what information from the memory should be put in the new hidden state?

$o_i = \sigma(W_o h_{i-1} + U_o x_i + b_o)$

o_i

New hidden state: get some information from "long-term" memory to serve as the "working memory" for this timestep

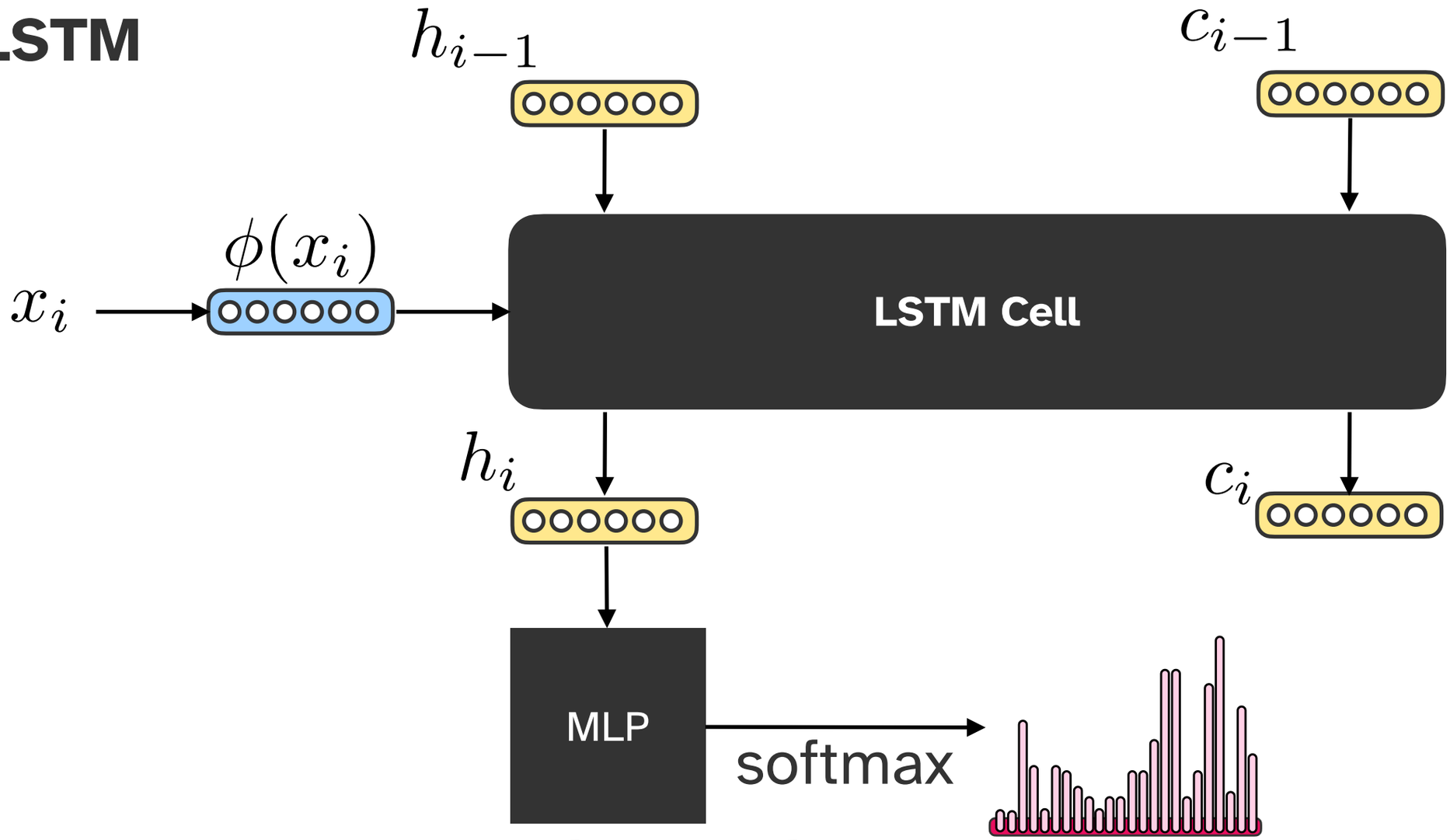
$h_i = o_i \circ \tanh(c_i)$

h_i

Vanishing gradients: no longer a problem, because we can keep around information indefinitely

Long Short-Term Memory

LSTM



Cell memory: same size as hidden state

$$s = Uh_{i-1} + b_2$$