DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 2: Recurrent Neural Networks (RNNs)

Caio Corro

LECTURE 1 RECALL

Language modeling with a multi-layer perceptron

2nd order Markov chain: $p(y_1, \ldots, y_n) = \quad p(y_1) \quad p(y_2 \mid y_1) \quad \prod_{i=3}^{n} p(y_i \mid y_{i-1}, y_{i-2})$

$$\mathbf{x} = \begin{bmatrix} \textit{Embedding of } y_{i-1} \\ \textit{Embedding of } y_{i-2} \end{bmatrix} \quad \mathbf{z} = \sigma\left(\mathbf{U}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) \quad \mathbf{w} = \mathbf{U}^{(2)}\mathbf{z} + \mathbf{b}^{(2)} \quad p(y_i \mid y_{i-1}, y_{i-2}) = \frac{\exp(\mathbf{w}_{y_i})}{\sum_{y'} \exp(\mathbf{w}_{y'})}$$

Concatenate the embeddings of the two previous words

Hidden representation

Output projection

Probability distribution

LECTURE 1 RECALL

Language modeling with a multi-layer perceptron

2nd order Markov chain: $p(y_1, \ldots, y_n) = \quad p(y_1) \quad p(y_2 \,|\, y_1) \quad \prod_{i=3}^{n} p(y_i \,|\, y_{i-1}, y_{i-2})$

$$\mathbf{x} = \begin{bmatrix} \textit{Embedding of } y_{i-1} \\ \textit{Embedding of } y_{i-2} \end{bmatrix} \quad \mathbf{z} = \sigma\left(\mathbf{U}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) \quad \mathbf{w} = \mathbf{U}^{(2)}\mathbf{z} + \mathbf{b}^{(2)} \quad p(y_i \,|\, y_{i-1}, y_{i-2}) = \frac{\exp(\mathbf{w}_{y_i})}{\sum_{y'} \exp(\mathbf{w}_{y'})}$$

Sentence classification with a Convolutional Neural Network

1. Convolution: sliding window of fixed size of the input sentence
2. Mean/max pooling over convolution outputs
3. Multi-linear perceptron

LECTURE 1 RECALL

Language modeling with a multi-layer perceptron

2nd order Markov chain: $p(y_1, \ldots, y_n) = \quad p(y_1) \quad p(y_2 \mid y_1) \quad \prod_{i=3}^{n} p(y_i \mid y_{i-1}, y_{i-2})$

$$\mathbf{x} = \begin{bmatrix} \textit{Embedding of } y_{i-1} \\ \textit{Embedding of } y_{i-2} \end{bmatrix} \quad \mathbf{z} = \sigma\left(\mathbf{U}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) \quad \mathbf{w} = \mathbf{U}^{(2)}\mathbf{z} + \mathbf{b}^{(2)} \quad p(y_i \mid y_{i-1}, y_{i-2}) = \frac{\exp(\mathbf{w}_{y_i})}{\sum_{y'} \exp(\mathbf{w}_{y'})}$$

Sentence classification with a Convolutional Neural Network

1. Convolution: sliding window of fixed size of the input sentence
2. Mean/max pooling over convolution outputs
3. Multi-linear perceptron

Main issue

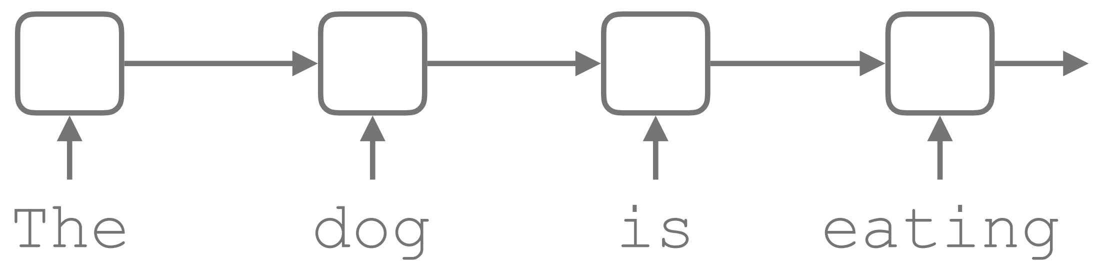
- These 2 networks only use local word-order information
- No long range dependencies

LONG RANGE DEPENDENCIES

Today

Recurrent neural networks

- Inputs are fed sequentially
- State representation updated at each input

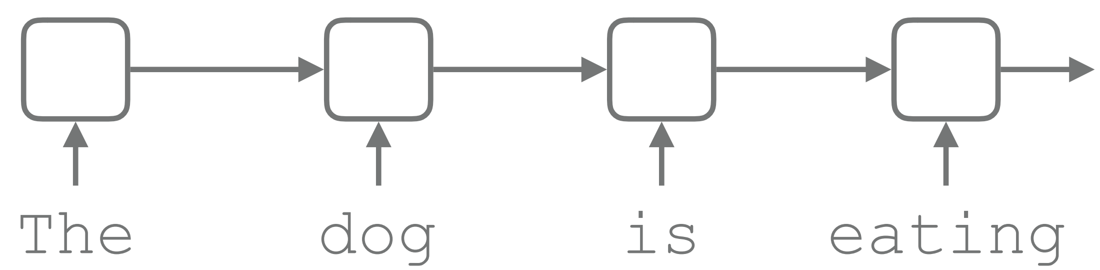


LONG RANGE DEPENDENCIES

Today

Recurrent neural networks

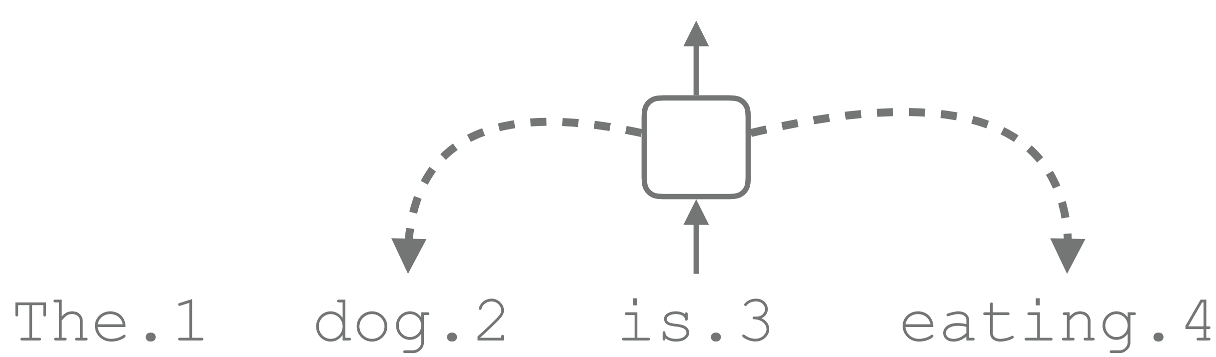
- Inputs are fed sequentially
- State representation updated at each input



Next time!

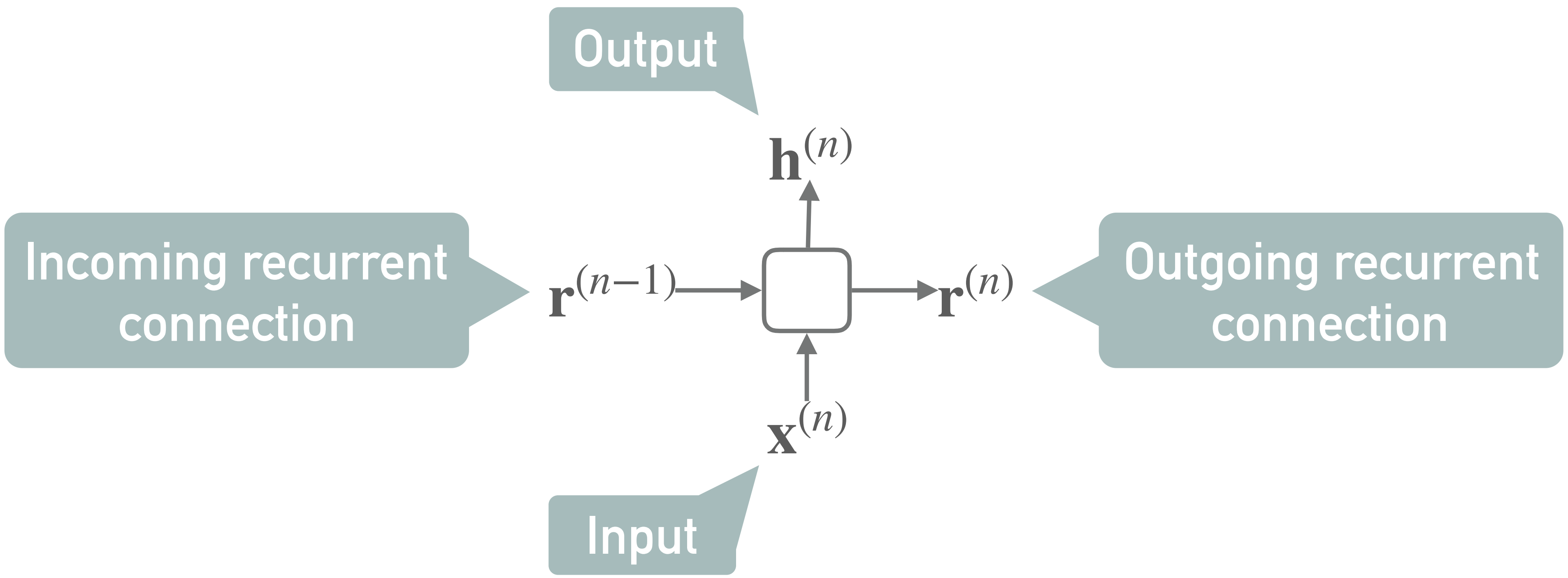
Attention network

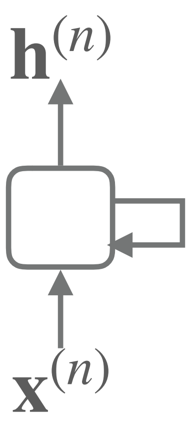
- Inputs contain position information
- At each position look at any input in the sentence



RECURRENT NEURAL NETWORK

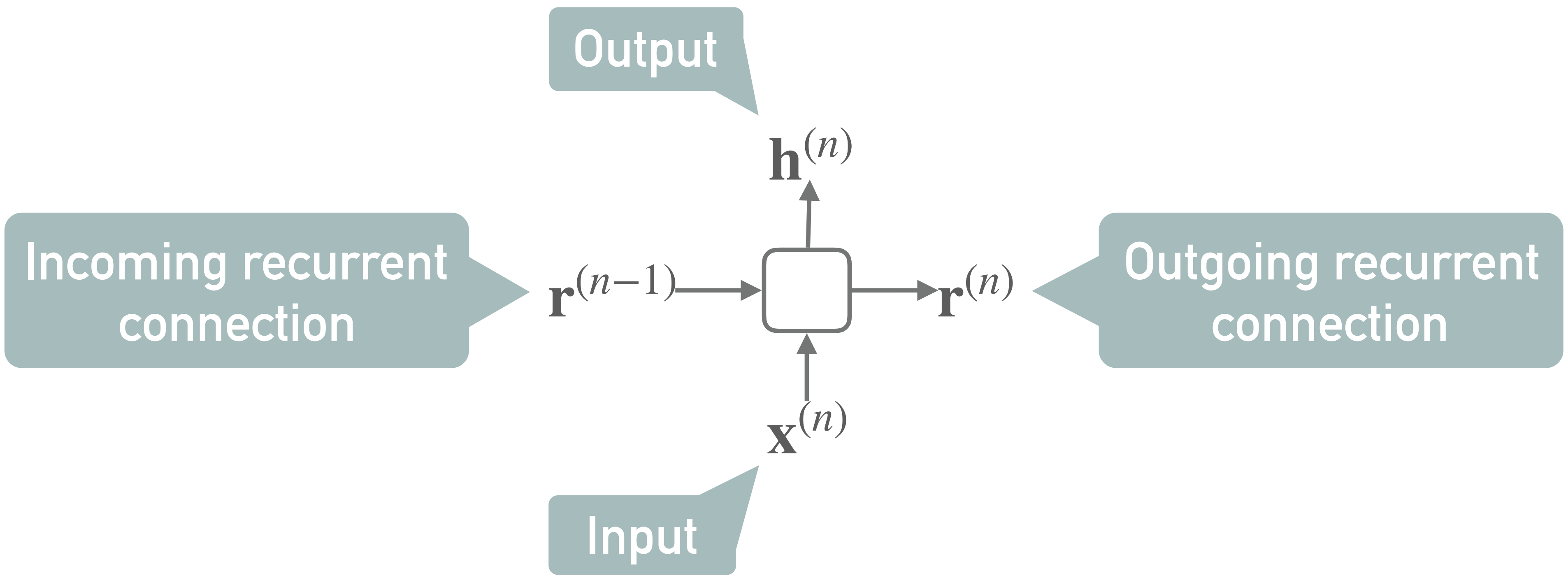
Recurrent neural network cell

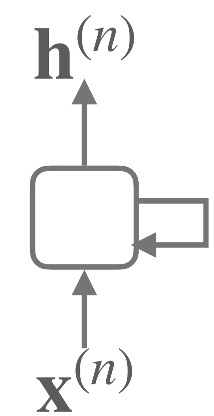




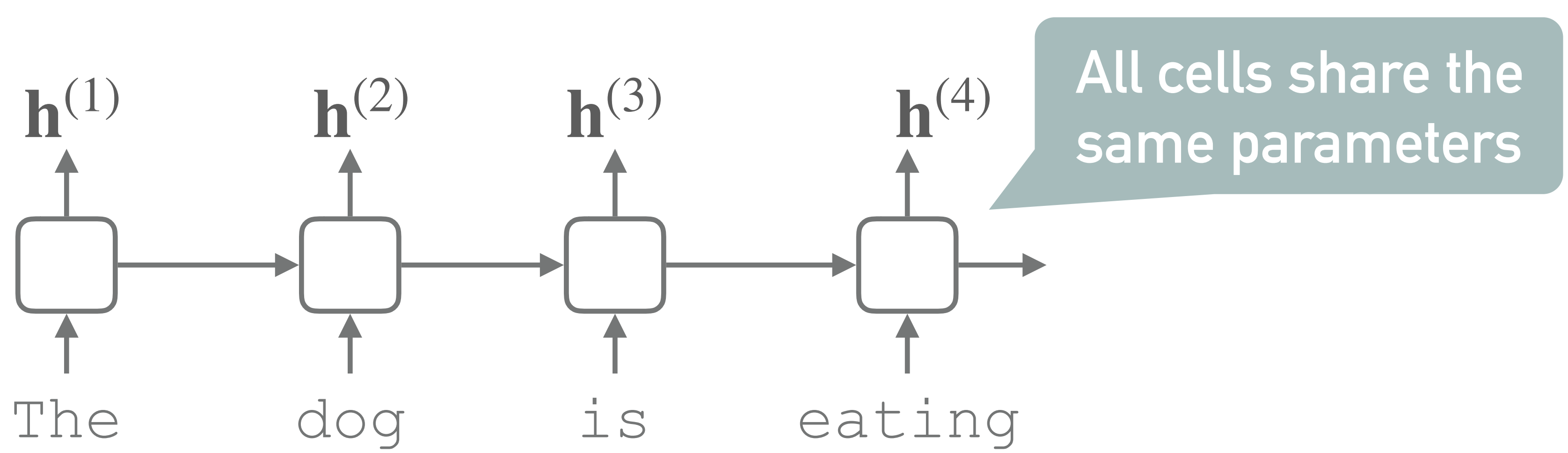
RECURRENT NEURAL NETWORK

Recurrent neural network cell





Dynamic neural network



LANGUAGE MODEL

Why do we usually make independence assumptions?

- Less parameters to learn
- Less sparsity

Non neural language model

$|V| \times |V|$ *parameters*

- 1st order Markov chain: $p(y_1, \ldots, y_n) = \quad p(y_1) \quad \prod_{i=2}^{n} p(y_i \,|\, y_{i-1})$
- 2nd order Markov chain: $p(y_1, \ldots, y_n) = \quad p(y_1) \quad p(y_2 \,|\, y_1) \quad \prod_{i=3}^{n} p(y_i \,|\, y_{i-1}, y_{i-2})$

$|V| \times |V| \times |V|$ *parameters*

Multi-layer perceptron language model

- No sparsity issue thanks to word embeddings
- Independence assumption, so no long range dependencies

LANGUAGE MODEL WITH RECURRENT NEURAL NETWORKS

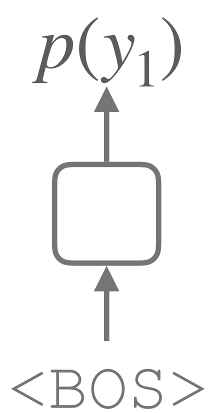
$$p(y_1 \ldots y_n) = p(y_1, \ldots, y_{n-1}) p(y_n \,|\, y_1, \ldots, y_{n-1})$$

No independence assumption!

LANGUAGE MODEL WITH RECURRENT NEURAL NETWORKS

$$p(y_1 \ldots y_n) = p(y_1, \ldots, y_{n-1}) p(y_n \mid y_1, \ldots, y_{n-1})$$

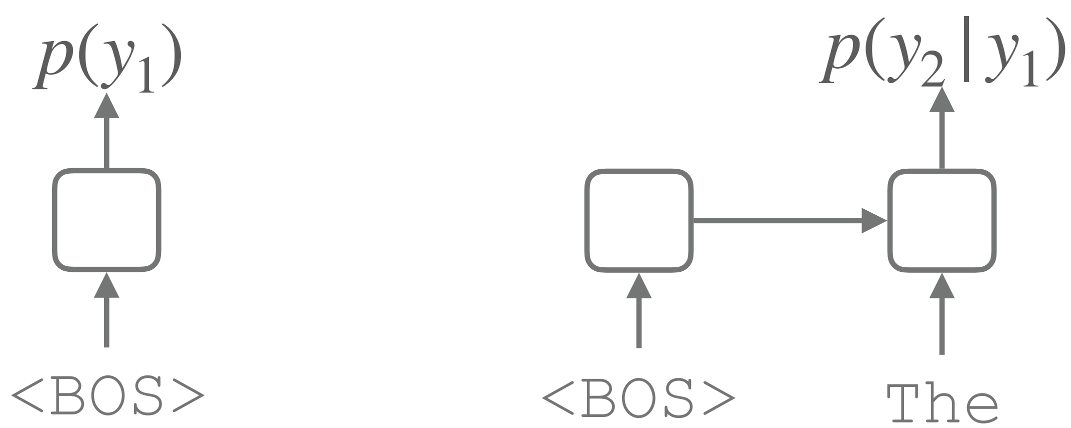
No independence assumption!



LANGUAGE MODEL WITH RECURRENT NEURAL NETWORKS

$$p(y_1 \ldots y_n) = p(y_1, \ldots, y_{n-1})p(y_n \mid y_1, \ldots, y_{n-1})$$

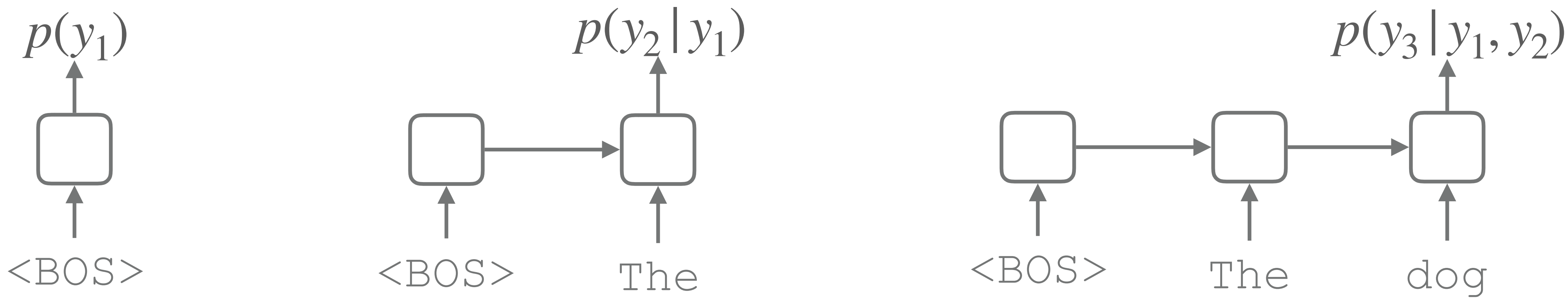
No independence assumption!



LANGUAGE MODEL WITH RECURRENT NEURAL NETWORKS

$$p(y_1 \ldots y_n) = p(y_1, \ldots, y_{n-1}) p(y_n \mid y_1, \ldots, y_{n-1})$$

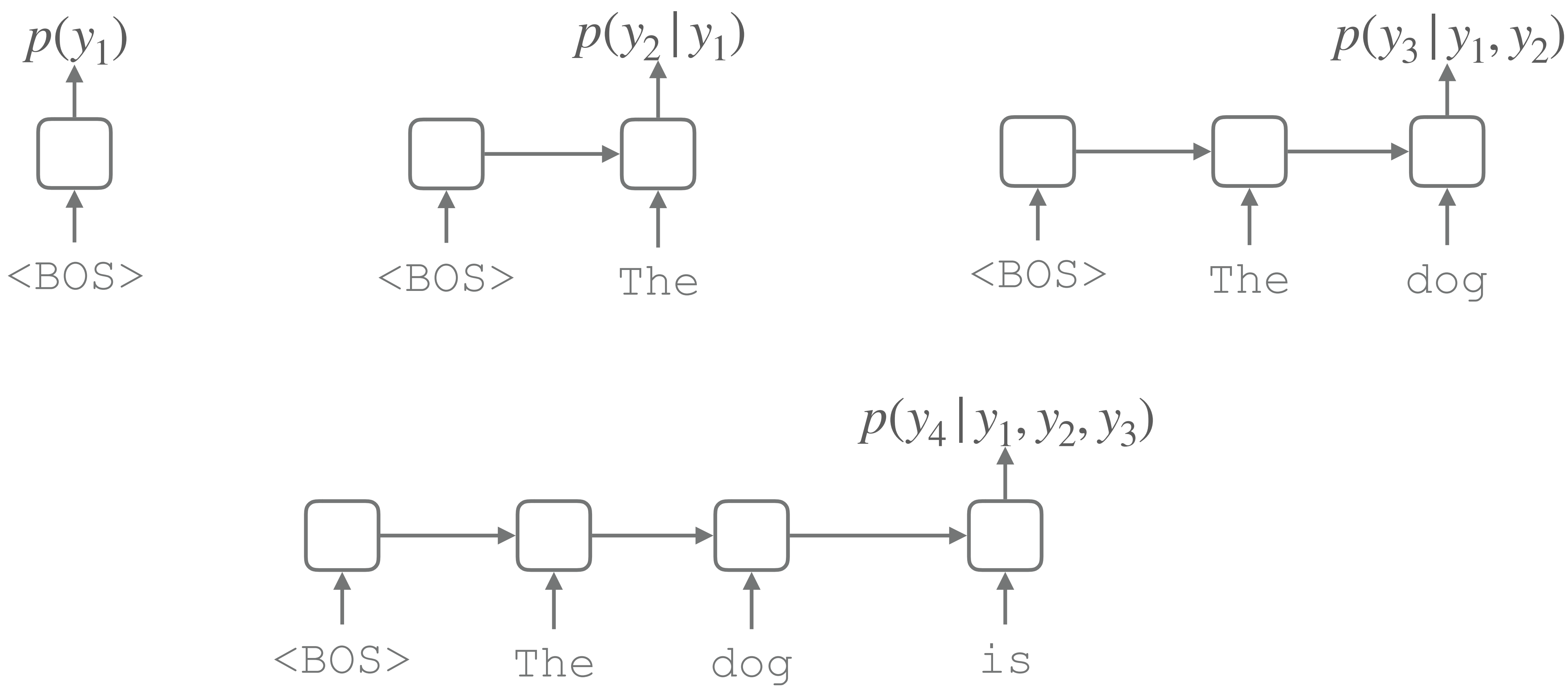
No independence assumption!



LANGUAGE MODEL WITH RECURRENT NEURAL NETWORKS

$$p(y_1 \ldots y_n) = p(y_1, \ldots, y_{n-1}) p(y_n \,|\, y_1, \ldots, y_{n-1})$$

No independence assumption!



SENTENCE CLASSIFICATION

Neural architecture

1. A recurrent neural network (RNN) compute a context sensitive representation of the sentence

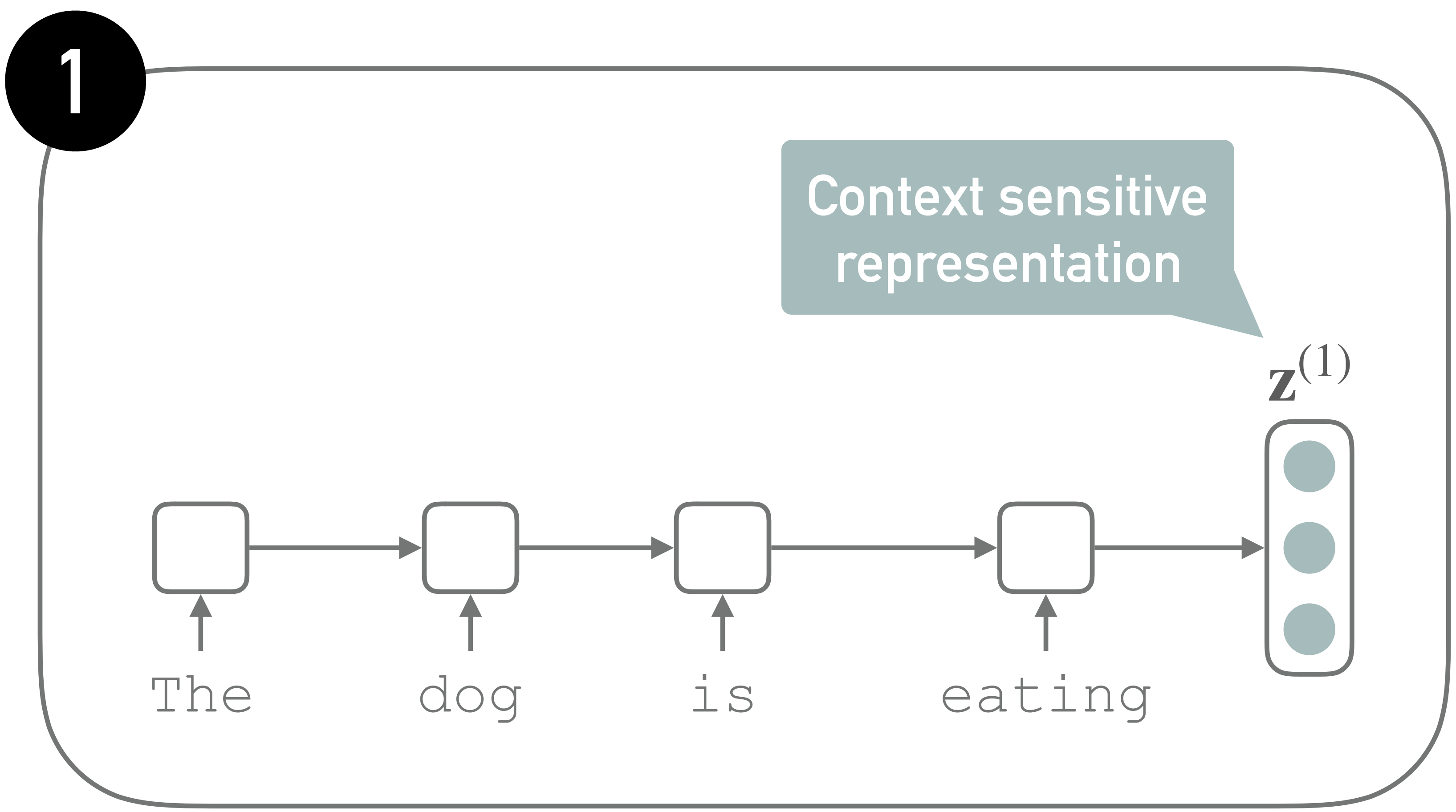
2. A multi-layer perceptron takes as input this representation and output class weights

SENTENCE CLASSIFICATION

Neural architecture

1. A recurrent neural network (RNN) compute a context sensitive representation of the sentence

2. A multi-layer perceptron takes as input this representation and output class weights

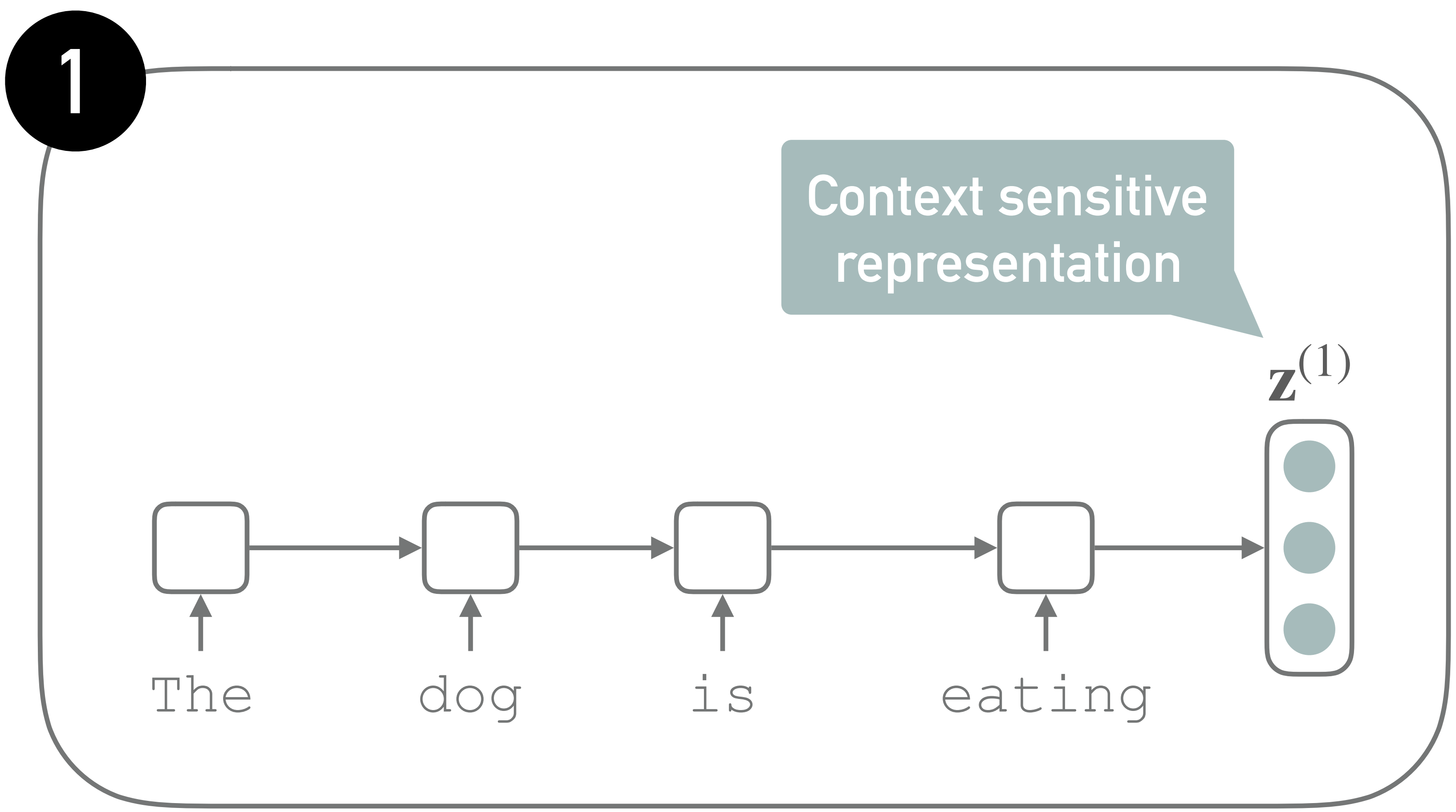


SENTENCE CLASSIFICATION

Neural architecture

1. A recurrent neural network (RNN) compute a context sensitive representation of the sentence

2. A multi-layer perceptron takes as input this representation and output class weights



2

MLP hidden layer

$$\mathbf{z}^{(2)} = \sigma\left(\mathbf{U}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}\right)$$

$$\mathbf{w} = \mathbf{U}^{(2)}\mathbf{z}^{(2)} + \mathbf{b}^{(2)}$$

Output weights

MACHINE TRANSLATION

Neural architecture: Encoder-Decoder

1. **Encoder:** a recurrent neural network (RNN) compute a context sensitive representation of the sentence

2. **Decoder:** a different recurrent neural network (RNN) compute the translation, word after word

Conditional language model

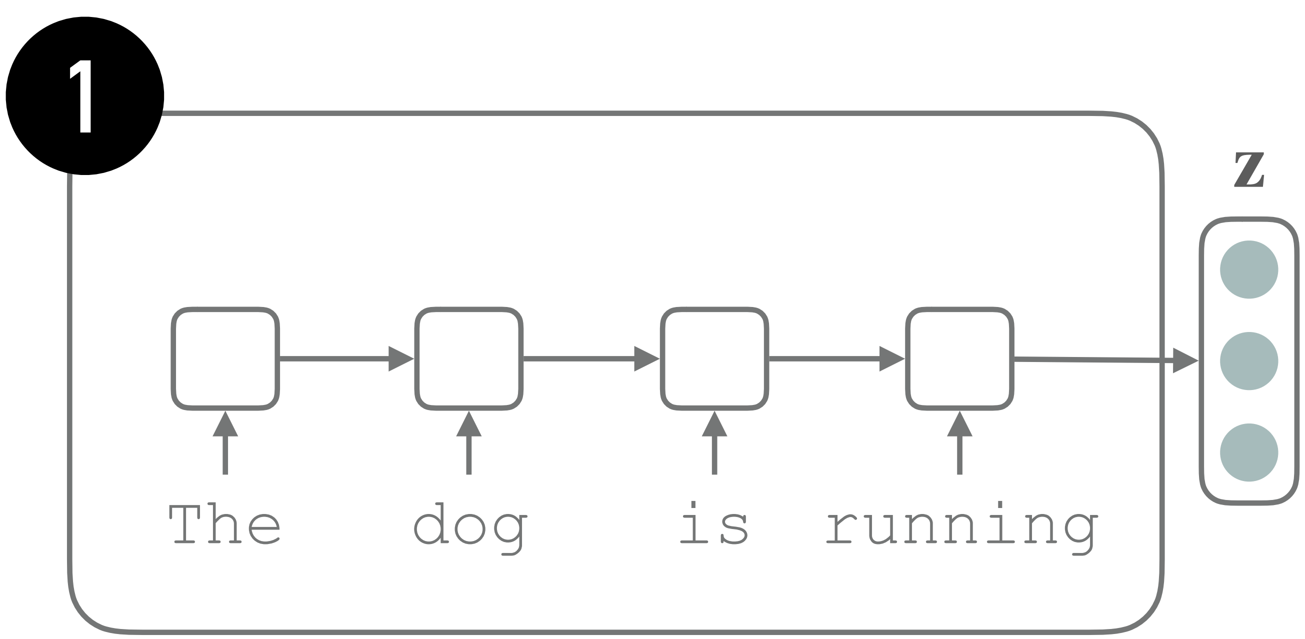
MACHINE TRANSLATION

Neural architecture: Encoder-Decoder

1. **Encoder:** a recurrent neural network (RNN) compute a context sensitive representation of the sentence

2. **Decoder:** a different recurrent neural network (RNN) compute the translation, word after word

Conditional language model



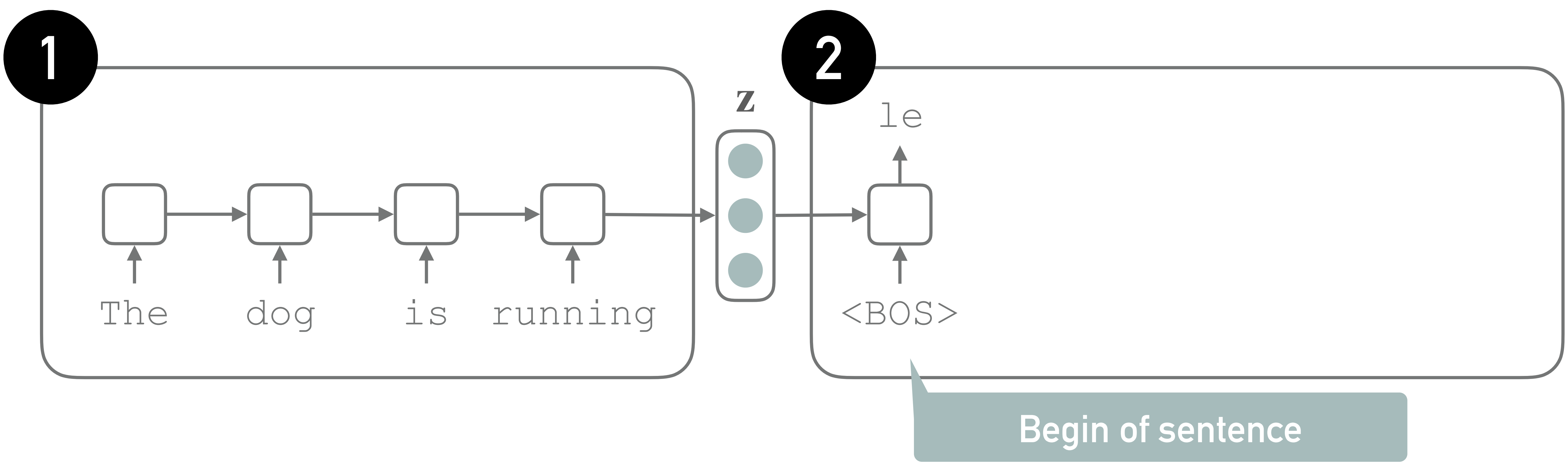
MACHINE TRANSLATION

Neural architecture: Encoder-Decoder

1. **Encoder:** a recurrent neural network (RNN) compute a context sensitive representation of the sentence

2. **Decoder:** a different recurrent neural network (RNN) compute the translation, word after word

Conditional language model



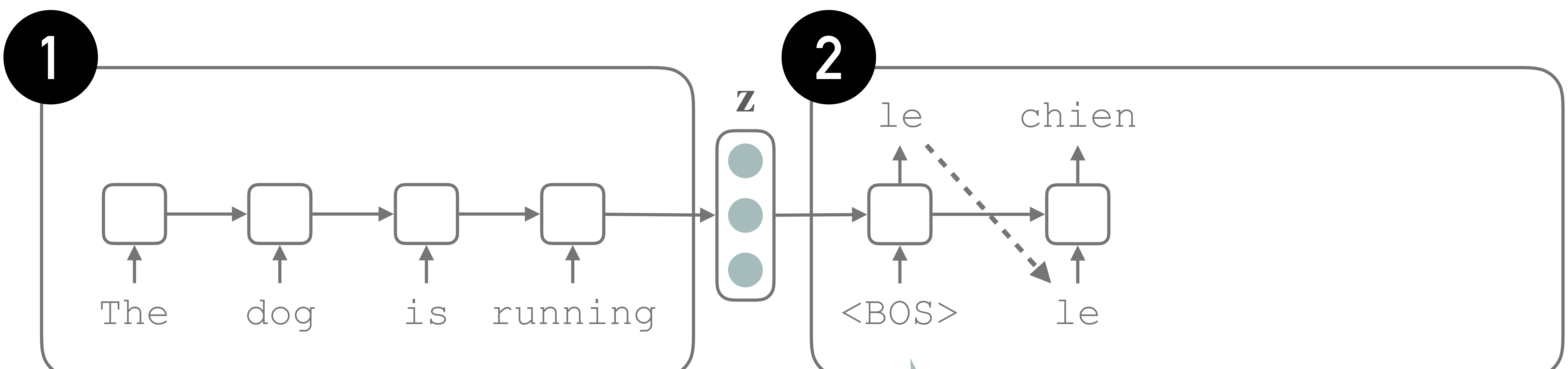
MACHINE TRANSLATION

Neural architecture: Encoder-Decoder

1. **Encoder:** a recurrent neural network (RNN) compute a context sensitive representation of the sentence

2. **Decoder:** a different recurrent neural network (RNN) compute the translation, word after word

Conditional language model



Begin of sentence

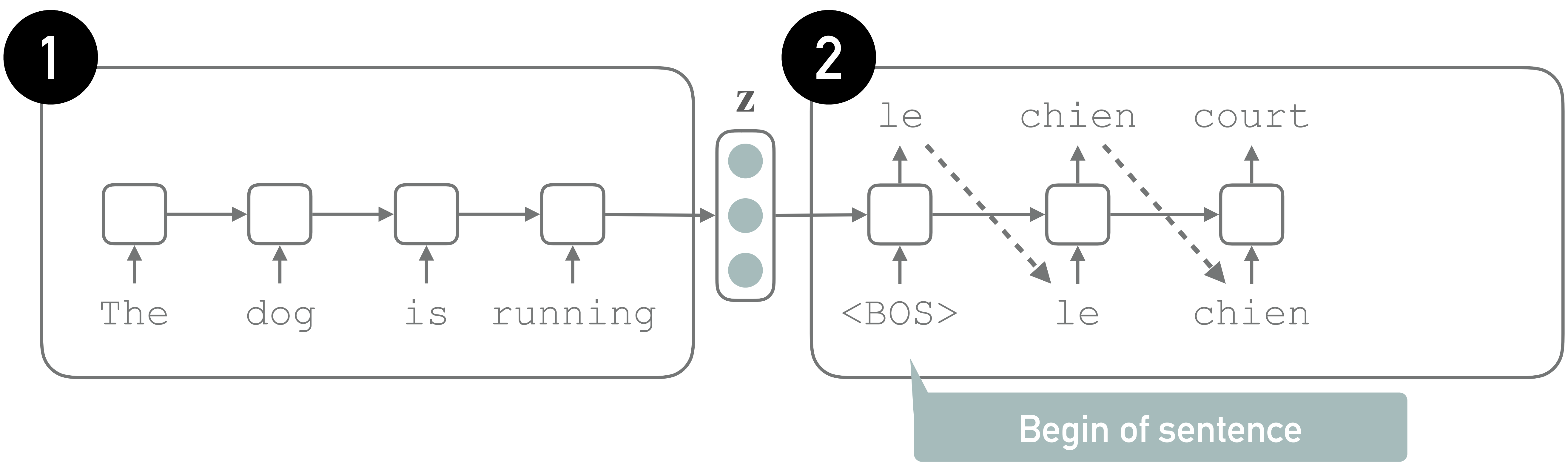
MACHINE TRANSLATION

Neural architecture: Encoder-Decoder

1. **Encoder:** a recurrent neural network (RNN) compute a context sensitive representation of the sentence

2. **Decoder:** a different recurrent neural network (RNN) compute the translation, word after word

Conditional language model



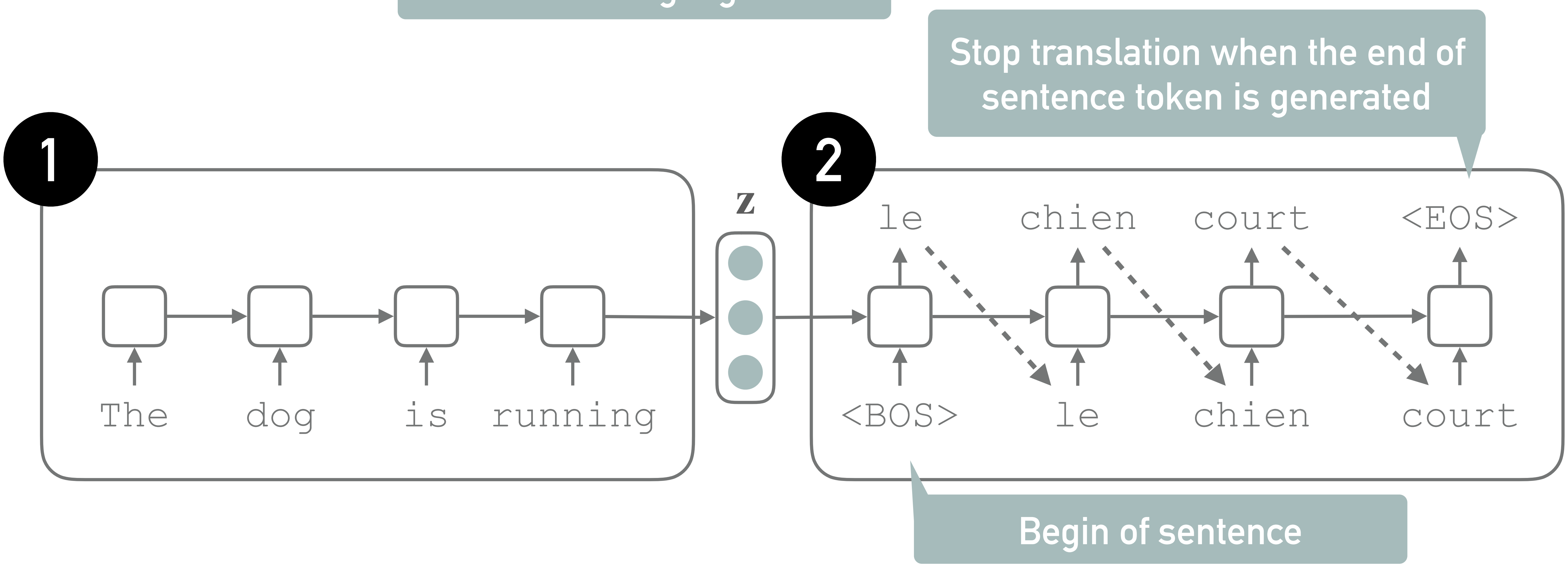
MACHINE TRANSLATION

Neural architecture: Encoder-Decoder

1. **Encoder:** a recurrent neural network (RNN) compute a context sensitive representation of the sentence

2. **Decoder:** a different recurrent neural network (RNN) compute the translation, word after word

Conditional language model



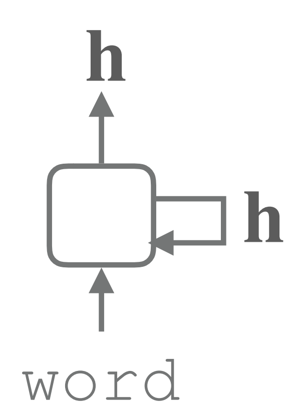
SIMPLE RECURRENT NEURAL NETWORK

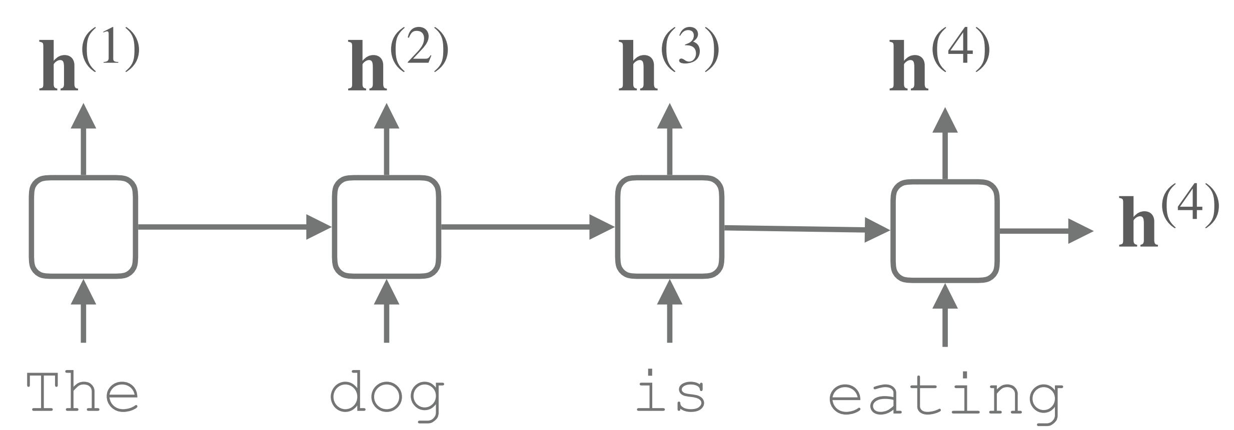
MULTI-LAYER PERCEPTRON RECURRENT NETWORK

Multi-linear perceptron cell

- Input: the current word and the previous output
- Output: the hidden representation

The recurrent connection is juste the output at each position



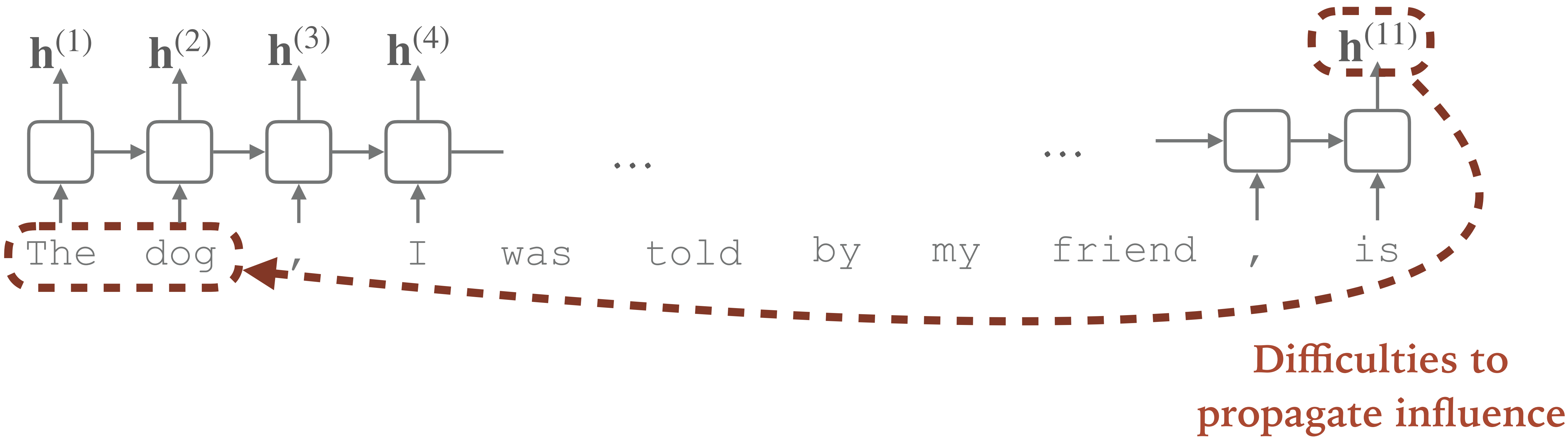


$$\mathbf{h}^{(n)} = \tanh\left(\mathbf{U}\begin{bmatrix}\mathbf{x}^{(n)}\\ \mathbf{h}^{(n-1)}\end{bmatrix} + \mathbf{b}\right)$$

GRADIENT BASED LEARNING PROBLEM

Does it work?

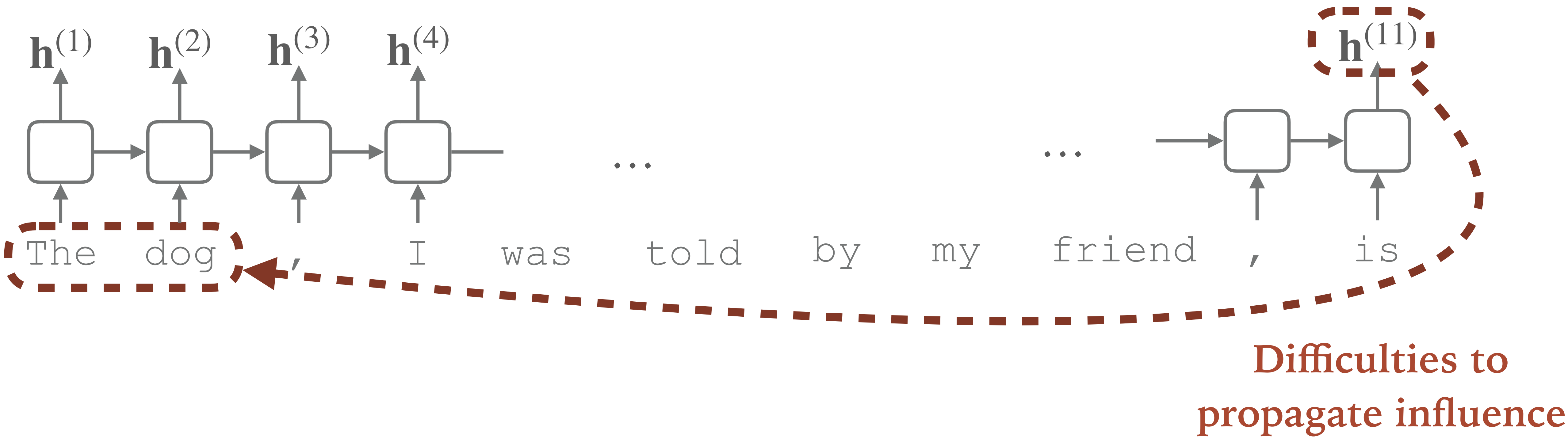
- In theory: yes
- In practice: no, gradient based learning of RNN fail to learn long range dependencies!



GRADIENT BASED LEARNING PROBLEM

Does it work?

- In theory: yes
- In practice: no, gradient based learning of RNN fail to learn long range dependencies!



Deep learning is not a « single tool fits all problem » solution

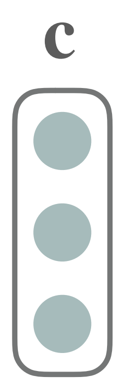
- You need to understand your data and prediction task
- You need to understand why a given neural architecture may fail for a given task
- You need to be able design tailored neural architectures for a given task

LONG SHORT-TERM MEMORY NETWORKS

LONG SHORT-TERM MEMORY NETWORKS (LSTM)

Intuition

Memory vector

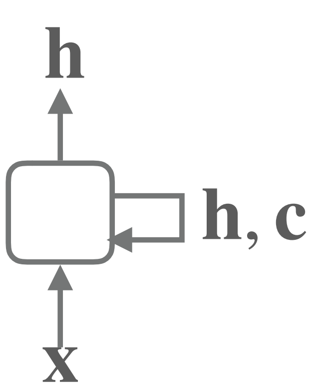


- Memory vector which is passed along the sequence
- At each time step, the network selects which cell of the memory to modify

The network can learn to keep track of long distance relationships

LSTM cell

- The recurrent connection pass the memory vector to the next cell



ERASING/WRITING VALUES IN A VECTOR

Erasing values in the memory

$$\begin{bmatrix} 3.02 \\ -4.11 \\ 21.00 \\ 4.44 \\ -6.9 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 \\ 0 \\ 21.00 \\ 4.44 \\ -6.9 \end{bmatrix}$$

« Forget » the first two cells

ERASING/WRITING VALUES IN A VECTOR

Erasing values in the memory

$$\begin{bmatrix} 3.02 \\ -4.11 \\ 21.00 \\ 4.44 \\ -6.9 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 \\ 0 \\ 21.00 \\ 4.44 \\ -6.9 \end{bmatrix}$$

« Forget » the first two cells

Writing values in the memory

$$\begin{bmatrix} 0 \\ 0 \\ 21.00 \\ 4.44 \\ -6.9 \end{bmatrix} + \begin{bmatrix} 10.0 \\ 5.0 \\ 1.0 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 10.0 \\ 5.0 \\ 22.00 \\ 4.44 \\ -6.9 \end{bmatrix}$$

Memory after update

Memory before update

Update

GATE MECHANISM

Erasing values in a vector

Not necessarily c, it can depends on anything

Let assume we want to remove some values from a vector $\mathbf{c}$:

1. A simple linear classifier compute the importance of each value in $\mathbf{c}$: $\mathbf{w} = \mathbf{U}\mathbf{c} + \mathbf{b}$
2. We erase non important value, i.e. values with a negative weight in $\mathbf{w}$

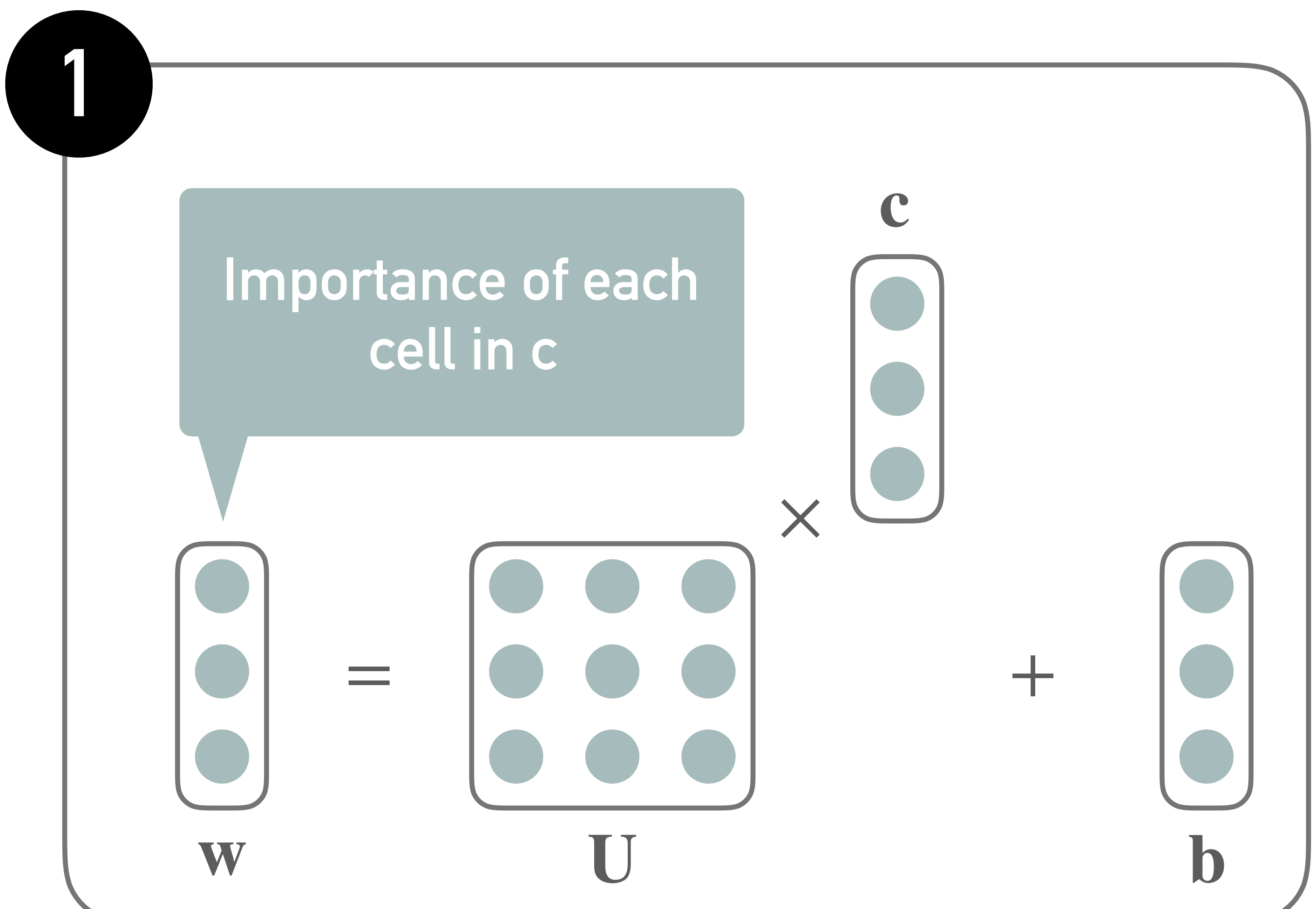
GATE MECHANISM

Erasing values in a vector

Not necessarily c, it can depends on anything

Let assume we want to remove some values from a vector $\mathbf{c}$:

1. A simple linear classifier compute the importance of each value in $\mathbf{c}$: $\mathbf{w} = \mathbf{U}\mathbf{c} + \mathbf{b}$
2. We erase non important value, i.e. values with a negative weight in $\mathbf{w}$



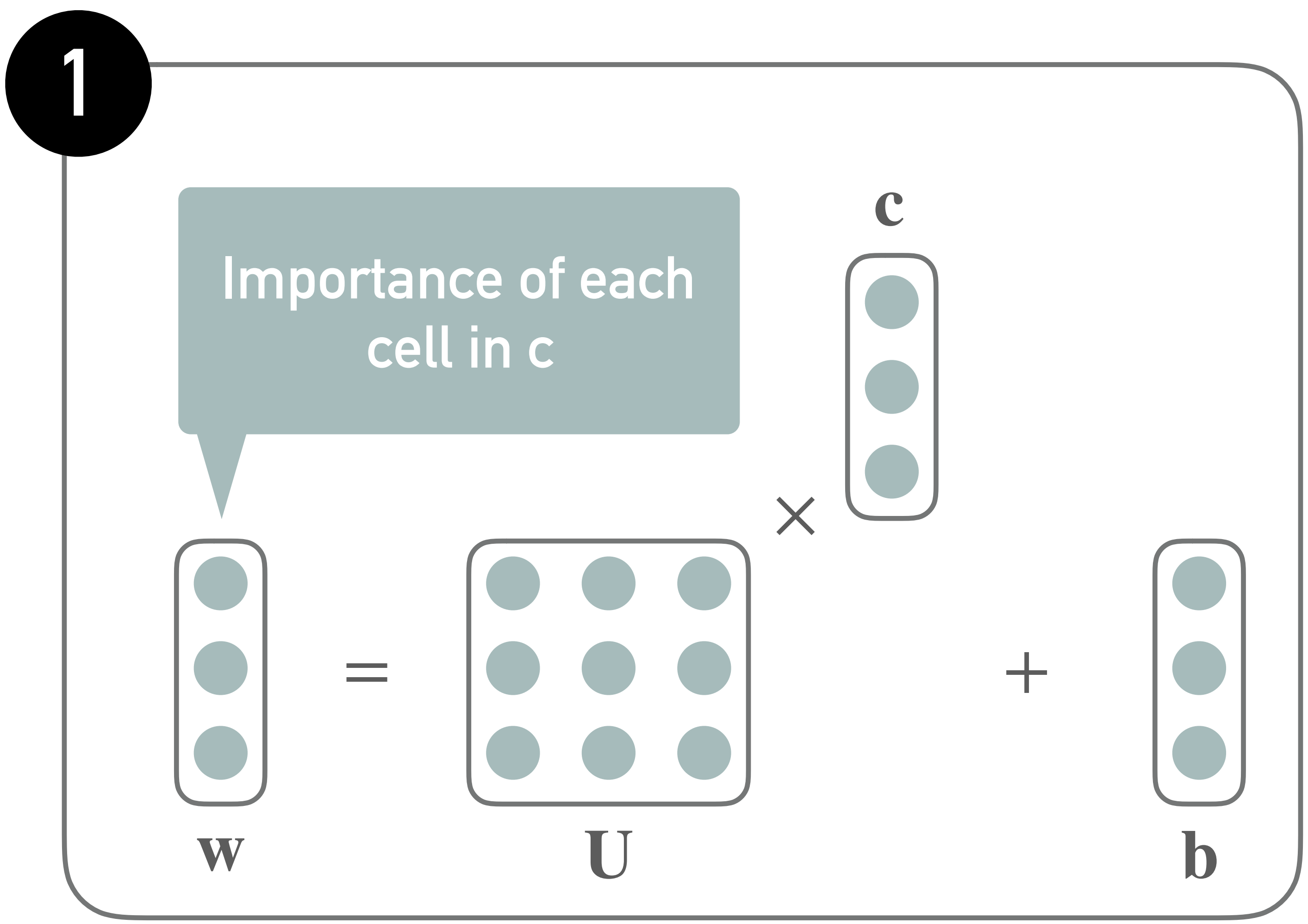
GATE MECHANISM

Erasing values in a vector

Not necessarily c, it can depends on anything

Let assume we want to remove some values from a vector $\mathbf{c}$:

1. A simple linear classifier compute the importance of each value in $\mathbf{c}$: $\mathbf{w} = \mathbf{U}\mathbf{c} + \mathbf{b}$
2. We erase non important value, i.e. values with a negative weight in $\mathbf{w}$



2

$$\mathbf{c}'_i = \begin{cases} \mathbf{c}_i \; if \; \mathbf{w}_i > 0, \\ 0 \; otherwise \end{cases}$$

GATE MECHANISM

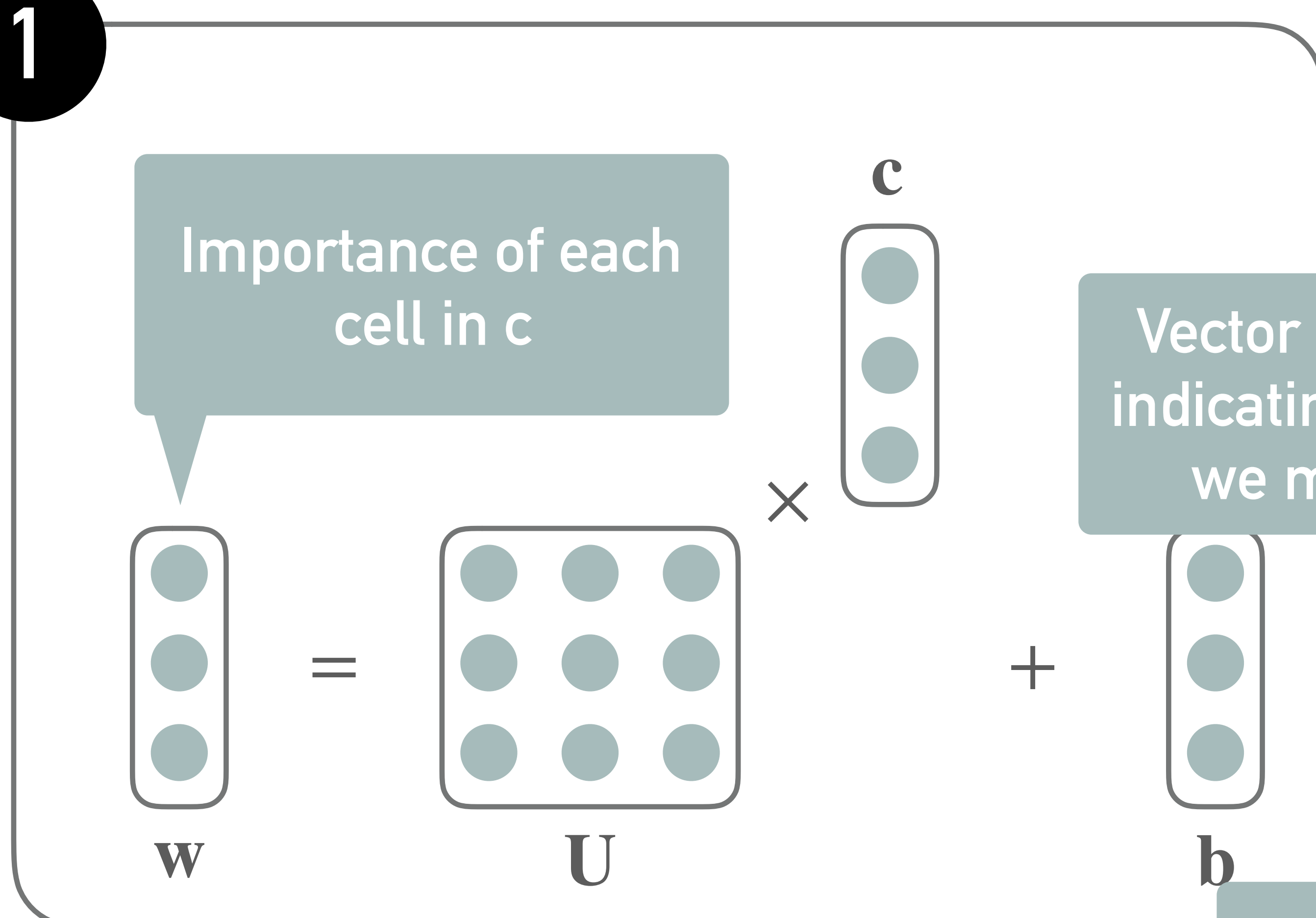
Erasing values in a vector

Let assume we want to remove some values from a vector $\mathbf{c}$:

Not necessarily c, it can depends on anything

1. A simple linear classifier compute the importance of each value in $\mathbf{c}$: $\mathbf{w} = \mathbf{U}\mathbf{c} + \mathbf{b}$
2. We erase non important value, i.e. values with a negative weight in $\mathbf{w}$

1



2

$$\mathbf{c}'_i = \begin{cases} \mathbf{c}_i \; if \; \mathbf{w}_i > 0, \\ 0 \; otherwise \end{cases}$$

OR

$$\mathbf{b}_i = \begin{cases} 1 \; if \; \mathbf{w}_i > 0, \\ 0 \; otherwise \end{cases}$$

$$\mathbf{c}' = \mathbf{c} \odot \mathbf{b}$$

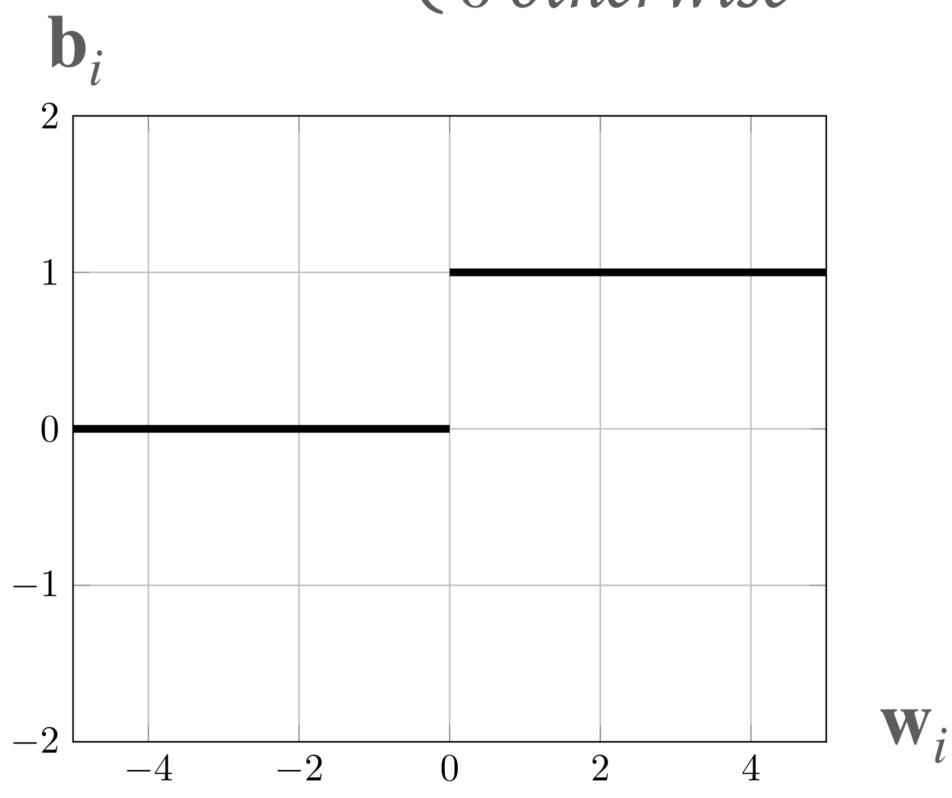
Element-wise multiplication

CELL SELECTION AND BACKPROPAGATION?

Forward pass

$$\mathbf{w} = \mathbf{U}\mathbf{c} + \mathbf{b}$$

$$\mathbf{b}_i = \begin{cases} 1 \ \textit{if} \ \mathbf{w}_i > 0, \\ 0 \ \textit{otherwise} \end{cases}$$

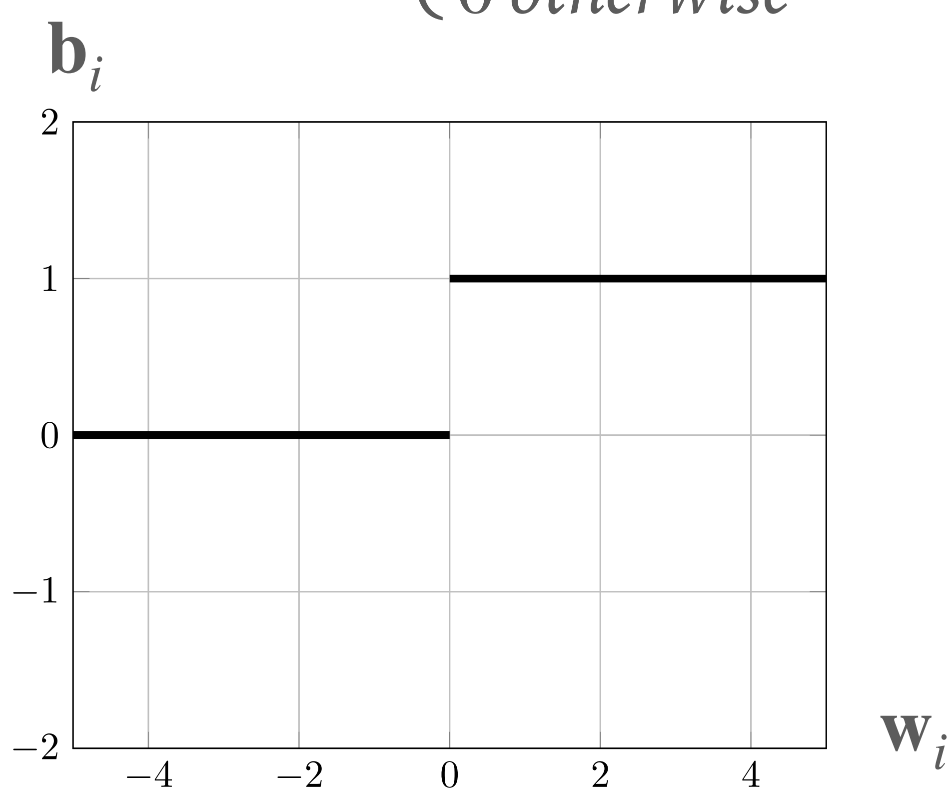


CELL SELECTION AND BACKPROPAGATION?

Forward pass

$$\mathbf{w} = \mathbf{U}\mathbf{c} + \mathbf{b}$$

$$\mathbf{b}_i = \begin{cases} 1 \; if \; \mathbf{w}_i > 0, \\ 0 \; otherwise \end{cases}$$



Backward pass

Gradient wrt the loss

By the chain rule: $$\frac{\partial \mathscr{L}}{\partial \mathbf{w}_i} = \frac{\partial \mathscr{L}}{\partial \mathbf{b}_i} \cdot \frac{\partial \mathbf{b}_i}{\partial \mathbf{w}_i} + \ldots$$

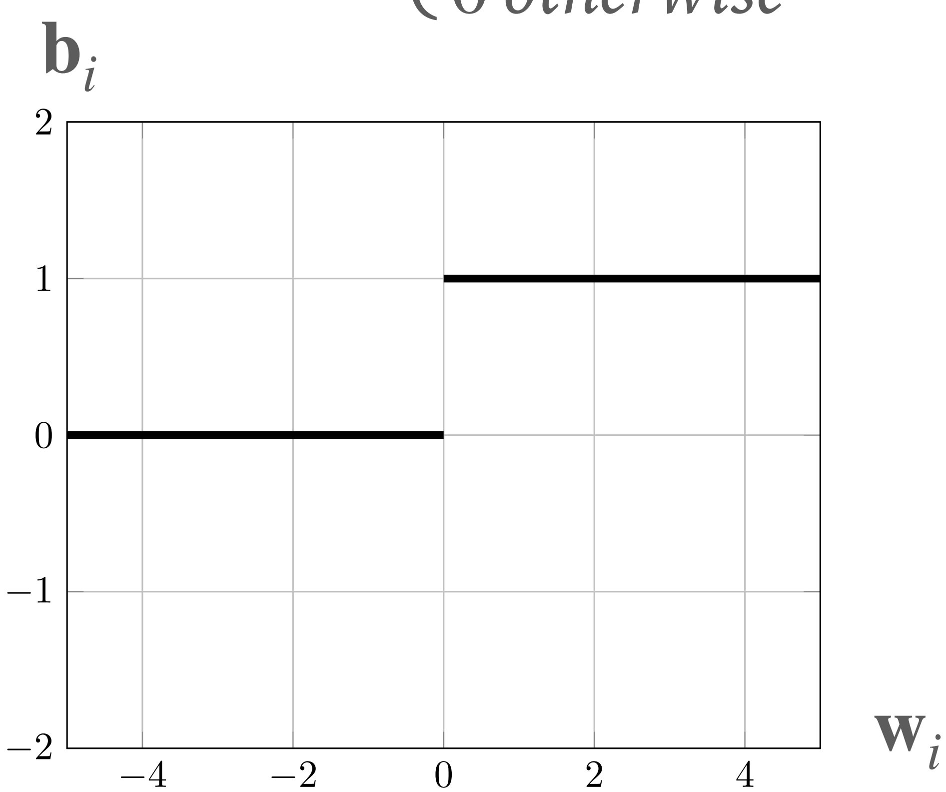
What does this term look like?

CELL SELECTION AND BACKPROPAGATION?

Forward pass

$$\mathbf{w} = \mathbf{U}\mathbf{c} + \mathbf{b}$$

$$\mathbf{b}_i = \begin{cases} 1 \; if \; \mathbf{w}_i > 0, \\ 0 \; otherwise \end{cases}$$

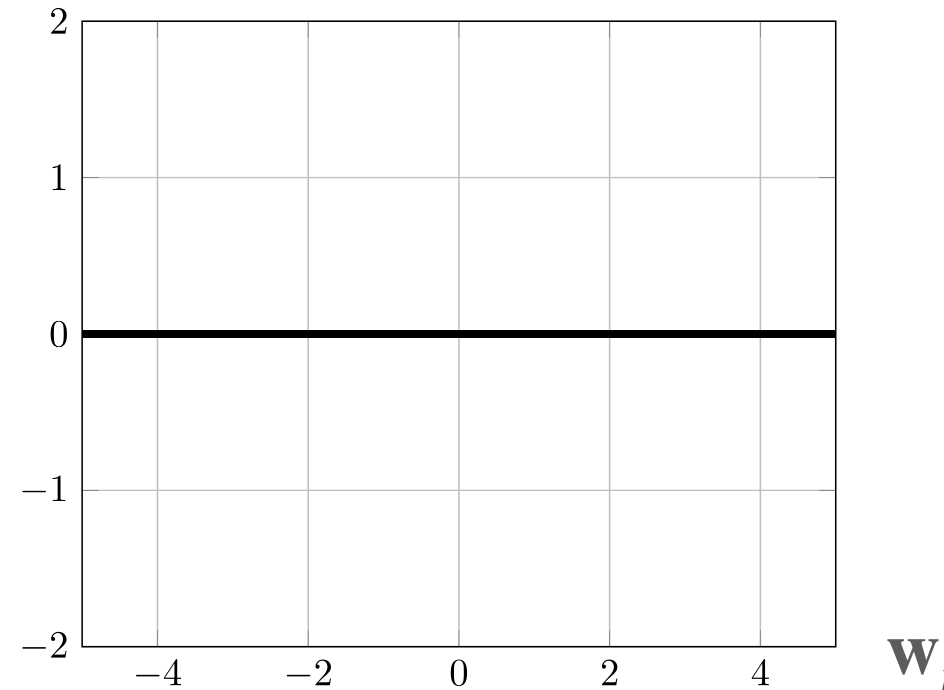


Backward pass

Gradient wrt the loss

By the chain rule: $$\frac{\partial \mathscr{L}}{\partial \mathbf{w}_i} = \frac{\partial \mathscr{L}}{\partial \mathbf{b}_i} \cdot \frac{\partial \mathbf{b}_i}{\partial \mathbf{w}_i} + \ldots$$

What does this term look like?



! Gradient is blocked!
No information is back propagated!

SMOOTH SELECTION 1/2

Equivalent formulation as a small optimization problem

$$\mathbf{b}_i = \begin{cases} 1 \; \mathit{if} \; \mathbf{w}_i > 0, \\ 0 \; \mathit{otherwise} \end{cases}$$

OR

$$\begin{aligned} \mathbf{b}_i = \operatorname{argmax}_{\mathbf{y}_i} \quad & \mathbf{y}_i \times \mathbf{w}_i \\ s.t. \quad & \mathbf{y}_i \leq 1 \\ & \mathbf{y}_i \geq 0 \end{aligned}$$

SMOOTH SELECTION 1/2

Equivalent formulation as a small optimization problem

$$\mathbf{b}_i = \begin{cases} 1 \; \textit{if} \; \mathbf{w}_i > 0, \\ 0 \; \textit{otherwise} \end{cases}$$

OR

$$\mathbf{b}_i = \operatorname{argmax}_{\mathbf{y}_i} \quad \mathbf{y}_i \times \mathbf{w}_i$$
$$s.t. \quad \mathbf{y}_i \leq 1$$
$$\mathbf{y}_i \geq 0$$

Intuition

- At the optimal solution, one of the <u>**constraint is tight**</u>
  => small perturbation on $\mathbf{w}_i$ will not change the solution
- We can introduce a <u>**penalty in the objective**</u> so that <u>**constraints are never tight**</u> at the optimal solution

SMOOTH SELECTION 1/2

Equivalent formulation as a small optimization problem

$$\mathbf{b}_i = \begin{cases} 1 \; \textit{if} \; \mathbf{w}_i > 0, \\ 0 \; \textit{otherwise} \end{cases}$$

OR

$$\begin{aligned} \mathbf{b}_i = \operatorname{argmax}_{\mathbf{y}_i} \quad & \mathbf{y}_i \times \mathbf{w}_i \\ s.t. \quad & \mathbf{y}_i \leq 1 \\ & \mathbf{y}_i \geq 0 \end{aligned}$$

Intuition

- At the optimal solution, one of the constraint is tight
  => small perturbation on $\mathbf{w}_i$ will not change the solution
- We can introduce a **penalty in the objective** so that **constraints are never tight** at the optimal solution

$$\begin{aligned} \mathbf{b}_i = \operatorname{argmax}_{\mathbf{y}_i} \quad & \mathbf{y}_i \times \mathbf{w}_i - \Omega(\mathbf{y}_i) \\ s.t. \quad & \mathbf{y}_i \leq 1 \\ & \mathbf{y}_i \geq 0 \end{aligned}$$

Strong convex regularizer

SMOOTH SELECTION 1/2

$$\mathbf{b}_i = \text{argmax}_{\mathbf{y}_i} \quad \mathbf{y}_i \times \mathbf{w}_i - \Omega(\mathbf{y}_i)$$
$$s.t. \quad \mathbf{y}_i \leq 1$$
$$\mathbf{y}_i \geq 0$$

How to choose the convex regularizer?

- We need to solve the program quickly
- We need to be able to back propagate easily
- Several solutions
  (i.e. similar to interior point method)

SMOOTH SELECTION 1/2

$$\mathbf{b}_i = \operatorname{argmax}_{\mathbf{y}_i} \quad \mathbf{y}_i \times \mathbf{w}_i - \Omega(\mathbf{y}_i)$$
$$s.t. \quad \mathbf{y}_i \leq 1$$
$$\mathbf{y}_i \geq 0$$

How to choose the convex regularizer?

- We need to solve the program quickly
- We need to be able to back propagate easily
- Several solutions
  (i.e. similar to interior point method)

Negative Fermi-Dirac entropy

$$\mathbf{b}_i = \operatorname{argmax}_{\mathbf{y}_i} \quad \mathbf{y}_i \times \mathbf{w}_i - \mathbf{y}_i \log \mathbf{y}_i - (1 - \mathbf{y}_i)\log(1 - \mathbf{y}_i)$$
$$s.t. \quad \mathbf{y}_i \leq 1$$
$$\mathbf{y}_i \geq 0$$

SMOOTH SELECTION 1/2

$$\mathbf{b}_i = \operatorname{argmax}_{\mathbf{y}_i} \quad \mathbf{y}_i \times \mathbf{w}_i - \Omega(\mathbf{y}_i)$$
$$s.t. \quad \mathbf{y}_i \leq 1$$
$$\mathbf{y}_i \geq 0$$

How to choose the convex regularizer?

- We need to solve the program quickly
- We need to be able to back propagate easily
- Several solutions
  (i.e. similar to interior point method)

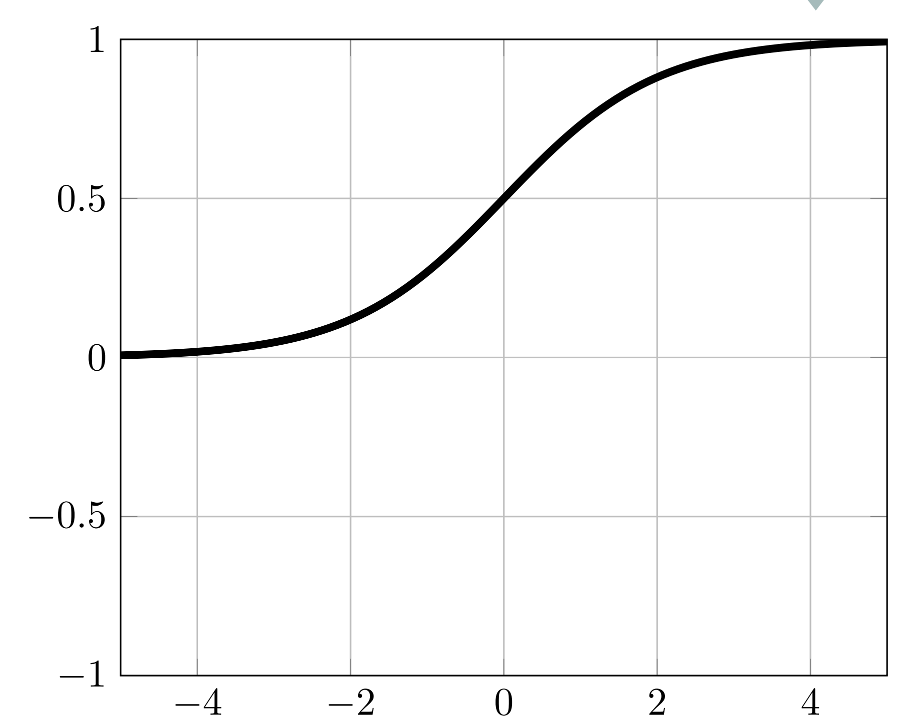
Negative Fermi-Dirac entropy

$$\mathbf{b}_i = \operatorname{argmax}_{\mathbf{y}_i} \quad \mathbf{y}_i \times \mathbf{w}_i - \mathbf{y}_i \log \mathbf{y}_i - (1 - \mathbf{y}_i) \log(1 - \mathbf{y}_i)$$
$$s.t. \quad \mathbf{y}_i \leq 1$$
$$\mathbf{y}_i \geq 0$$

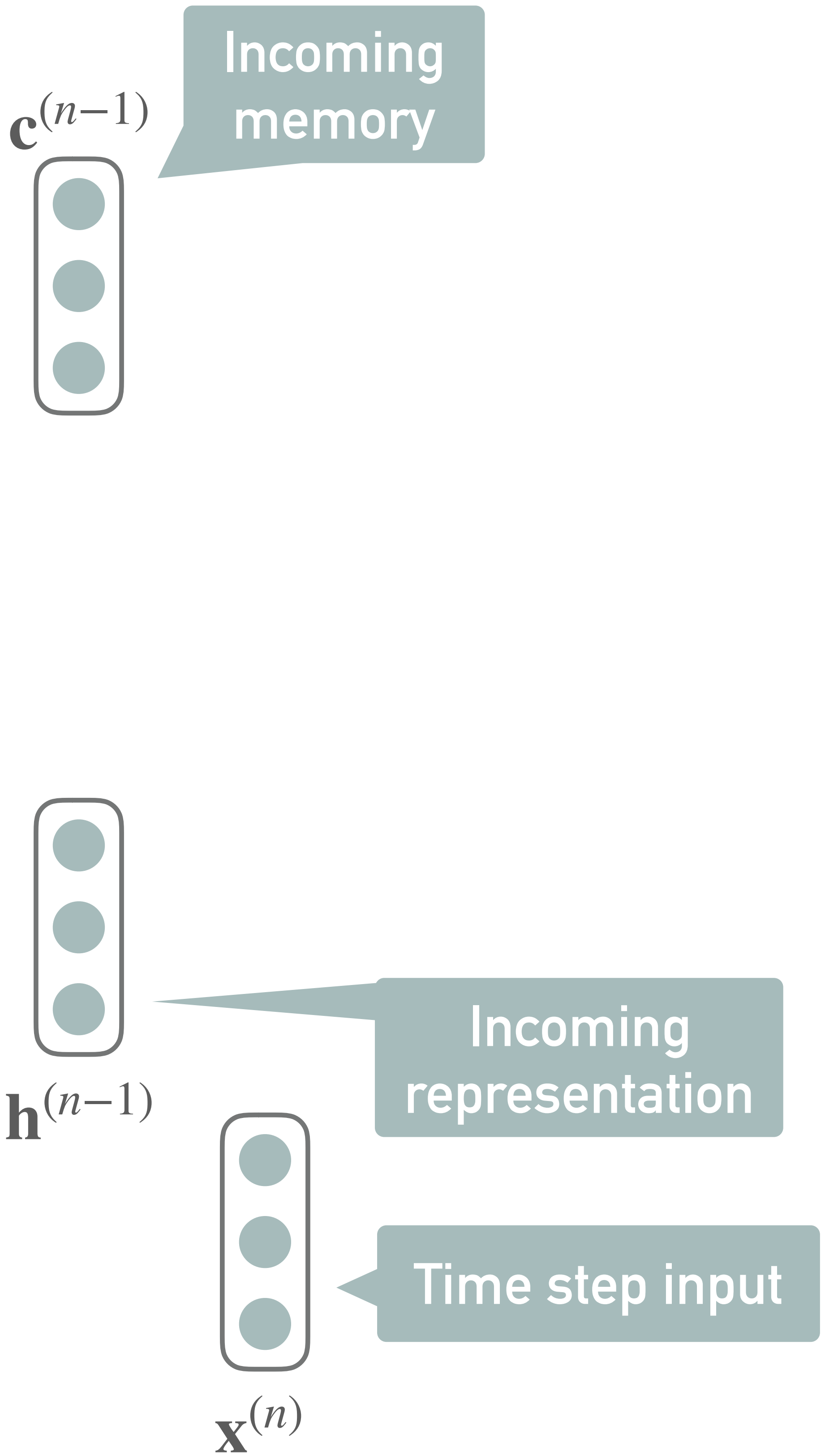
This is actually the sigmoid
(solve the KKT condition to see that)

$$\mathbf{b}_i = \frac{1}{(1 + exp(-\mathbf{w}_i))} = \sigma(\mathbf{w}_i)$$

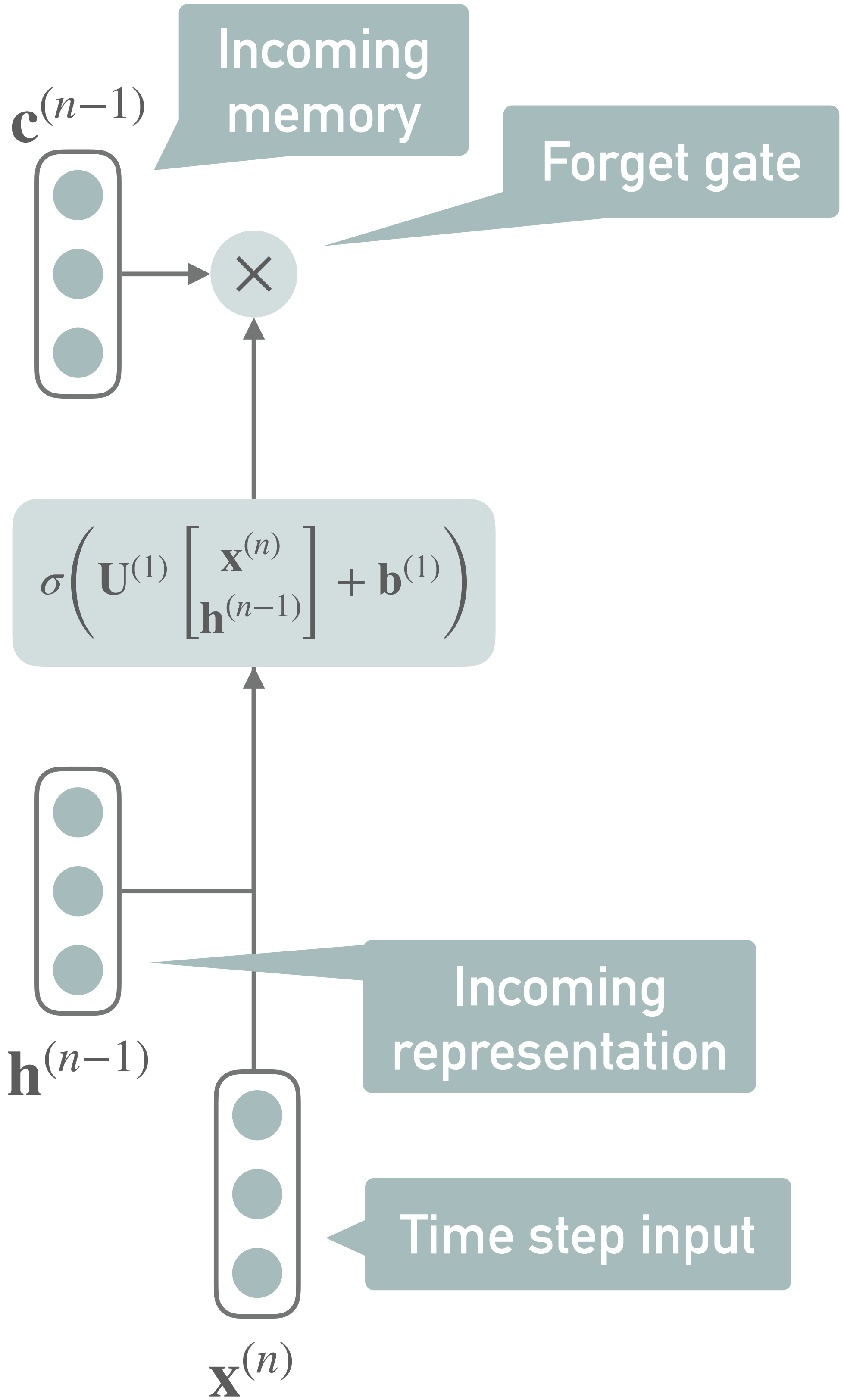
Smooth and differentiable approximation! :)



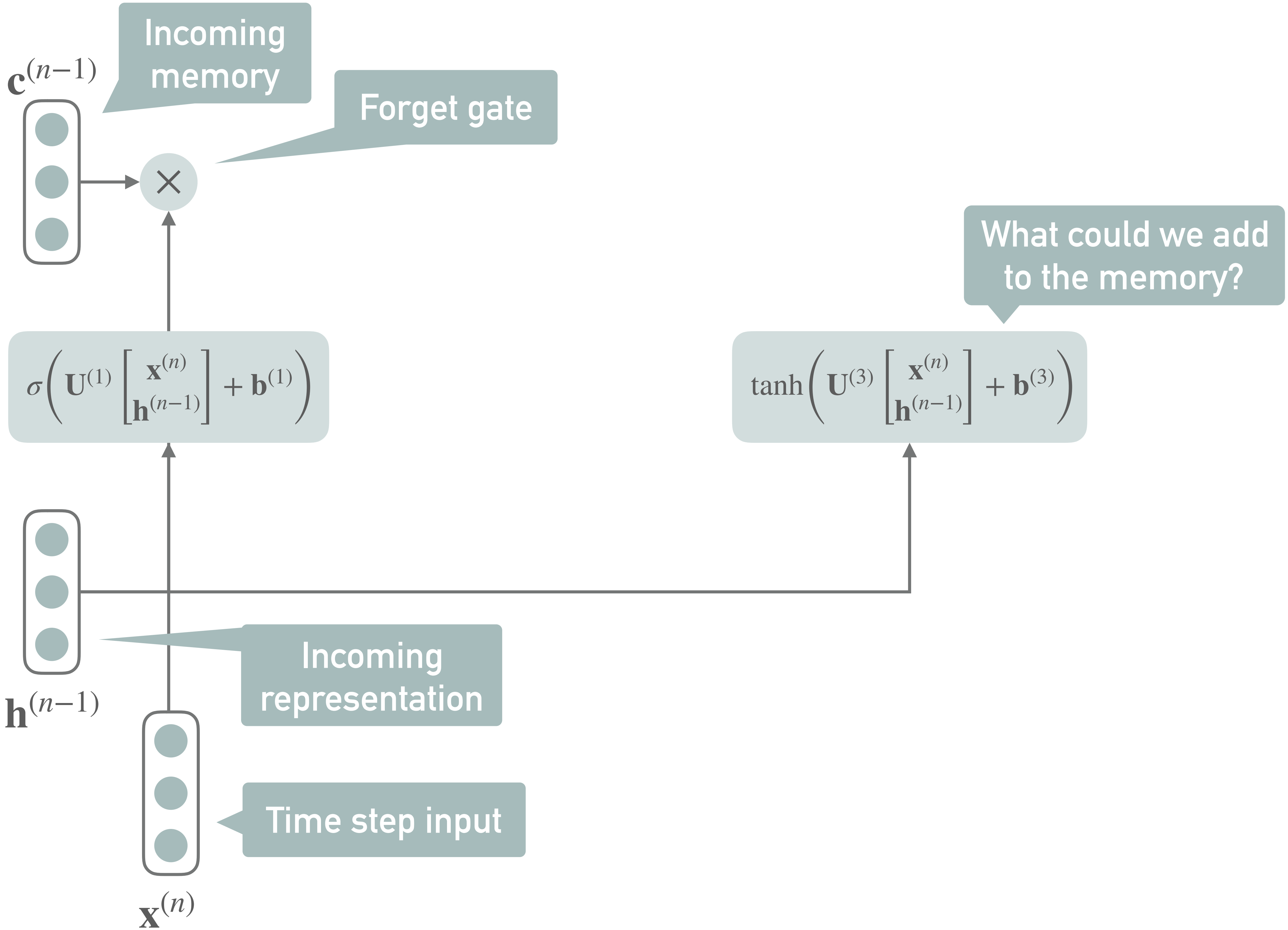
LSTM CELL 1/2


$\mathbf{c}^{(n-1)}$
Incoming memory
Incoming representation
$\mathbf{h}^{(n-1)}$
Time step input
$\mathbf{x}^{(n)}$

LSTM CELL 1/2

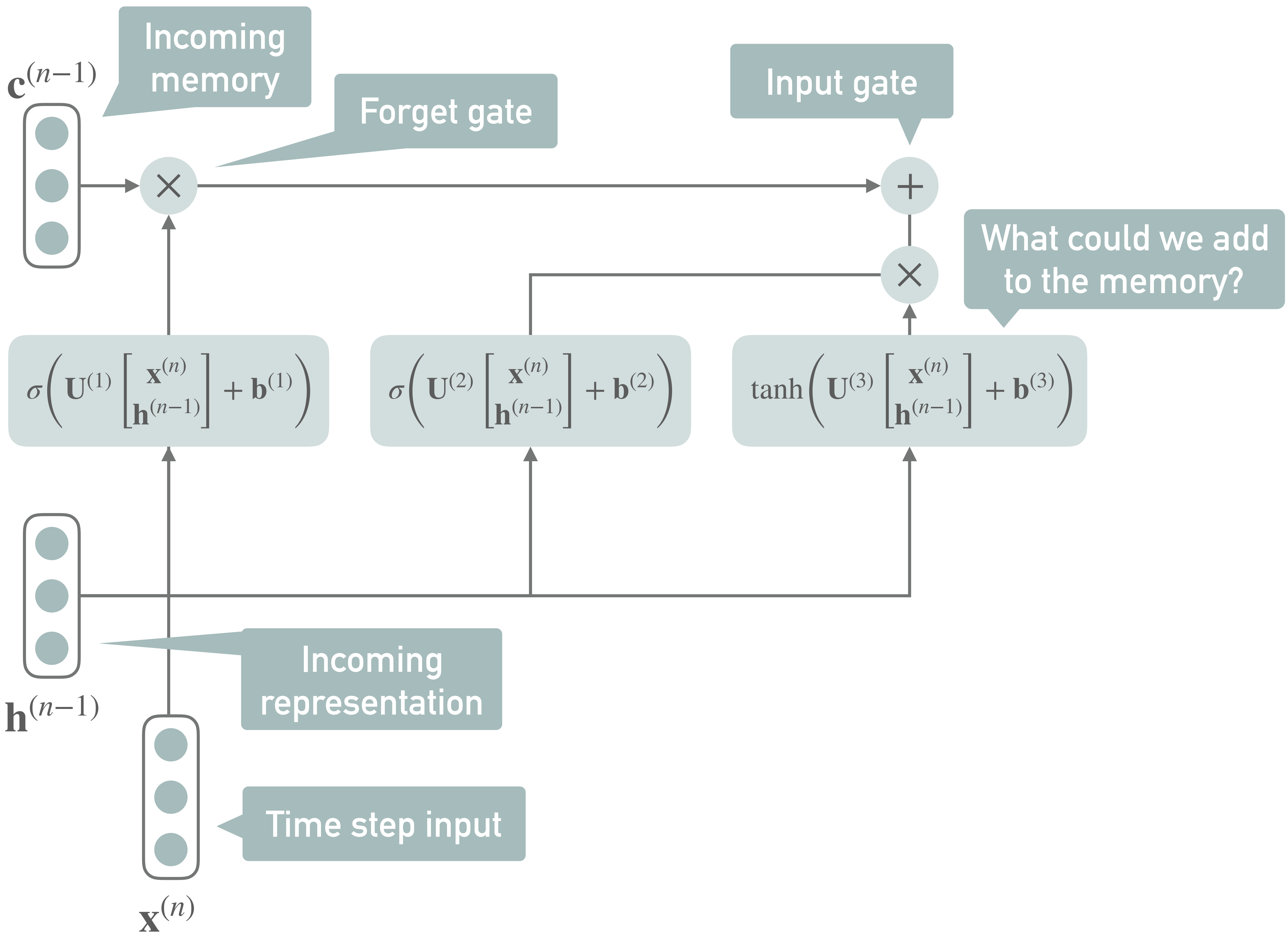

Incoming memory
$\mathbf{c}^{(n-1)}$
Forget gate
$\sigma\left(\mathbf{U}^{(1)}\begin{bmatrix}\mathbf{x}^{(n)}\\ \mathbf{h}^{(n-1)}\end{bmatrix}+\mathbf{b}^{(1)}\right)$
Incoming representation
$\mathbf{h}^{(n-1)}$
Time step input
$\mathbf{x}^{(n)}$

LSTM CELL 1/2



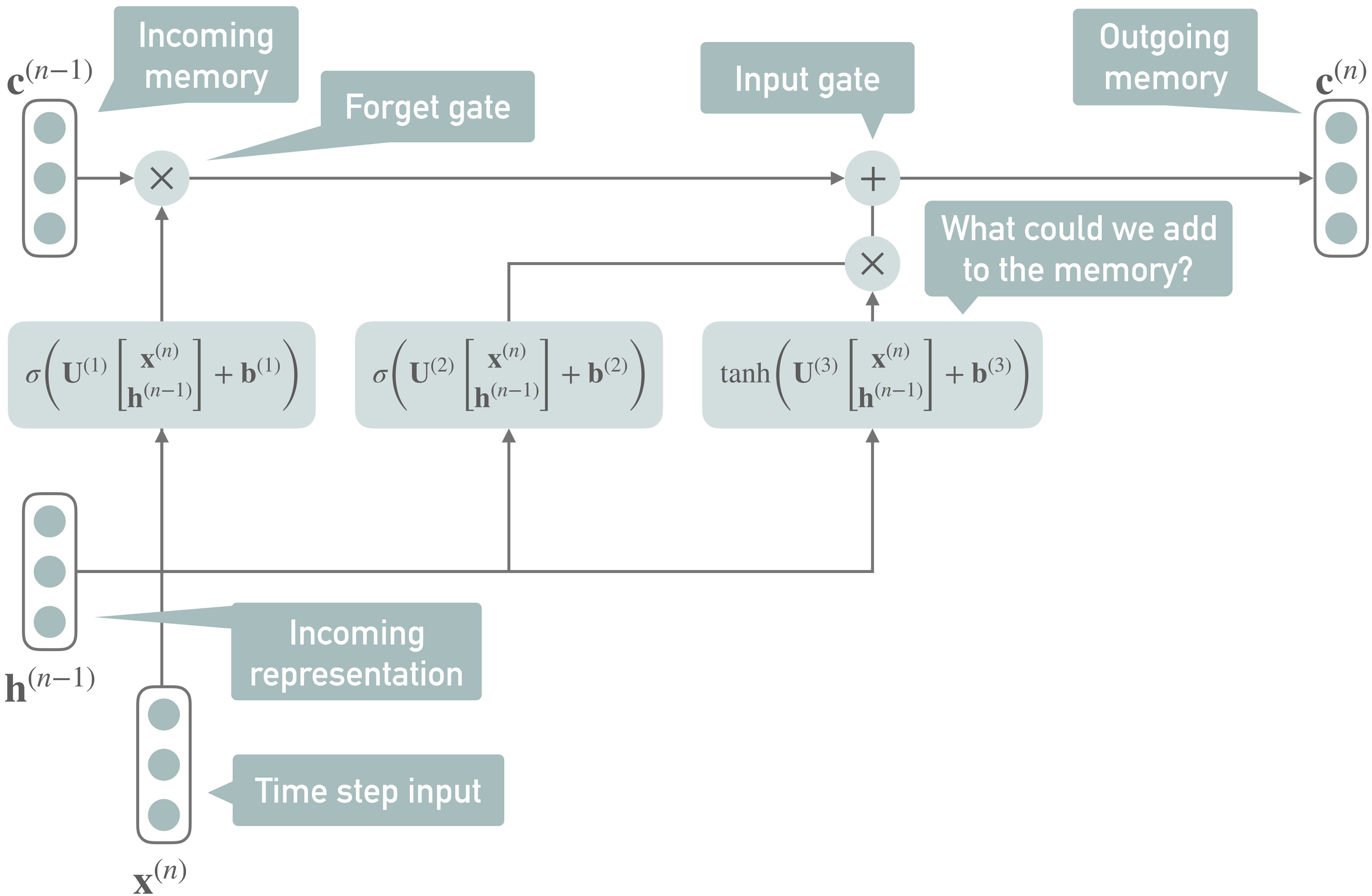
$\mathbf{c}^{(n-1)}$
Incoming memory
Forget gate
$\times$
What could we add to the memory?
$\sigma\left(\mathbf{U}^{(1)}\begin{bmatrix}\mathbf{x}^{(n)}\\\mathbf{h}^{(n-1)}\end{bmatrix}+\mathbf{b}^{(1)}\right)$
$\tanh\left(\mathbf{U}^{(3)}\begin{bmatrix}\mathbf{x}^{(n)}\\\mathbf{h}^{(n-1)}\end{bmatrix}+\mathbf{b}^{(3)}\right)$
Incoming representation
$\mathbf{h}^{(n-1)}$
Time step input
$\mathbf{x}^{(n)}$

LSTM CELL 1/2



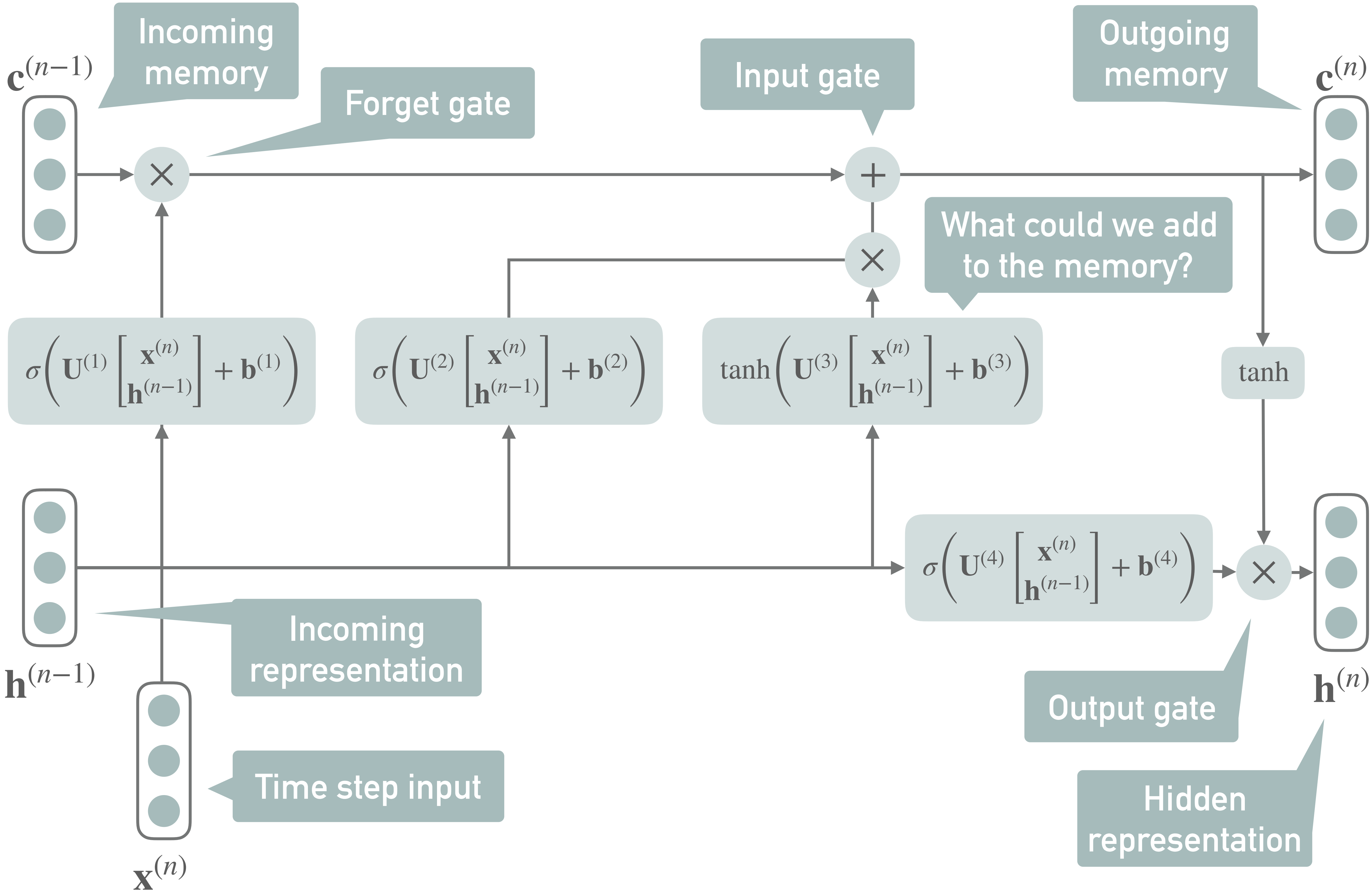
$\mathbf{c}^{(n-1)}$
Incoming memory
Forget gate
Input gate
What could we add to the memory?
$\sigma\left(\mathbf{U}^{(1)}\begin{bmatrix}\mathbf{x}^{(n)}\\ \mathbf{h}^{(n-1)}\end{bmatrix}+\mathbf{b}^{(1)}\right)$
$\sigma\left(\mathbf{U}^{(2)}\begin{bmatrix}\mathbf{x}^{(n)}\\ \mathbf{h}^{(n-1)}\end{bmatrix}+\mathbf{b}^{(2)}\right)$
$\tanh\left(\mathbf{U}^{(3)}\begin{bmatrix}\mathbf{x}^{(n)}\\ \mathbf{h}^{(n-1)}\end{bmatrix}+\mathbf{b}^{(3)}\right)$
Incoming representation
$\mathbf{h}^{(n-1)}$
Time step input
$\mathbf{x}^{(n)}$

LSTM CELL 1/2



Incoming memory
Forget gate
Input gate
Outgoing memory
What could we add to the memory?
Incoming representation
Time step input
$\mathbf{c}^{(n-1)}$
$\mathbf{c}^{(n)}$
$\mathbf{h}^{(n-1)}$
$\mathbf{x}^{(n)}$
$\sigma\left(\mathbf{U}^{(1)}\begin{bmatrix}\mathbf{x}^{(n)}\\ \mathbf{h}^{(n-1)}\end{bmatrix}+\mathbf{b}^{(1)}\right)$
$\sigma\left(\mathbf{U}^{(2)}\begin{bmatrix}\mathbf{x}^{(n)}\\ \mathbf{h}^{(n-1)}\end{bmatrix}+\mathbf{b}^{(2)}\right)$
$\tanh\left(\mathbf{U}^{(3)}\begin{bmatrix}\mathbf{x}^{(n)}\\ \mathbf{h}^{(n-1)}\end{bmatrix}+\mathbf{b}^{(3)}\right)$

LSTM CELL 1/2



$\mathbf{c}^{(n-1)}$
Incoming memory
Forget gate
Input gate
Outgoing memory
$\mathbf{c}^{(n)}$
What could we add to the memory?
$\sigma\left(\mathbf{U}^{(1)}\begin{bmatrix}\mathbf{x}^{(n)}\\\mathbf{h}^{(n-1)}\end{bmatrix}+\mathbf{b}^{(1)}\right)$
$\sigma\left(\mathbf{U}^{(2)}\begin{bmatrix}\mathbf{x}^{(n)}\\\mathbf{h}^{(n-1)}\end{bmatrix}+\mathbf{b}^{(2)}\right)$
$\tanh\left(\mathbf{U}^{(3)}\begin{bmatrix}\mathbf{x}^{(n)}\\\mathbf{h}^{(n-1)}\end{bmatrix}+\mathbf{b}^{(3)}\right)$
tanh
$\sigma\left(\mathbf{U}^{(4)}\begin{bmatrix}\mathbf{x}^{(n)}\\\mathbf{h}^{(n-1)}\end{bmatrix}+\mathbf{b}^{(4)}\right)$
Incoming representation
$\mathbf{h}^{(n-1)}$
Output gate
$\mathbf{h}^{(n)}$
Time step input
$\mathbf{x}^{(n)}$
Hidden representation

LSTM CELL 2/2

Gates

$$\mathbf{f}^{(n)} = \sigma\left(\mathrm{U}^{(1)} \begin{bmatrix} \mathbf{x}^{(n)} \\ \mathbf{h}^{(n-1)} \end{bmatrix} + \mathbf{b}^{(1)}\right)$$

$$\mathbf{i}^{(n)} = \sigma\left(\mathrm{U}^{(2)} \begin{bmatrix} \mathbf{x}^{(n)} \\ \mathbf{h}^{(n-1)} \end{bmatrix} + \mathbf{b}^{(2)}\right)$$

$$\mathbf{o}^{(n)} = \sigma\left(\mathrm{U}^{(4)} \begin{bmatrix} \mathbf{x}^{(n)} \\ \mathbf{h}^{(n-1)} \end{bmatrix} + \mathbf{b}^{(4)}\right)$$

Outputs

$$\mathbf{c}^{(n)} = \underbrace{\mathbf{f}^{(n)} \times \mathbf{c}^{(n-1)}}_{\text{Erase memory}} + \underbrace{\mathbf{i}^{(n)} \times \tanh\left(\mathrm{U}^{(3)} \begin{bmatrix} \mathbf{x}^{(n)} \\ \mathbf{h}^{(n-1)} \end{bmatrix} + \mathbf{b}^{(3)}\right)}_{\text{Update memory}}$$

$$\mathbf{h}^{(n)} = \mathbf{o}^{(n)} \times \tanh(\mathbf{c}^{(n)})$$

Compute output wrt memory

Number of parameters

4 times more parameters than a simple recurrent neural network!

LSTM VARIANT: COUPLED FORGET AND INPUT GATES

Intuition

- Tie forget and input gates
- Each memory cell is either kept as it or replaced by a new value

Gates

$$\mathbf{f}^{(n)} = \sigma\left(\mathbf{U}^{(1)} \begin{bmatrix} \mathbf{x}^{(n)} \\ \mathbf{h}^{(n-1)} \end{bmatrix} + \mathbf{b}^{(1)}\right)$$

$$\boxed{\mathbf{i}^{(n)} = 1 - \mathbf{f}^{(n)}}$$

Input gate is tied to the forget gate

$$\mathbf{o}^{(n)} = \sigma\left(\mathbf{U}^{(4)} \begin{bmatrix} \mathbf{x}^{(n)} \\ \mathbf{h}^{(n-1)} \end{bmatrix} + \mathbf{b}^{(4)}\right)$$

Outputs

$$\mathbf{c}^{(n)} = \boxed{\mathbf{f}^{(n)} \times \mathbf{c}^{(n-1)}} + \boxed{\mathbf{i}^{(n)} \times \tanh\left(\mathbf{U}^{(3)} \begin{bmatrix} \mathbf{x}^{(n)} \\ \mathbf{h}^{(n-1)} \end{bmatrix} + \mathbf{b}^{(3)}\right)}$$

$$\mathbf{h}^{(n)} = \mathbf{o}^{(n)} \times \tanh(\mathbf{c}^{(n)})$$

LSTM VARIANT: PEEPHOLES

Intuition

- In standard LSTMs, gates **are not dependent** on the memory state
- In peephole LSTMs, gates **depend** on the memory

LSTM VARIANT: PEEPHOLES

Intuition

- In standard LSTMs, gates **are not dependent** on the memory state
- In peephole LSTMs, gates **depend** on the memory

Gates

$$\mathbf{f}^{(n)} = \sigma\left(\mathbf{U}^{(1)}\begin{bmatrix}\mathbf{x}^{(n)}\\ \mathbf{h}^{(n-1)}\\ \mathbf{c}^{(n-1)}\end{bmatrix} + \mathbf{b}^{(1)}\right)$$

$$\mathbf{i}^{(n)} = \sigma\left(\mathbf{U}^{(2)}\begin{bmatrix}\mathbf{x}^{(n)}\\ \mathbf{h}^{(n-1)}\\ \mathbf{c}^{(n-1)}\end{bmatrix} + \mathbf{b}^{(2)}\right)$$

Look memory content to choose which cell to change

LSTM VARIANT: PEEPHOLES

Intuition

- In standard LSTMs, gates **are not dependent** on the memory state
- In peephole LSTMs, gates **depend** on the memory

Gates

$$\mathbf{f}^{(n)} = \sigma\left(\mathbf{U}^{(1)}\begin{bmatrix}\mathbf{x}^{(n)}\\ \mathbf{h}^{(n-1)}\\ \mathbf{c}^{(n-1)}\end{bmatrix} + \mathbf{b}^{(1)}\right)$$

$$\mathbf{i}^{(n)} = \sigma\left(\mathbf{U}^{(2)}\begin{bmatrix}\mathbf{x}^{(n)}\\ \mathbf{h}^{(n-1)}\\ \mathbf{c}^{(n-1)}\end{bmatrix} + \mathbf{b}^{(2)}\right)$$

Look memory content to choose which cell to change

$$\mathbf{o}^{(n)} = \sigma\left(\mathbf{U}^{(4)}\begin{bmatrix}\mathbf{x}^{(n)}\\ \mathbf{h}^{(n-1)}\\ \mathbf{c}^{(n)}\end{bmatrix} + \mathbf{b}^{(4)}\right)$$

Output gate depend on the new memory state

LSTM VARIANT: PEEPHOLES

Intuition

- In standard LSTMs, gates **are not dependent** on the memory state
- In peephole LSTMs, gates **depend** on the memory

Gates

$$\mathbf{f}^{(n)} = \sigma\left(\mathbf{U}^{(1)} \boxed{\begin{bmatrix} \mathbf{x}^{(n)} \\ \mathbf{h}^{(n-1)} \\ \mathbf{c}^{(n-1)} \end{bmatrix}} + \mathbf{b}^{(1)}\right)$$

$$\mathbf{i}^{(n)} = \sigma\left(\mathbf{U}^{(2)} \boxed{\begin{bmatrix} \mathbf{x}^{(n)} \\ \mathbf{h}^{(n-1)} \\ \mathbf{c}^{(n-1)} \end{bmatrix}} + \mathbf{b}^{(2)}\right)$$

$$\mathbf{o}^{(n)} = \sigma\left(\mathbf{U}^{(4)} \boxed{\begin{bmatrix} \mathbf{x}^{(n)} \\ \mathbf{h}^{(n-1)} \\ \mathbf{c}^{(n)} \end{bmatrix}} + \mathbf{b}^{(4)}\right)$$

Look memory content to choose which cell to change

Output gate depend on the new memory state

Outputs

Unchanged

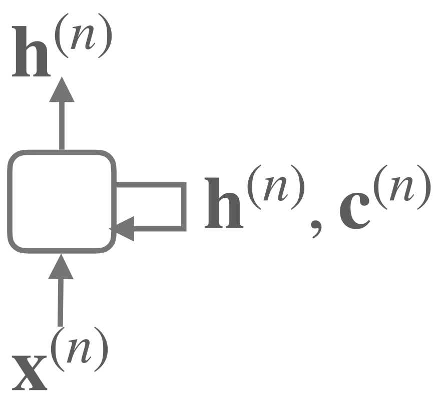
$$\mathbf{c}^{(n)} = \boxed{\mathbf{f}^{(n)} \times \mathbf{c}^{(n-1)}} + \boxed{\mathbf{i}^{(n)} \times \tanh\left(\mathbf{U}^{(3)} \begin{bmatrix} \mathbf{x}^{(n)} \\ \mathbf{h}^{(n-1)} \end{bmatrix} + \mathbf{b}^{(3)}\right)}$$

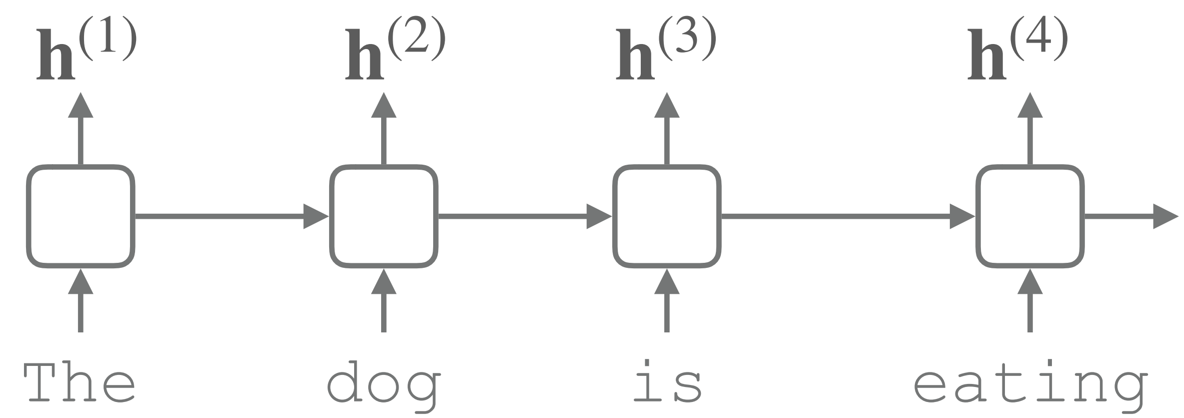
$$\mathbf{h}^{(n)} = \mathbf{o}^{(n)} \times \tanh(\mathbf{c}^{(n)})$$

RNN-BASED ARCHITECTURES

MULTI-LAYER RNN

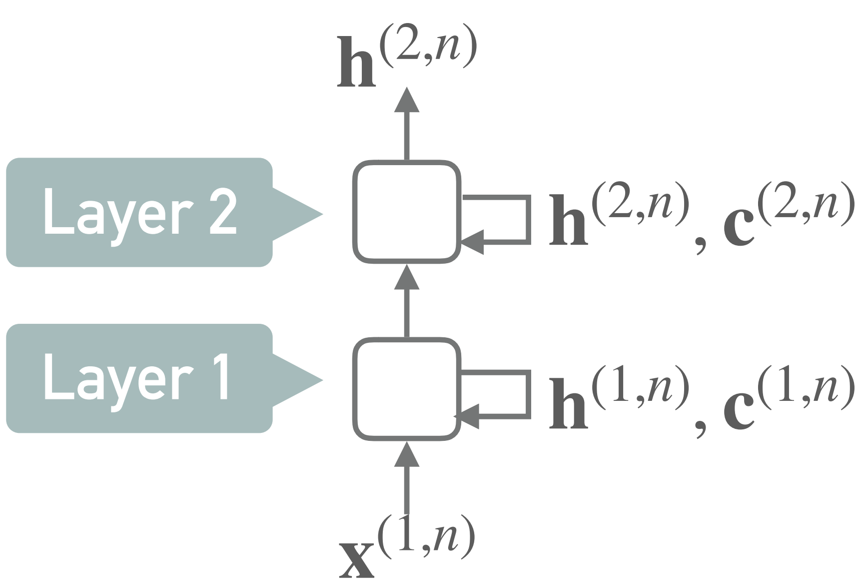
RNN with one layer

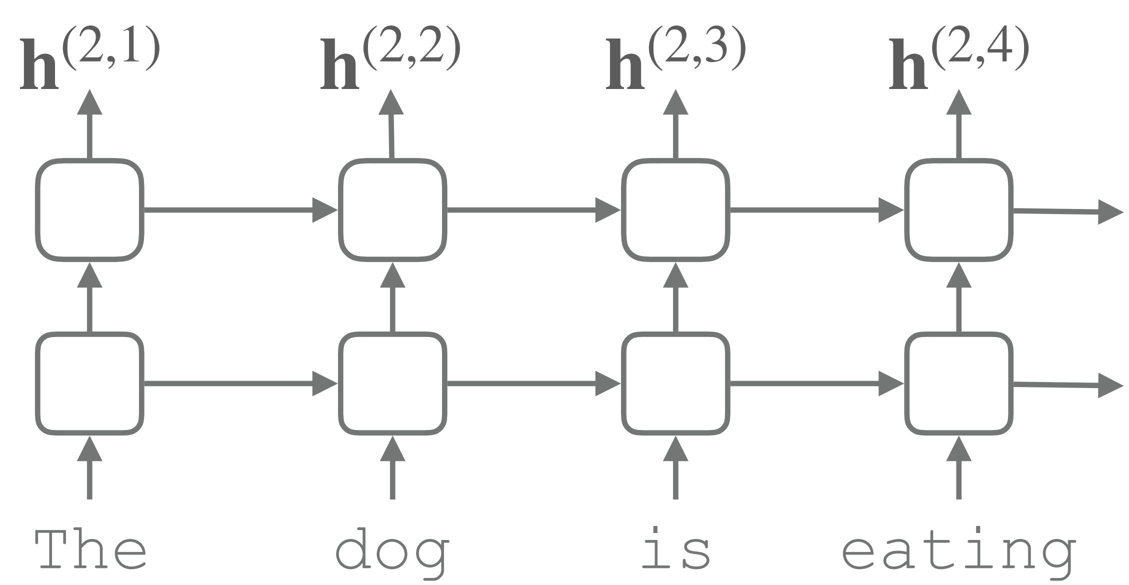




RNN with two layers

- Each layer as it own set of trainable parameters
- The recurrent connection is layer-dependent
- The input of layer n > 1 is the hidden representation at layer n





TAGGING WITH LSTMS

Part-of-speech tagging

PRP	**VB**	**DET**	**NN**
They	walk	the	dog

Named entity recognition

B-Per	**I-Per**	**O**	**O**	**B-Loc**
Neil	Armstrong	visited	the	moon

TAGGING WITH LSTMS

Part-of-speech tagging

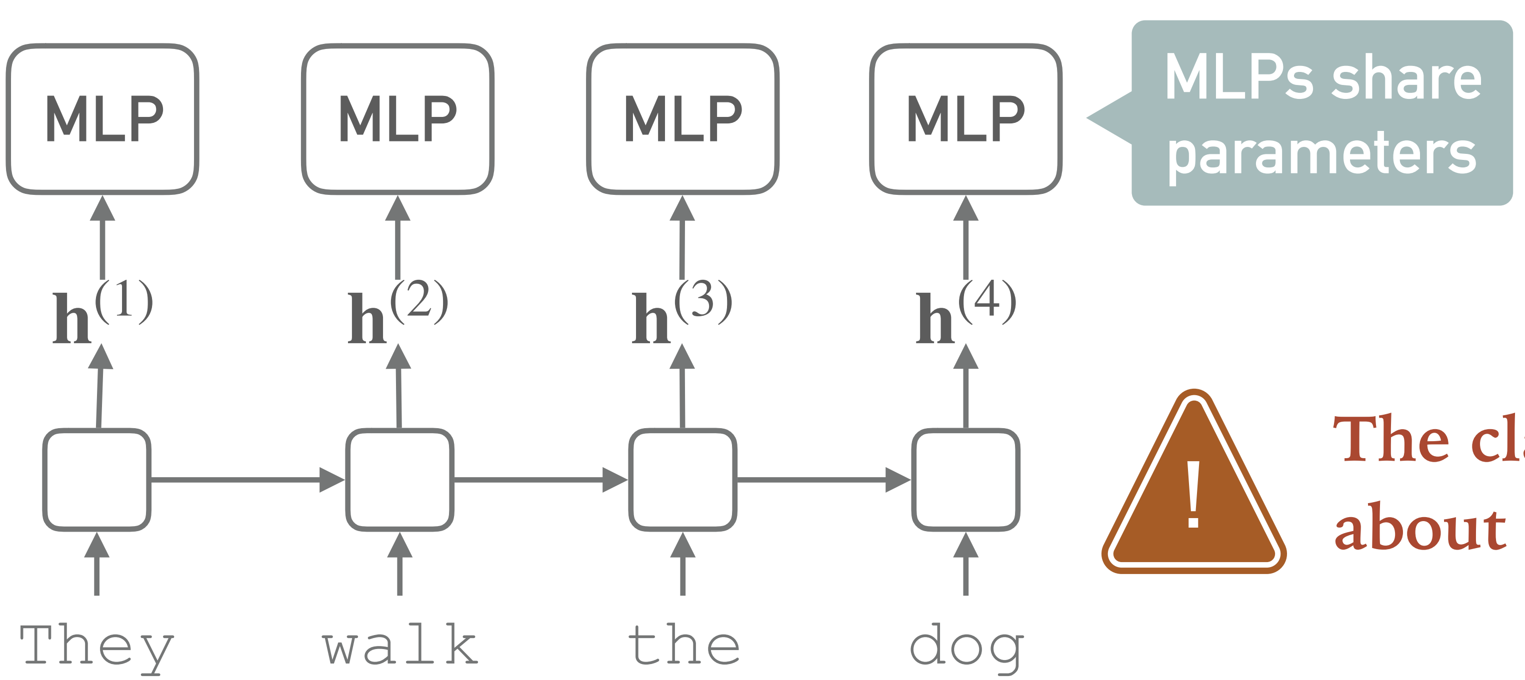
PRP	VB	DET	NN
They	walk	the	dog

Named entity recognition

B-Per	I-Per	O	O	B-Loc
Neil	Armstrong	visited	the	moon

Neural architecture

1. A RNN computes a context sensitive representation of each word
2. At each time step, the output of the RNN if fed to a MLP for classification



! The classifiers receive no information about context on the right of each word!

BIRNN

Intuition

Use two RNNs with **different trainable parameters**:

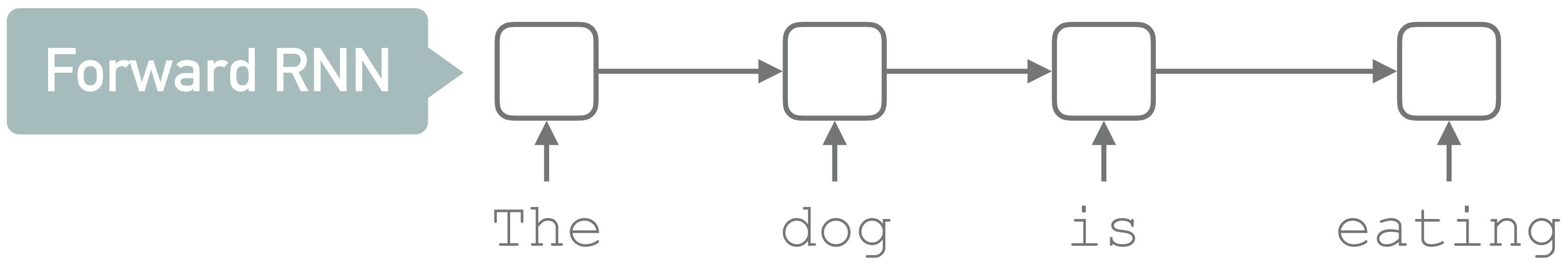
- Forward RNN: visit the sentence from left to right
- Backward RNN: visit the sentence from right to left

BIRNN

Intuition

Use two RNNs with **<u>different trainable parameters</u>**:

- Forward RNN: visit the sentence from left to right
- Backward RNN: visit the sentence from right to left

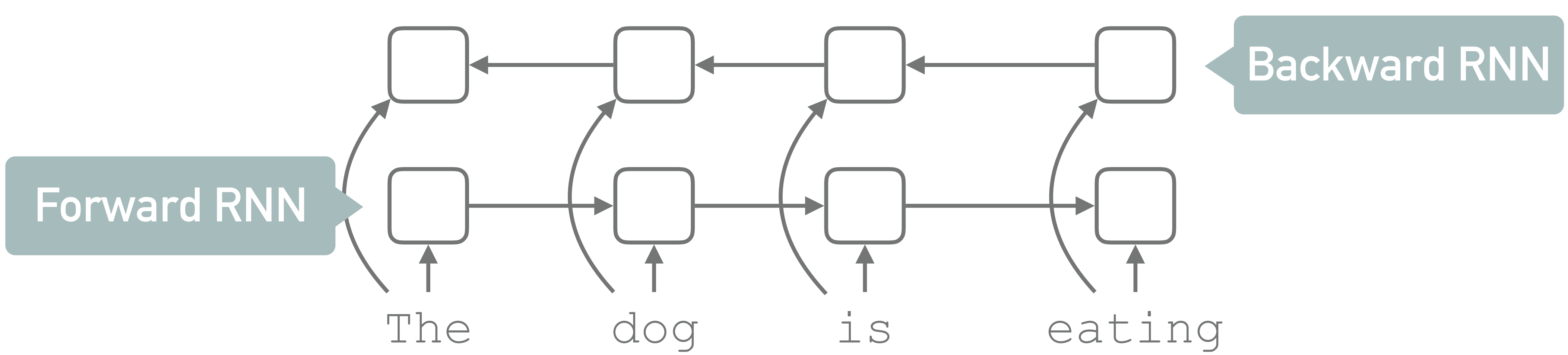


BIRNN

Intuition

Use two RNNs with **<u>different trainable parameters</u>**:

- Forward RNN: visit the sentence from left to right
- Backward RNN: visit the sentence from right to left

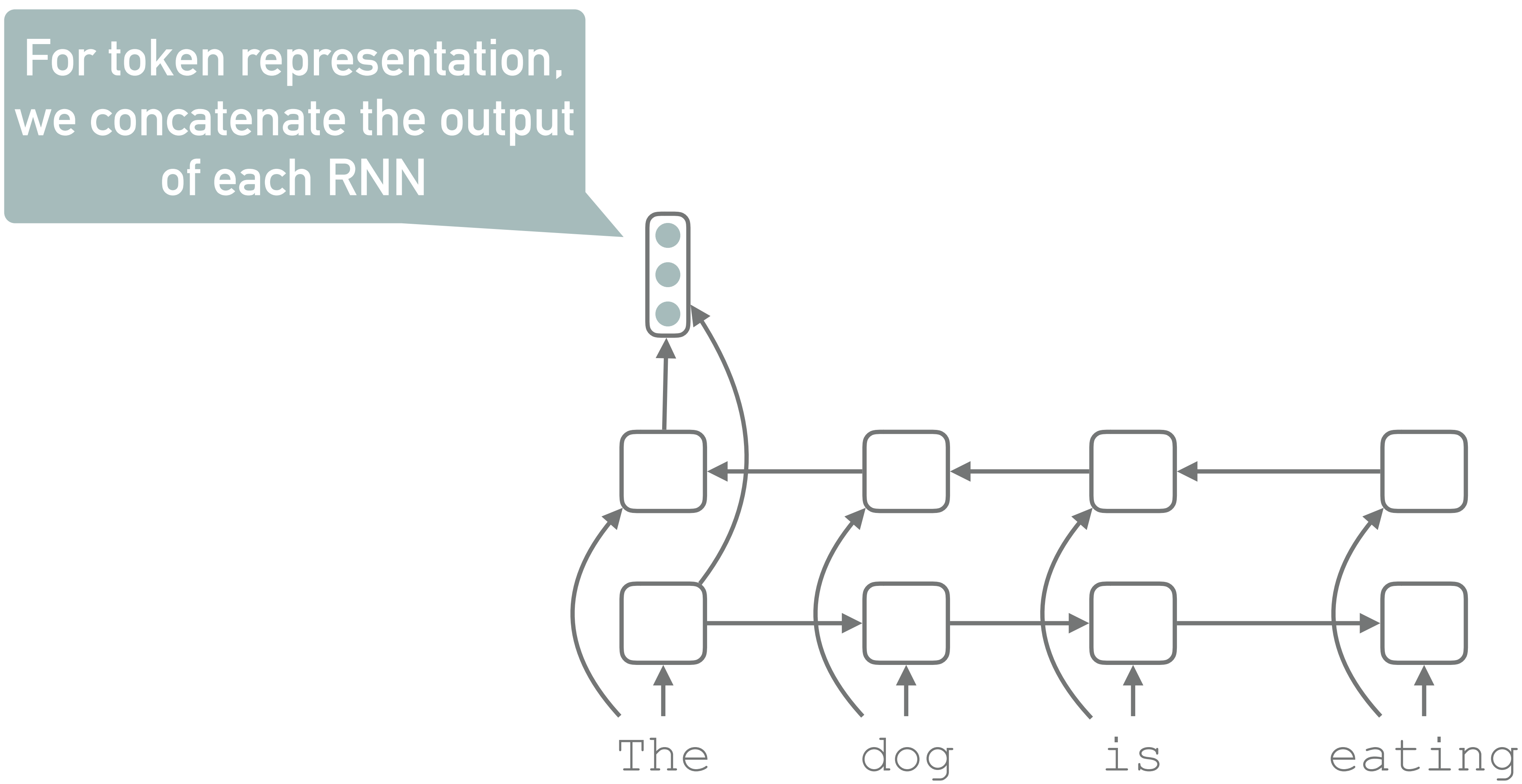


BIRNN

Intuition

Use two RNNs with **<u>different trainable parameters</u>**:

- ➤ Forward RNN: visit the sentence from left to right
- ➤ Backward RNN: visit the sentence from right to left



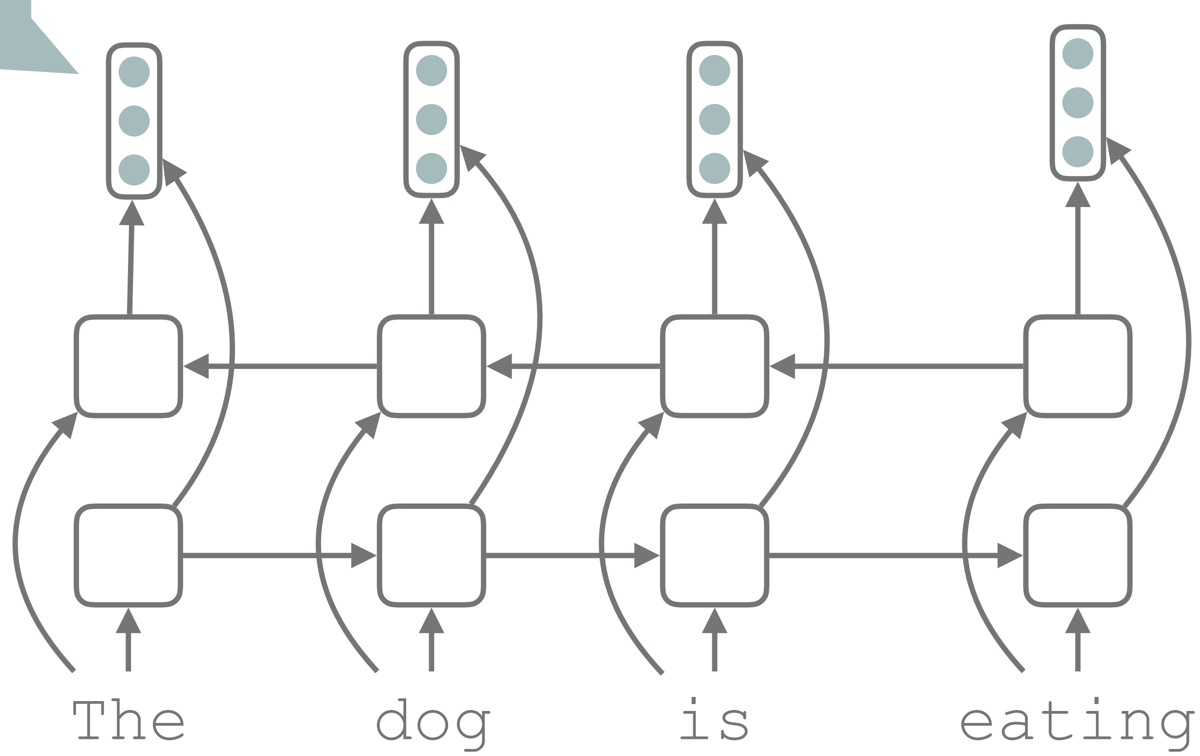
BIRNN

Intuition

Use two RNNs with **<u>different trainable parameters</u>**:

- Forward RNN: visit the sentence from left to right
- Backward RNN: visit the sentence from right to left

For token representation, we concatenate the output of each RNN

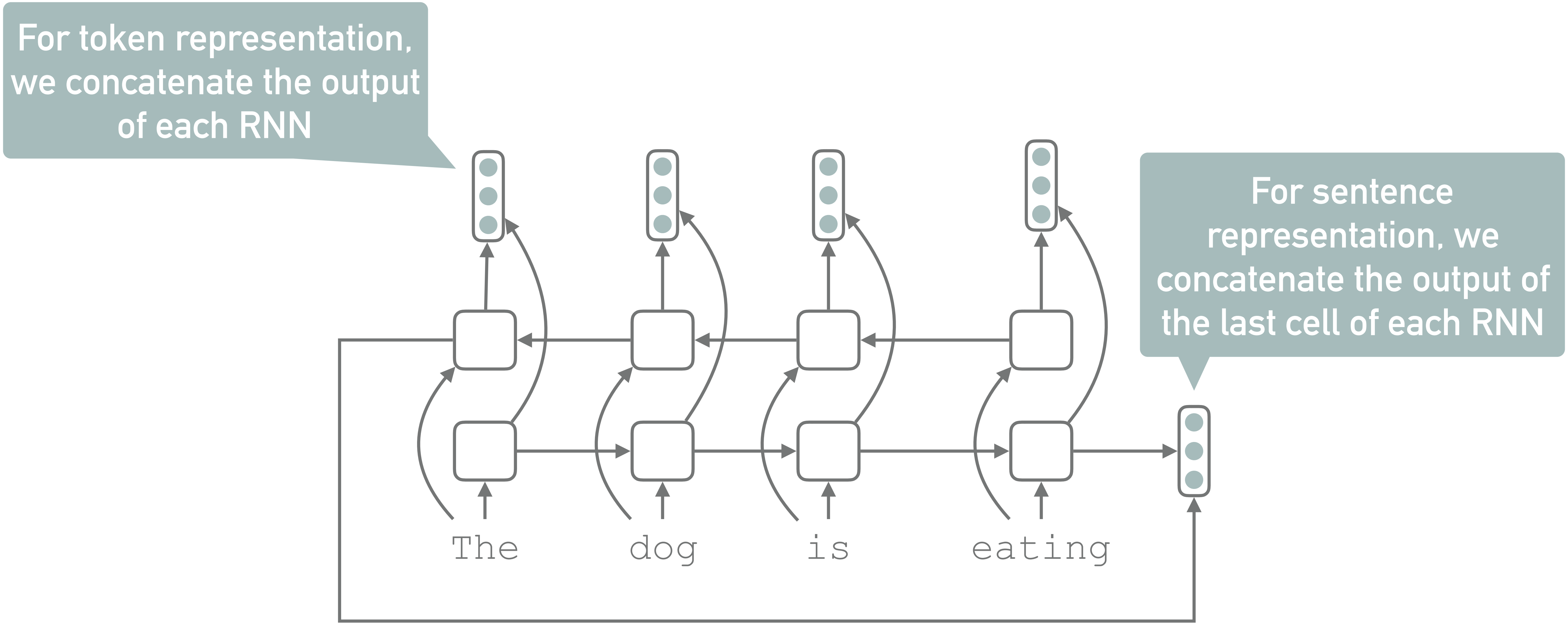


BIRNN

Intuition

Use two RNNs with **<u>different trainable parameters</u>**:

- Forward RNN: visit the sentence from left to right
- Backward RNN: visit the sentence from right to left



MULTI-STACK BIRNN

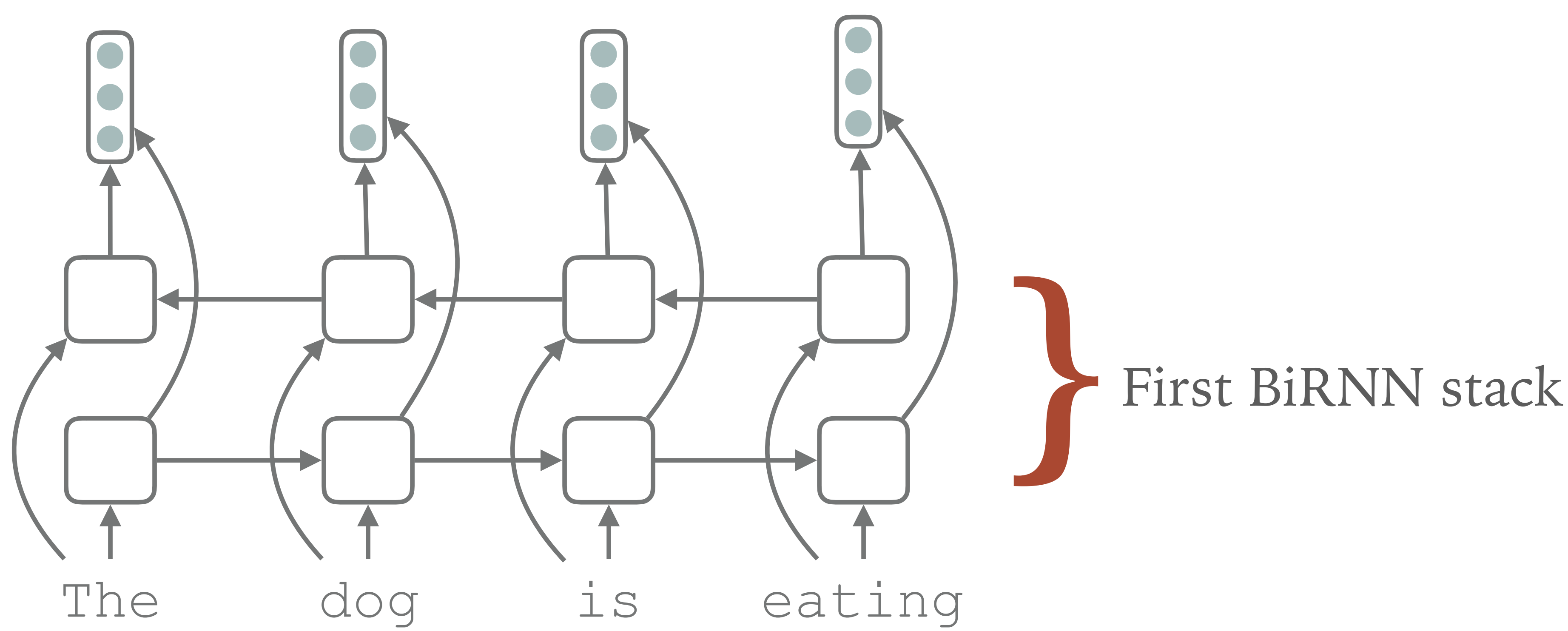
Intuition

Multi-layer RNNs have information only about previous words 👎

MULTI-STACK BIRNN

Intuition

Multi-layer RNNs have information only about previous words



MULTI-STACK BIRNN

Intuition

Multi-layer RNNs have information only about previous words