# CSCE 689: Special Topics in Trustworthy NLP

## Lecture 4: Convolutional Neural Network, Recurrent Neural Network

Kuan-Hao Huang
khhuang@tamu.edu

(Some slides adapted from Chris Manning, Abigail See, Karthik Narasimhan, and Danqi Chen)

# LaTeX Assignment

- LaTeX Assignment (1%)
- Due: Sep 11, 11:59pm

## CSCE 689: LaTeX Assignment

**Your Name**
Your UID and email

### Overview

This assignment is designed to give you practice with LaTeX, which you are expected to use for your literature review, project proposal, and final report in this course.

### Instructions

For this assignment, you will create a PDF containing your answers using LaTeX. If this is your first time working with LaTeX, we recommend starting with this short tutorial, which covers the basic features you will need for this course. Please use the Association for Computational Linguistics LaTeX template (link), a template widely used in major NLP conferences. We suggest using Overleaf as your online editor, since it automatically manages packages for you.

By default, the template is set to *review mode*. To switch to *final mode*, change:

Review Mode (Default)

```
\usepackage[review]{acl}
```

to:

Final Mode

```
\usepackage[final]{acl}
```

This allows you to display the author information. Be sure to include your name, UIN, and email.

The following sections contain questions on some commonly used LaTeX commands. There are a total of 100 points for this assignment. Please answer each question in a separate *section*, and submit the final PDF generated using LaTeX.

### 1 Including Equations [20pts]

Typeset the following expression using LaTeX:

$$\frac{\partial \mathcal{L}_{\text{total}}}{\partial \mathbf{w}_j} = -\frac{1}{m}\sum_{i=1}^{m}\big(y_i - \sigma(z_i)\big) \cdot \mathbf{x}_{i,j}$$

### 2 Including Images [20pts]

Select a picture of a cat and include it with a caption. The figure below is provided as an example.

Figure 1: This is a cute cat!

### 3 Including Tables [20pts]

Create a table that displays your name, UIN, and email. You can follow the example below as a template.

| | |
|---|---|
| Name | Kuan-Hao Huang |
| UIN | 123456789 |
| Email | khhuang@tamu.edu |

Table 1: Example table.

### 4 Including Lists [20pts]

Create a list that displays your name, UIN, and email. You can follow the example below as a template.

- Name: Kuan-Hao Huang
- UIN: 123456789
- Email: khhuang@tamu.edu

### 5 Including Citations [20pts]

Use *BibTex* to include the following paper: Paper 1 (Vaswani et al., 2017) and Paper 2 (Devlin et al., 2019). You can learn more about *BibTex* here.

# Topic Sign-Up

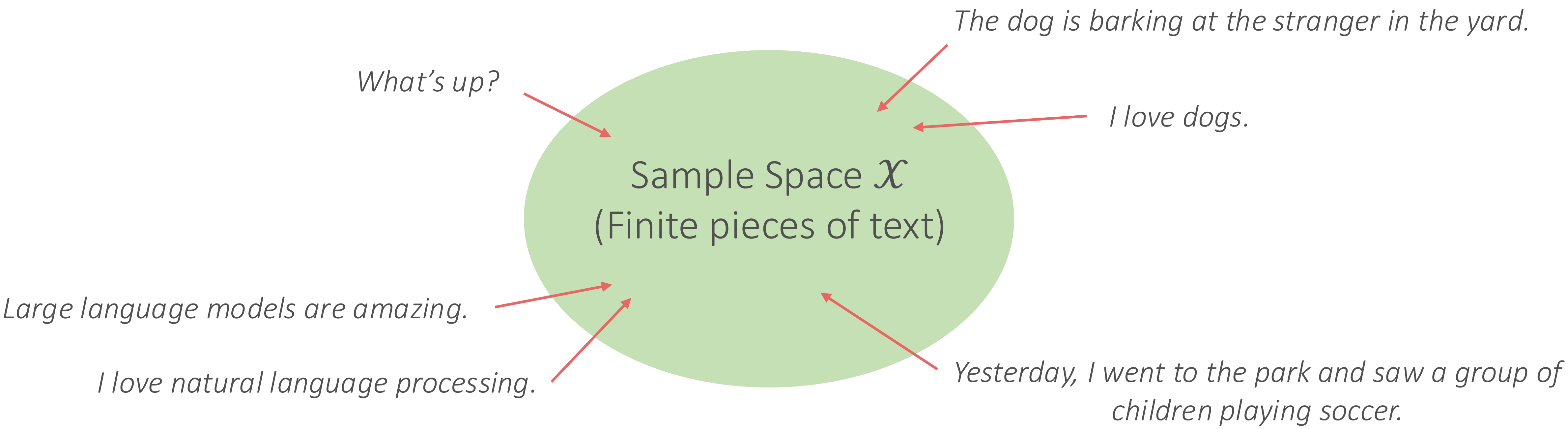
- Sign-up: https://tinyurl.com/2p9mr2wa
  - Log in with TAMU account
  - Due: Sep 10 before lecture

| Put Preference with Topic IDs | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Team | Member 1 | Member 2 (optional) | Preference 1 | Preference 2 | Preference 3 | Preference 4 | Preference 5 | Preference 6 | Preference 7 | Preference 8 | Preference 9 |
| Example | First_name Last_name | First_name Last_name | 4 | 10 | 1 | 7 | 3 | 9 | 8 | 6 | 12 |
| 1 | Kunal Jain | | 3 | 12 | 8 | 6 | 11 | 10 | 9 | | |
| 2 | Muhan Gao | | 5 | 6 | 4 | 9 | 10 | 7 | 8 | 11 | 12 |
| 3 | Serhii Honcharenko | | 9 | 10 | 4 | 7 | 6 | 1 | | | |
| 4 | Oscar Chew | | 9 | 3 | 11 | 7 | 8 | 10 | 4 | 2 | 1 |
| 5 | Junggeun Do | | 8 | 7 | 11 | 10 | 2 | 6 | 12 | 11 | 5 |
| 6 | Jiongran Wang | | 3 | 9 | 10 | 6 | 4 | 2 | 7 | 5 | 1 |
| 7 | Sicong Liang | | 10 | 9 | 3 | 13 | 7 | | | | |
| 8 | Kowsalya Balamuralei Umamaheswari | | 3 | 10 | 12 | 8 | | | | | |
| 9 | Yi Wen | | 12 | 4 | 7 | 9 | 6 | 8 | | | |
| 10 | Quang Nguyen | | 6 | 3 | 4 | 12 | 2 | 1 | 8 | 9 | 7 |
| 11 | Aaron Xu | | 4 | 8 | 7 | 3 | 9 | 6 | 11 | 2 | 1 |
| 12 | Bhaskar Ruthvik Bikkina | | 7 | 11 | 12 | 9 | 10 | 5 | 3 | 4 | 6 |
| 13 | | | | | | | | | | | |
| 14 | | | | | | | | | | | |
| 15 | | | | | | | | | | | |
| 16 | | | | | | | | | | | |
| 17 | | | | | | | | | | | |

# Recap: Language Models

- Learn the probability distribution over texts $x = [w_1, w_2, \ldots, w_l] \in \mathcal{X}$

$$P(x) = P(w_1, w_2, \ldots, w_l)$$



# Recap: Auto-Regressive Language Models

$$P(w_1, w_2, w_3, \dots, w_l) = P(w_1)P(w_2, w_3, \dots, w_l | w_1)$$

$$= P(w_1)P(w_2|w_1)(w_3, \dots, w_l | w_1, w_2)$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)(w_4, \dots, w_l | w_1, w_2, w_3)$$

$$= \prod_{i=1}^{l} P(w_i | w_1, w_2, \dots, w_{i-1})$$

$P(\textit{She likes to go hiking}) = P(\textit{She}) \cdot P(\text{likes}|\text{She}) \cdot P(\text{to}|\text{She likes})$
$\cdot\, P(\text{go}|\text{She likes to}) \cdot P(\text{hiking}|\text{She likes to go})$

# Recap: N-Gram Language Models

Assumption: $P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(w_i|w_{i-n+1}, \dots, w_{i-1})$

$$P(w_1, w_2, w_3, \dots, w_l) \approx \prod_i P(w_i|w_{i-n+1}, \dots, w_{i-1})$$

The prediction of the next token depends on the previous **n** tokens

A count-based solution: Collect training corpus and count

$$P(w_i|w_{i-n+1}, \dots, w_{i-1}) = \frac{C_{train}(w_{i-n+1}, \dots, w_{i-1}, w_i)}{C_{train}(w_{i-n+1}, \dots, w_{i-1})}$$

# Recap: What Can Language Models Do?

- Score texts

$P(\textit{The dog is barking at the stranger in the yard.})$ → High

$P(\textit{Cats upon they chairs sleeping their dreams fall.})$ → Low

Compute perplexity

- Generate texts

$\tilde{x} \sim P(\mathcal{X})$

Sample from word distribution

# N-Gram Language Models

Assumption: $P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(w_i|w_{i-n+1}, \dots, w_{i-1})$

$$P(w_1, w_2, w_3, \dots, w_l) \approx \prod_i P(w_i|w_{i-n+1}, \dots, w_{i-1})$$

The prediction of the next token depends on the previous **n** tokens

A count-based solution: Collect training corpus and count

$$P(w_i|w_{i-n+1}, \dots, w_{i-1}) = \frac{C_{train}(w_{i-n+1}, \dots, w_{i-1}, w_i)}{C_{train}(w_{i-n+1}, \dots, w_{i-1})}$$

Any Problems?

# Unseen Patterns in Training Corpus

- Not all n-grams in the test set will be observed in training corpus
- Training corpus
  - I like apples
  - I love oranges
- Test set
  - I love apples
  - I like oranges
- This problem becomes severe when n is large

# Laplace Smoothing

- Handle sparsity by making sure all probabilities are non-zero in our model
  - Just add $\alpha$ to all counts and renormalize

Bigram language model before smoothing

$$P(w_i|w_{i-1}) = \frac{C_{train}(w_{i-1}, w_i)}{C_{train}(w_{i-1})}$$

Bigram language model after smoothing

$$P(w_i|w_{i-1}) = \frac{C_{train}(w_{i-1}, w_i) + \alpha}{C_{train}(w_{i-1}) + \alpha|\mathcal{V}|}$$

# Linear Interpolation

$$\hat{P}(w_i|w_{i-1}, w_{i-2}) = \lambda_1 P(w_i|w_{i-1}, w_{i-2})$$ Trigram

$$+\lambda_2 P(w_i|w_{i-1})$$ Bigram

$$+\lambda_3 P(w_i)$$ Unigram

$$\sum_i \lambda_i = 1$$

Strong empirical performance!

# N-Gram Language Models

Assumption: $P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(w_i|w_{i-n+1}, \dots, w_{i-1})$

$$P(w_1, w_2, w_3, \dots, w_l) \approx \prod_i P(w_i|w_{i-n+1}, \dots, w_{i-1})$$

The prediction of the next token depends on the previous **n** tokens

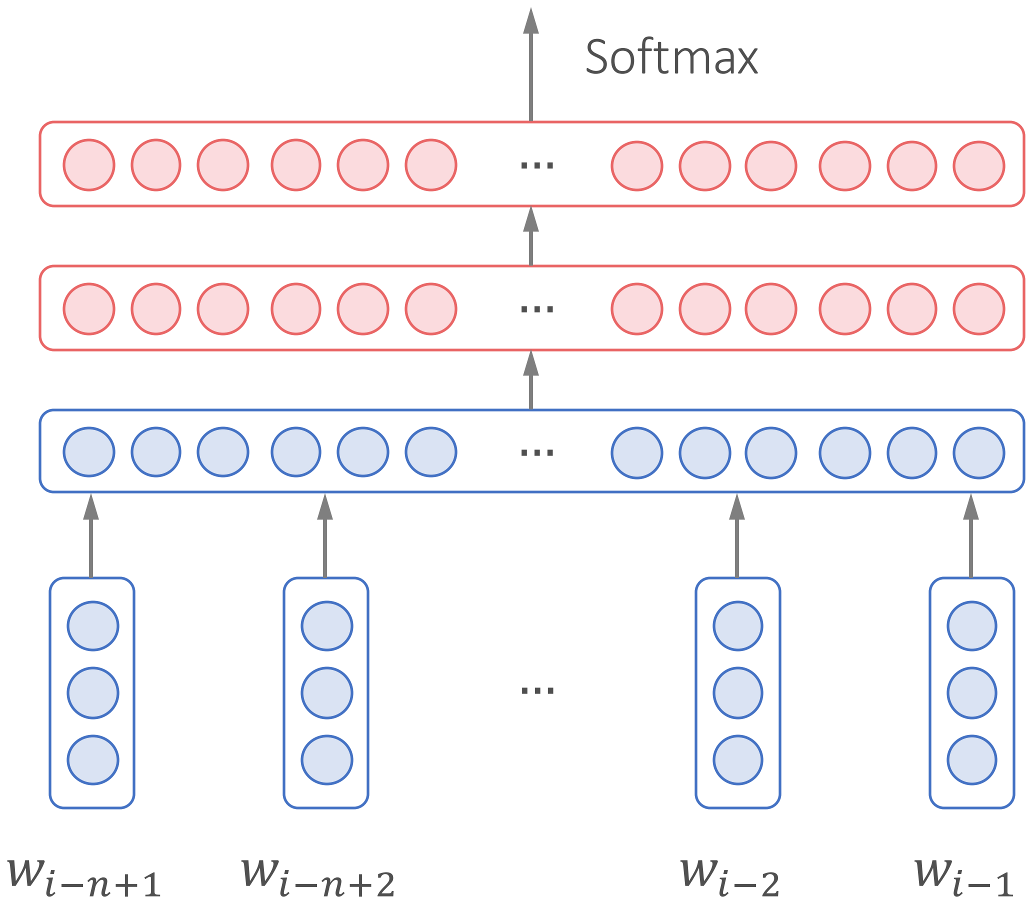
A count-based solution: Collect training corpus and count

$$P(w_i|w_{i-n+1}, \dots, w_{i-1}) = \frac{C_{train}(w_{i-n+1}, \dots, w_{i-1}, w_i)}{C_{train}(w_{i-n+1}, \dots, w_{i-1})}$$

Can we compute the probability in a different way?

# Neural Language Models

$$P(w_i | w_{i-n+1}, \ldots, w_{i-1})$$




Training corpus

- I like apples
- I love oranges

Test set

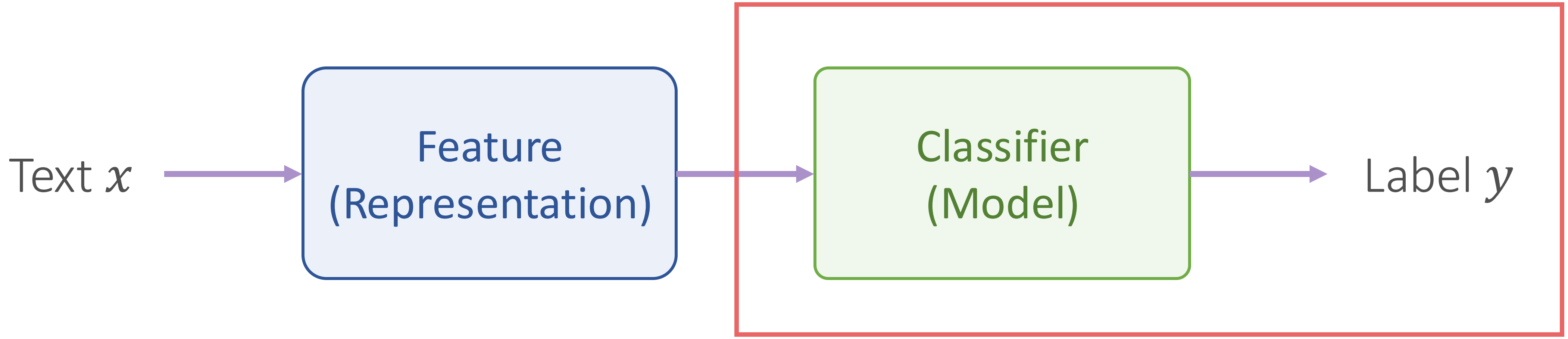
- I love apples
- I like oranges

# Auto-Regressive Language Models

$$
\begin{aligned}
P(w_1, w_2, w_3, \dots, w_l) &= P(w_1)P(w_2, w_3, \dots, w_l | w_1) \\
&= P(w_1)P(w_2|w_1)(w_3, \dots, w_l | w_1, w_2) \\
&= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)(w_4, \dots, w_l | w_1, w_2, w_3) \\
&= \prod_{i=1}^{l} P(w_i | w_1, w_2, \dots, w_{i-1})
\end{aligned}
$$

$$
\begin{aligned}
P(\textit{She likes to go hiking}) = {} & P(\textit{She}) \cdot P(\text{likes}|\text{She}) \cdot P(\text{to}|\text{She likes}) \\
& \cdot P(\text{go}|\text{She likes to}) \cdot P(\text{hiking}|\text{She likes to go})
\end{aligned}
$$

# Recap: A General Framework for Text Classification




- Teach the model how to make prediction $y$
- Logistic regression, neural networks, CNN, RNN, LSTM, Transformers

# Recap: Logistic Regression

- Logistic Regression for multiclass classification

Feature Vector $\mathbf{x} = [x_1, x_2, x_3, \dots, x_d]$ Label $y = 0,1, \dots, C-1$

$\boxed{\text{Weight Vectors } \mathbf{w}_c = [w_{c,1}, w_{c,2}, w_{c,3}, \dots, w_{c,d}] \quad \text{Bias } b_c}$

Learnable Parameters

$$z_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

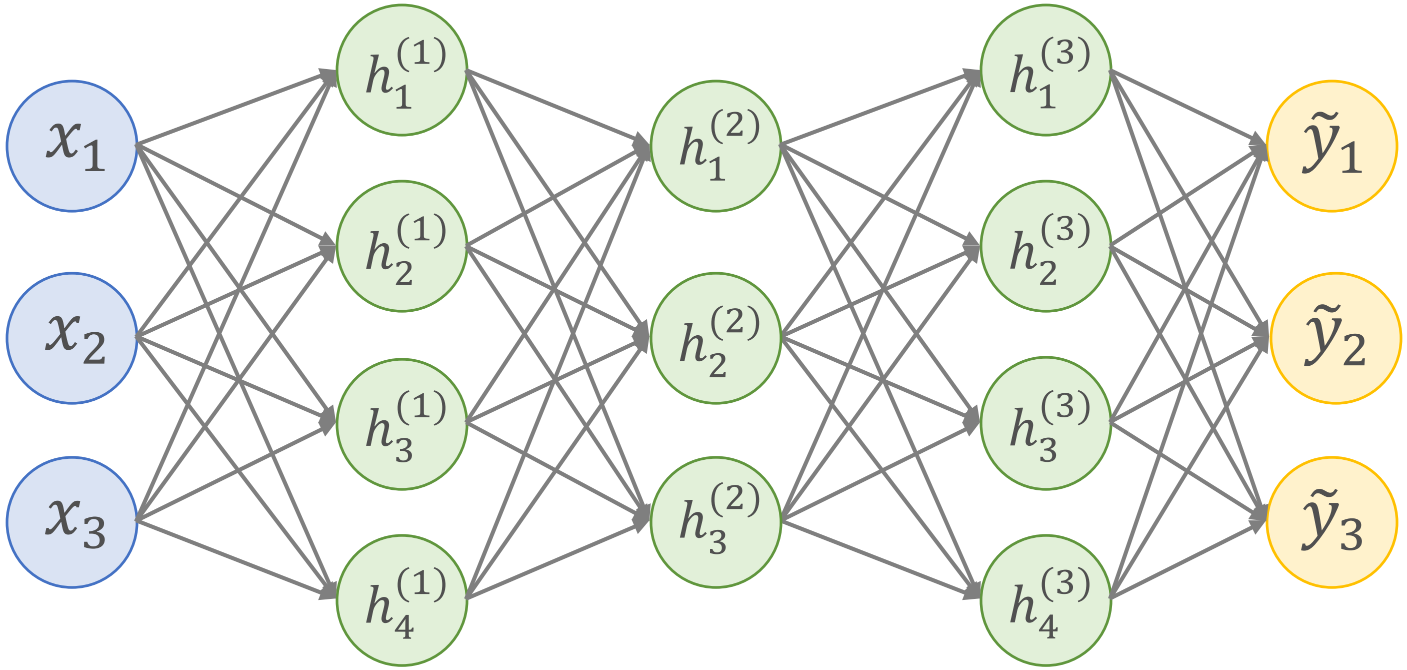
$$P(y = c |\, \mathbf{x}) = \text{softmax}(z_c)$$

$$\text{softmax}(z_c) = \frac{e^{z_c}}{\sum_t e^{z_t}}$$

Softmax Function

$$\text{Prediction} = \arg\max_{c} P(y = c |\, \mathbf{x})$$

# Recap: Neural Networks



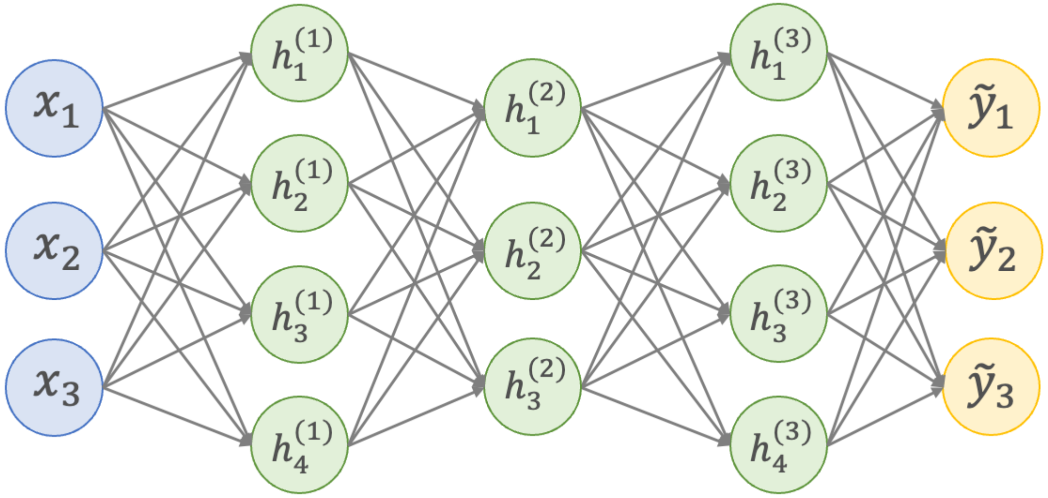

Prediction $= \arg\max_{\mathbf{c}} \tilde{y}_c$

Multiclass Cross Entropy Loss

$$\mathcal{L}_{CE}(y, \tilde{y}) = -\sum_{c=0}^{C} y_c \log P(y = c \mid \mathbf{x})$$

# A Simple Approach: Averaged Embeddings + DNN

| | Alice | treats | Bob | well | |
|---|---|---|---|---|---|
| Dimension 1 | 0.7 | 2.7 | -0.1 | -5.7 | -0.6 |
| Dimension 2 | 8.6 | -3.9 | 6.7 | -9.8 | 0.4 |
| Dimension 3 | -2.4 | -5.6 | 1.5 | -1.6 | -1.6 |
| Dimension 4 | 2.3 | 1.1 | 2.0 | -1.0 | 1.1 |



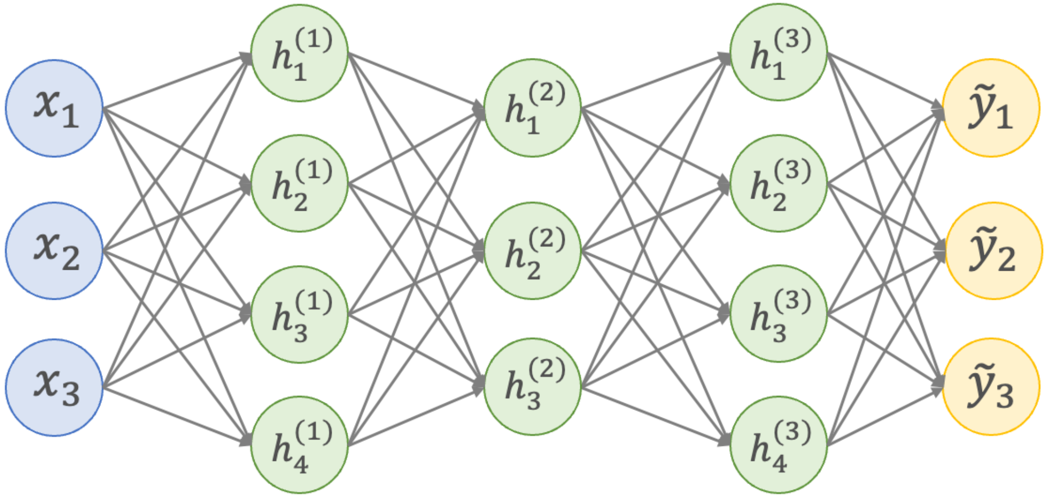

$$\mathcal{L}_{CE}(y, \tilde{y}) = -\sum_{c=0}^{C} y_c \log P(y = c \mid \mathbf{x})$$

Any problems?

# A Simple Approach: Concatenated Embeddings + DNN

| | Alice | treats | Bob | well |
|---|---|---|---|---|
| Dimension 1 | 0.7 | 2.7 | -0.1 | -5.7 |
| Dimension 2 | 8.6 | -3.9 | 6.7 | -9.8 |
| Dimension 3 | -2.4 | -5.6 | 1.5 | -1.6 |
| Dimension 4 | 2.3 | 1.1 | 2.0 | -1.0 |

0.7
8.6
-2.4
2.3
2.7
-3.9
-5.6
1.1
...
-5.7
-9.8
-1.6
-1.0




$$\mathcal{L}_{CE}(y, \tilde{y}) = -\sum_{c=0}^{C} y_c \log P(y = c |\, \mathbf{x})$$

Any problems?

# Challenges

- Averaged Embeddings
  - Lose order information
- Concatenated Embeddings
  - Cannot handle various lengths

# Solution: Capture Local Order Information

*Bob likes Alice very much*

| | |
|---|---|
| **Unigram** | *{Bob, likes, Alice, very, much}* |
| **Bigram** | *{Bob likes, likes Alice, Alice very, very much}* |
| **Trigram** | *{Bob likes Alice, likes Alice very, Alice very much}* |
| **4-gram** | *{Bob likes Alice very, likes Alice very much}* |

We can infer global order information from local order information

# Convolutional Neural Network (CNN)

- Capture local features (N-grams)
  - Filters (Kernels)
- Hierarchical feature learning
  - Multiple layers

# Convolutional Neural Network (CNN)

Learnable Weight (Filter)
Filter Size = 3

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix}$$

| | Alice | treats | Bob | well |
|---|---|---|---|---|
| Dimension 1 | 0.7 | 2.7 | -0.1 | -5.7 |
| Dimension 2 | 8.6 | -3.9 | 6.7 | -9.8 |
| Dimension 3 | -2.4 | -5.6 | 1.5 | -1.6 |
| Dimension 4 | 2.3 | 1.1 | 2.0 | -1.0 |

$w_{1,1}$*0.7 + $w_{1,2}$*2.7 + $w_{1,3}$*-0.1

$w_{2,1}$*8.6 + $w_{2,2}$*-3.9 + $w_{2,3}$*6.7

$w_{3,1}$*-2.4 + $w_{3,2}$*-5.6 + $w_{3,3}$*1.5

$w_{4,1}$*2.3 + $w_{4,2}$*1.1 + $w_{4,3}$*2.0

$v_{1,1}$
+
$v_{1,2}$
+
$v_{1,3}$
+
$v_{1,4}$

$v_1$

# Convolutional Neural Network (CNN)

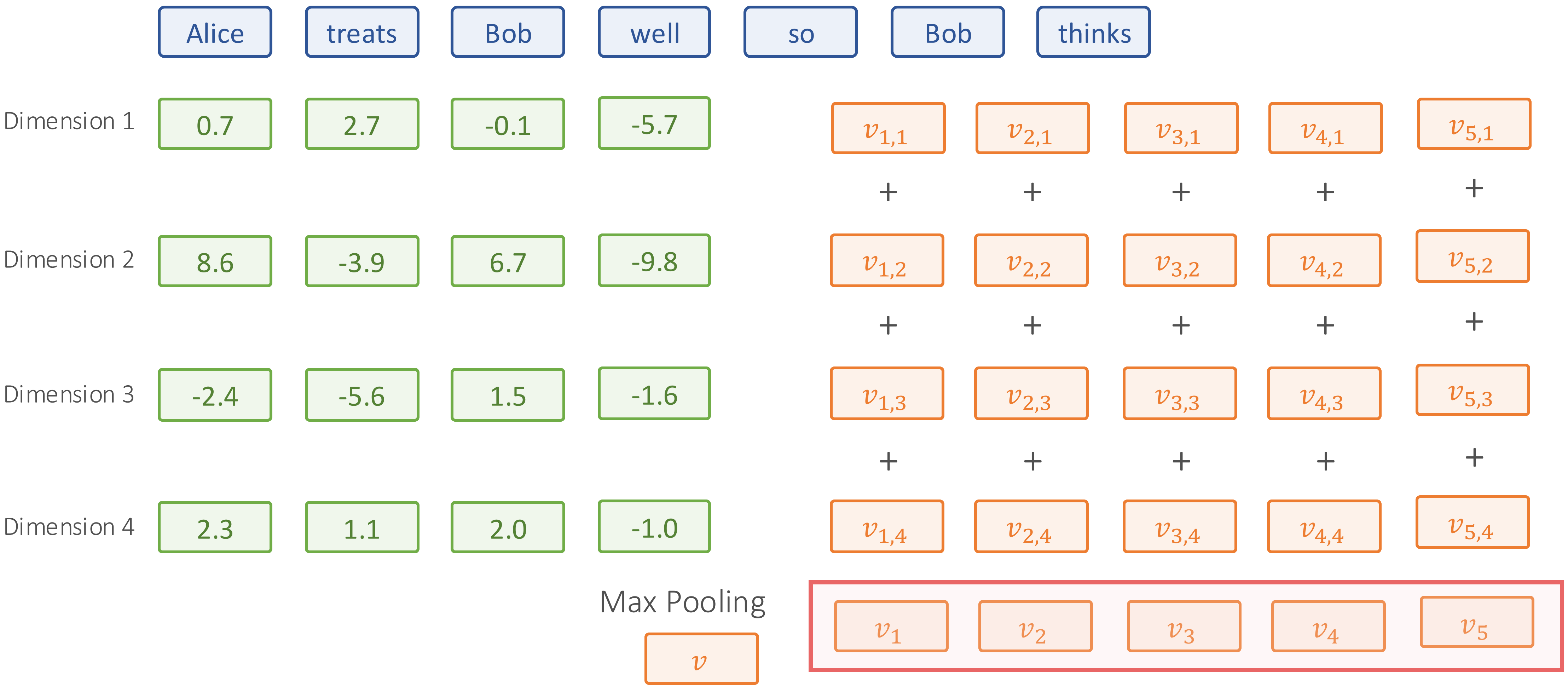
Learnable Weight (Filter)
Filter Size = 3

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix}$$

# Convolutional Neural Network (CNN)

Learnable Weight (Filter)
Filter Size = 3

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix}$$



# Convolutional Neural Network (CNN)

Learnable Weight (Filter)
Filter Size = 3

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix}$$

| | Alice | treats | Bob | well |
|---|---|---|---|---|
| Dimension 1 | 0.7 | 2.7 | -0.1 | -5.7 |
| Dimension 2 | 8.6 | -3.9 | 6.7 | -9.8 |
| Dimension 3 | -2.4 | -5.6 | 1.5 | -1.6 |
| Dimension 4 | 2.3 | 1.1 | 2.0 | -1.0 |

$v$ $v$ $v$

# Convolutional Neural Network (CNN)

Filter Size = 3 $\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix}$ Filter Size = 2 $\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} \\ \cdots & \cdots \\ w_{4,1} & w_{4,2} \end{bmatrix}$ Filter Size = 4 $\mathbf{W} = \begin{bmatrix} w_{1,1} & \cdots & w_{1,4} \\ \cdots & \cdots & \cdots \\ w_{4,1} & \cdots & w_{4,4} \end{bmatrix}$

| | Alice | treats | Bob | well |
|---|---|---|---|---|
| Dimension 1 | 0.7 | 2.7 | -0.1 | -5.7 |
| Dimension 2 | 8.6 | -3.9 | 6.7 | -9.8 |
| Dimension 3 | -2.4 | -5.6 | 1.5 | -1.6 |
| Dimension 4 | 2.3 | 1.1 | 2.0 | -1.0 |

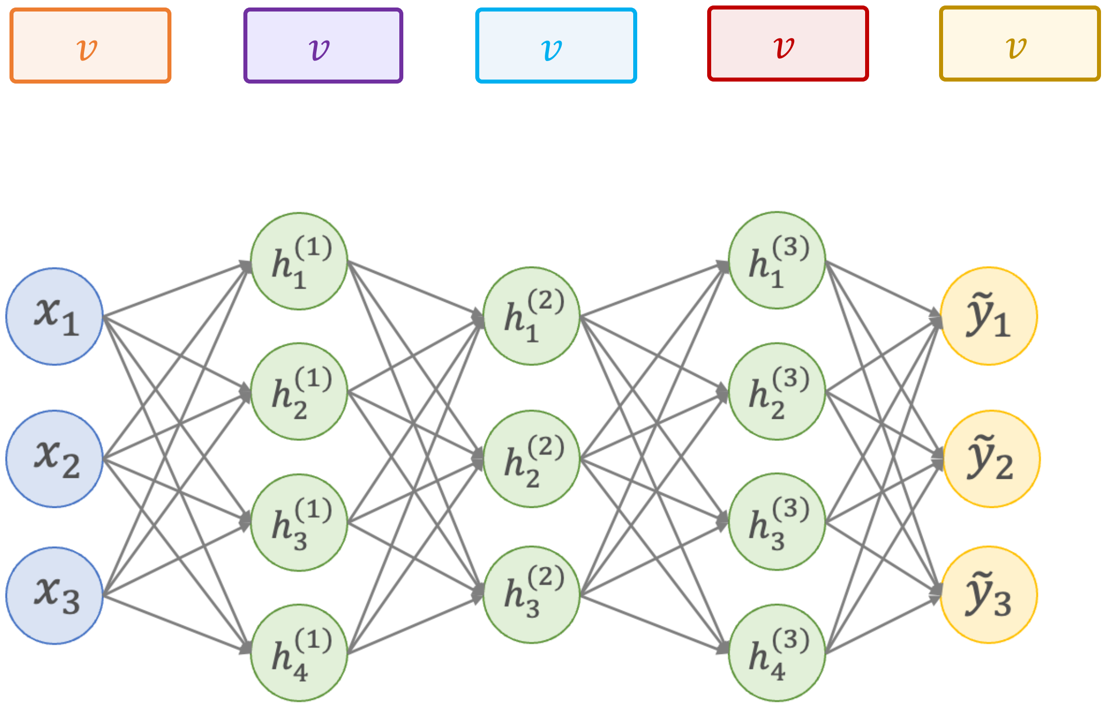
$w_{1,1}$*0.7 + $w_{1,2}$*2.7

$w_{2,1}$*8.6 + $w_{2,2}$*-3.9 + $w_{2,3}$*6.7 + $w_{2,4}$*-9.8

$v$ $v$ $v$

# Convolutional Neural Network (CNN)

| | Alice | treats | Bob | well |
|---|---|---|---|---|
| Dimension 1 | 0.7 | 2.7 | -0.1 | -5.7 |
| Dimension 2 | 8.6 | -3.9 | 6.7 | -9.8 |
| Dimension 3 | -2.4 | -5.6 | 1.5 | -1.6 |
| Dimension 4 | 2.3 | 1.1 | 2.0 | -1.0 |




$$\mathcal{L}_{CE}(y, \tilde{y}) = -\sum_{c=0}^{C} y_c \log P(y = c | \mathbf{x})$$

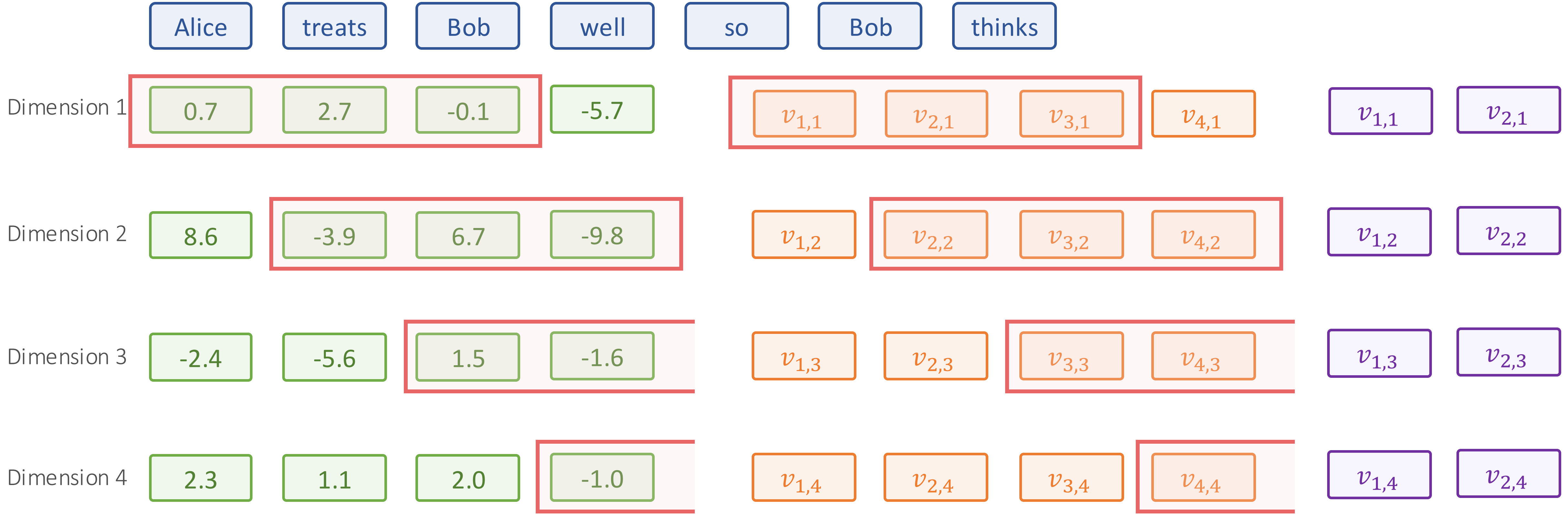
# From Single Layer to Multiple Layers

Learnable Weight (Layer 1)
Filter Size = 3

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix}$$

Learnable Weight (Layer 2)
Filter Size = 3

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix}$$



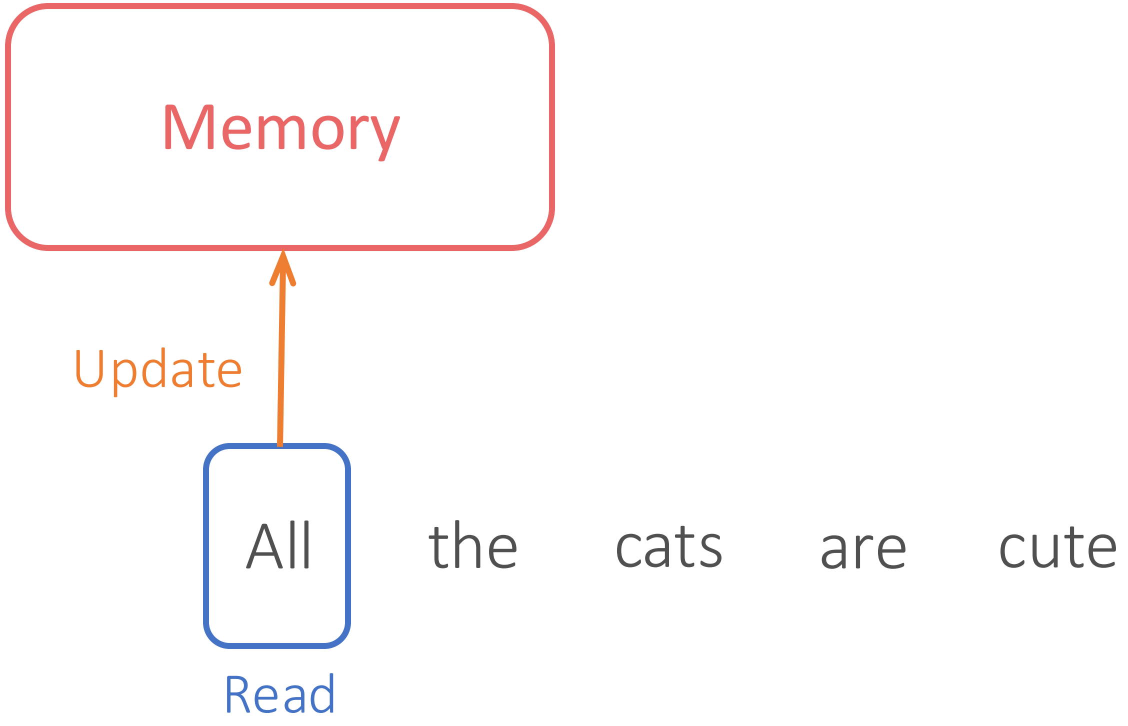

Capture high-order or hierarchical information

# Convolutional Neural Network (CNN)

- Capture local features (N-grams)
  - Filters (Kernels)
- Hierarchical feature learning
  - Multiple layers
- The whole process is still not similar to how human read texts
- Can we model reading texts in a sequential way?

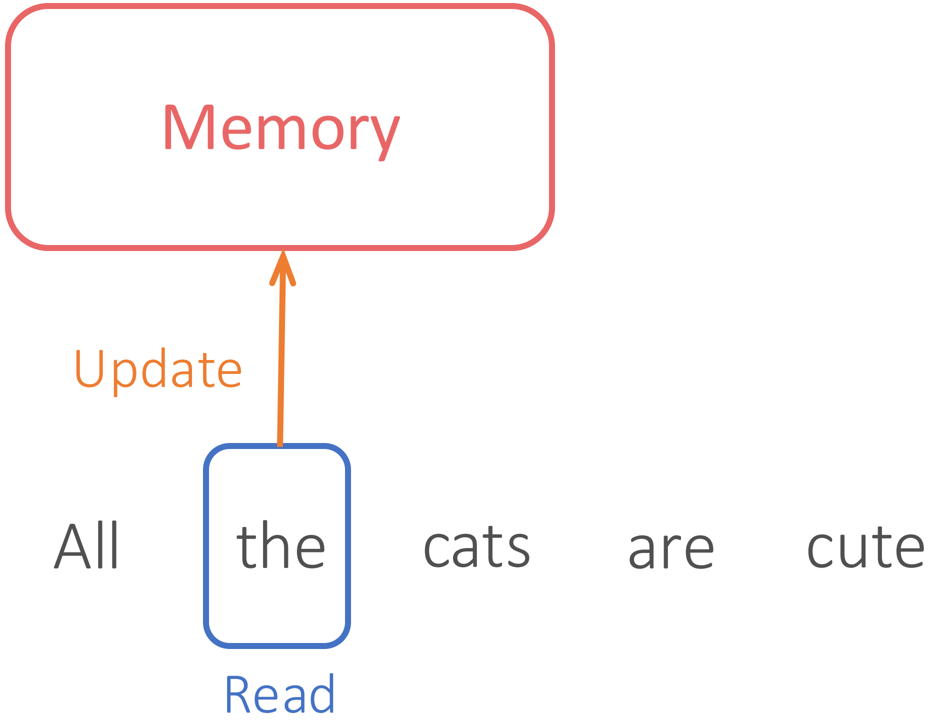
# Recurrent Neural Network (RNN)

- Read texts sequentially like a human with memory
  - Read → update memory



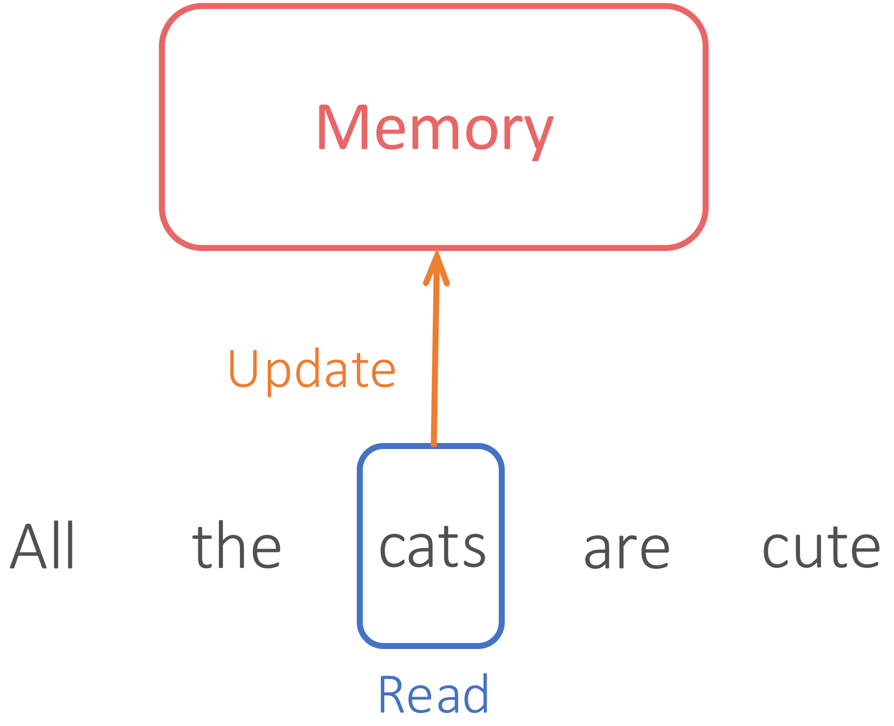
# Recurrent Neural Network (RNN)

- Read texts sequentially like a human with memory
  - Read → update memory



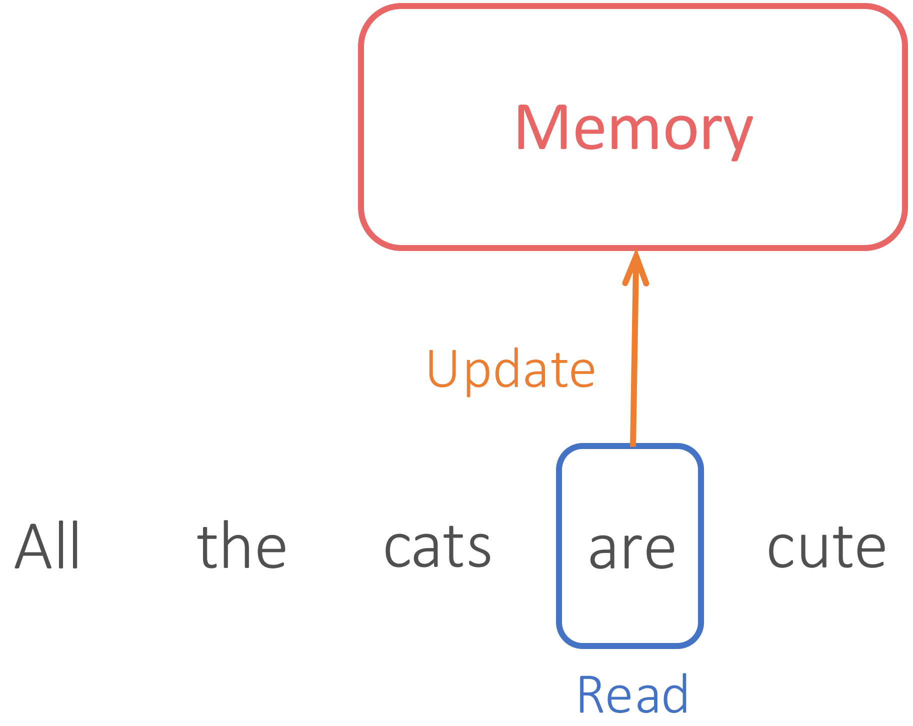
# Recurrent Neural Network (RNN)

- Read texts sequentially like a human with memory
  - Read → update memory



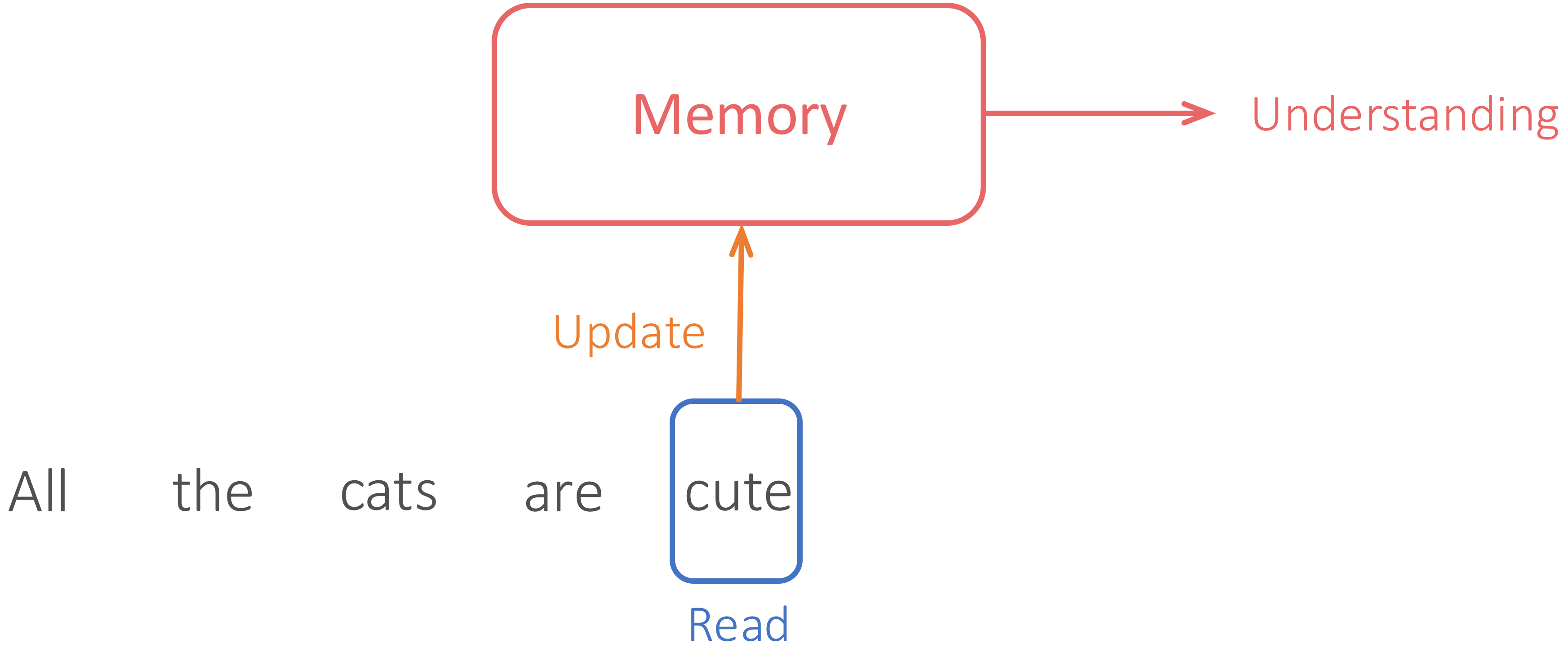
# Recurrent Neural Network (RNN)

- Read texts sequentially like a human with memory
  - Read → update memory



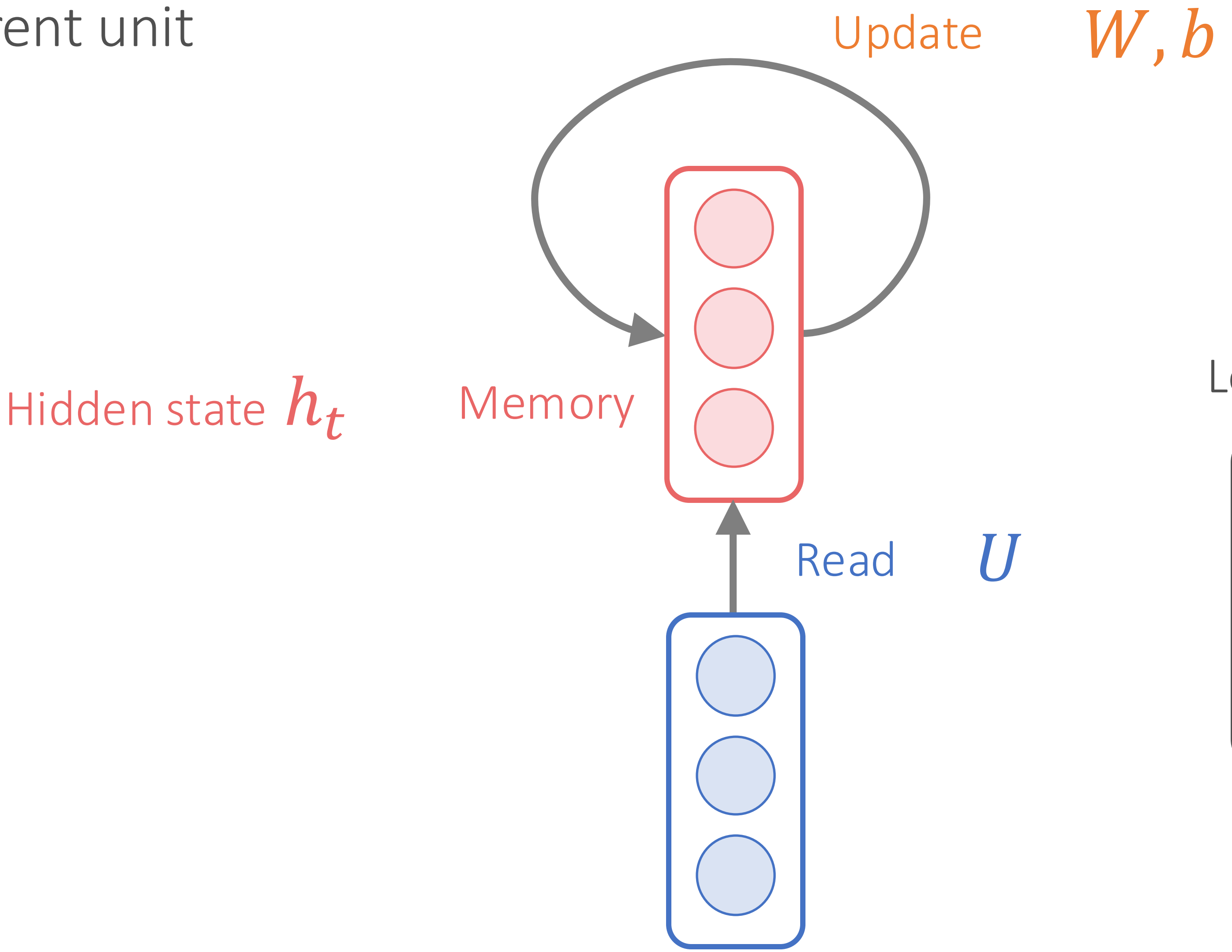
# Recurrent Neural Network (RNN)

- Read texts sequentially like a human with memory
  - Read → update memory

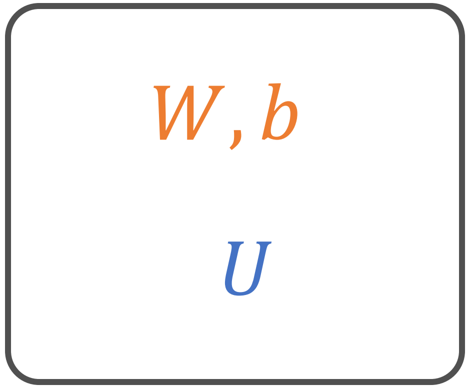


# Recurrent Neural Network (RNN)

- Recurrent unit




Learnable parameters



# Recurrent Neural Network (RNN)

Hidden States

Read

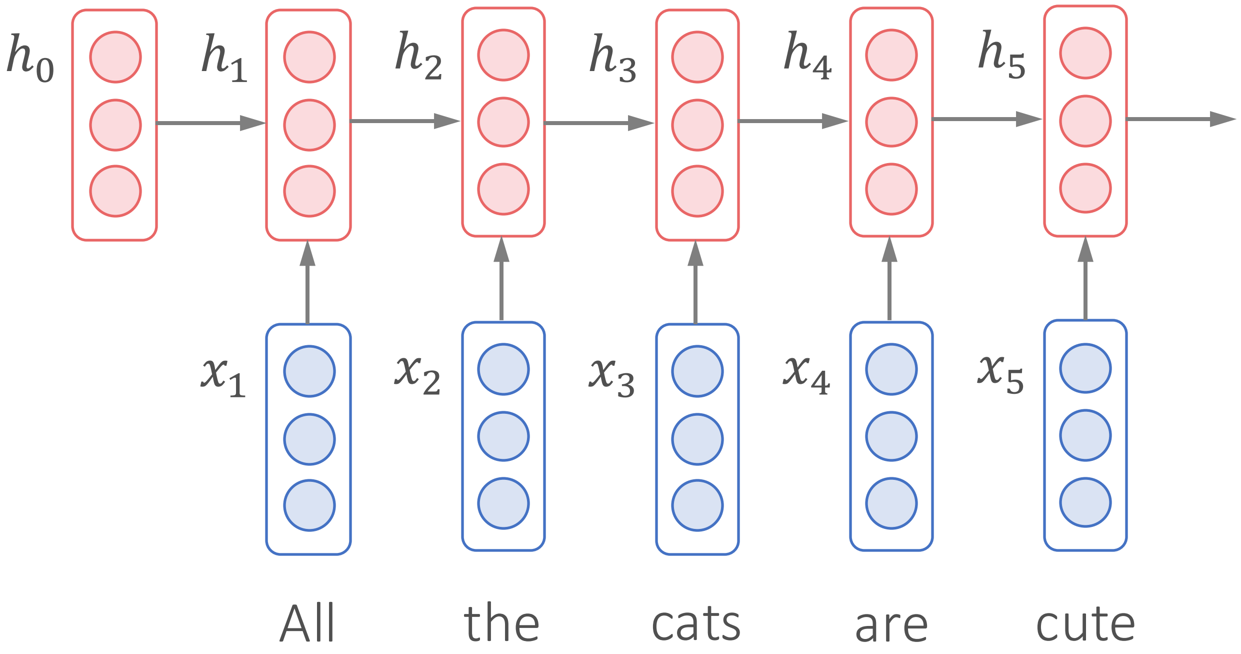
$$h_t = \sigma(W h_{t-1} + U x_t + b)$$

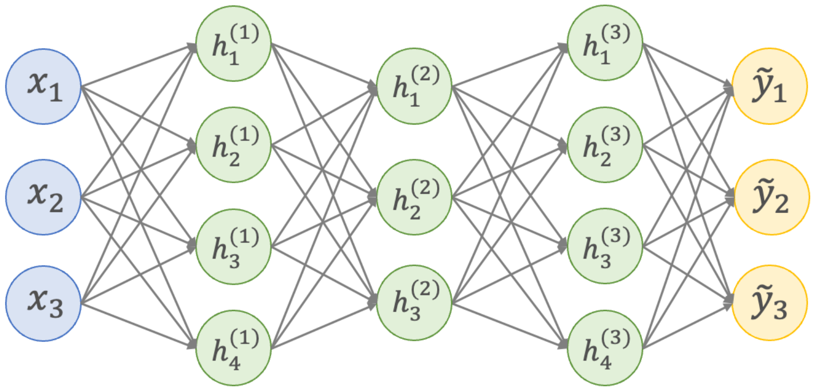
Activation Function (tanh, sigmoid)

Update

# Recurrent Neural Network (RNN)

$$h_t = \sigma(W h_{t-1} + U x_t + b)$$



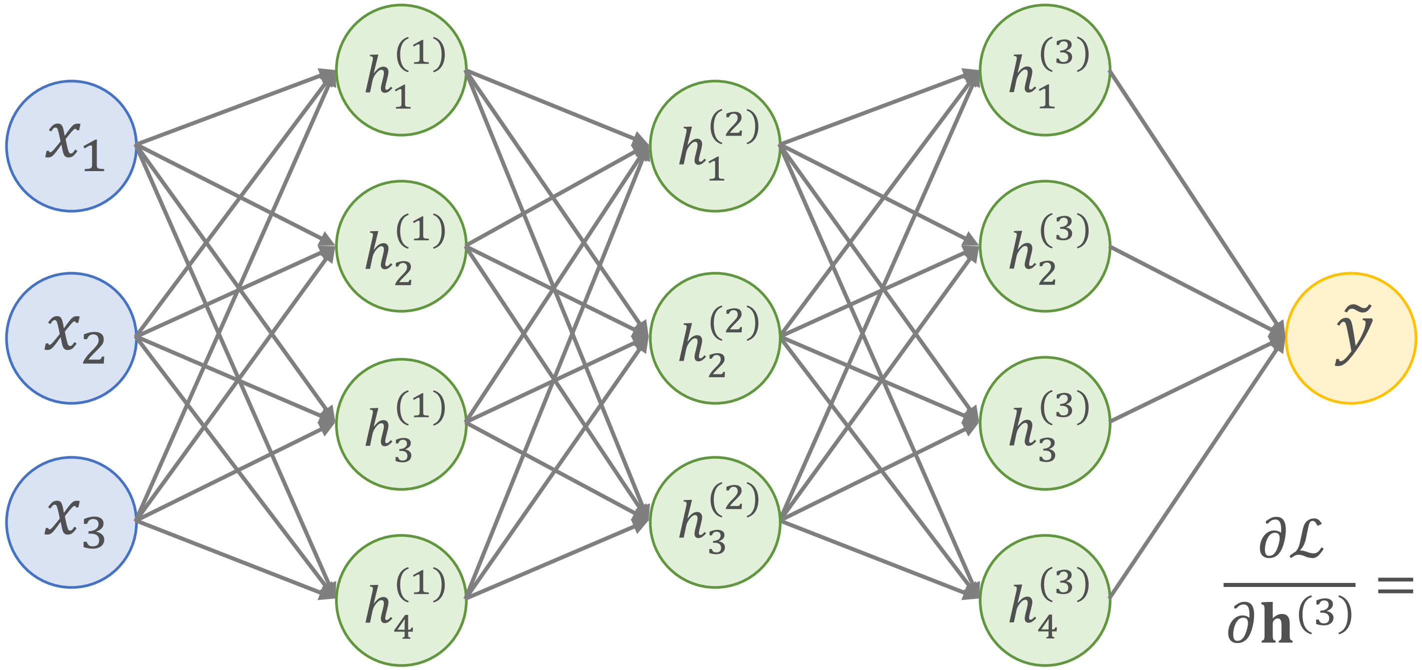




$$\mathcal{L}_{CE}(y, \tilde{y}) = -\sum_{c=0}^{C} y_c \log P(y = c |\, \mathbf{x})$$

# Recurrent Neural Network (RNN)

- Advantages
  - Can process any length input
  - Model size doesn’t increase for longer input context
  - Computation for step t can (in theory) use information from many steps back
- Disadvantages
  - Recurrent computation is slow
  - In practice, difficult to access information from many steps back
  - Vanishing gradient

# Back-Propagation




$$\frac{\partial \mathcal{L}}{\partial \tilde{y}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(3)}} = \frac{\partial \mathcal{L}}{\partial \tilde{y}} \cdot \frac{\partial \tilde{y}}{\partial \mathbf{h}^{(3)}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(2)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(3)}} \cdot \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(2)}} \cdot \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(1)}} \cdot \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}^{(1)}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(1)}} \cdot \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{b}^{(1)}}$$

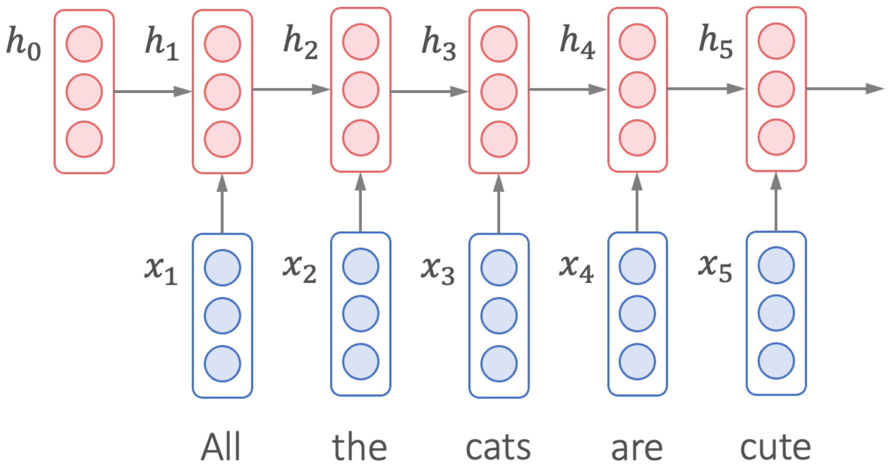
# Vanishing Gradient Problem

$$h_2 = \sigma(Wh_1 + Ux_2 + b)$$

$$h_3 = \sigma(Wh_2 + Ux_3 + b)$$

$$h_4 = \sigma(Wh_3 + Ux_4 + b)$$

$$h_5 = \sigma(Wh_4 + Ux_3 + b)$$




$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial h_5} \cdot \frac{\partial h_5}{\partial W} + \frac{\partial \mathcal{L}}{\partial h_5} \cdot \boxed{\frac{\partial h_5}{\partial h_4}} \cdot \frac{\partial h_4}{\partial W} + \frac{\partial \mathcal{L}}{\partial h_5} \cdot \boxed{\frac{\partial h_5}{\partial h_4} \cdot \frac{\partial h_4}{\partial h_3}} \cdot \frac{\partial h_3}{\partial W}$$

$$+ \frac{\partial \mathcal{L}}{\partial h_5} \cdot \boxed{\frac{\partial h_5}{\partial h_4} \cdot \frac{\partial h_4}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_2}} \cdot \frac{\partial h_2}{\partial W} + \frac{\partial \mathcal{L}}{\partial h_5} \cdot \boxed{\frac{\partial h_5}{\partial h_4} \cdot \frac{\partial h_4}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1}} \cdot \frac{\partial h_1}{\partial W}$$

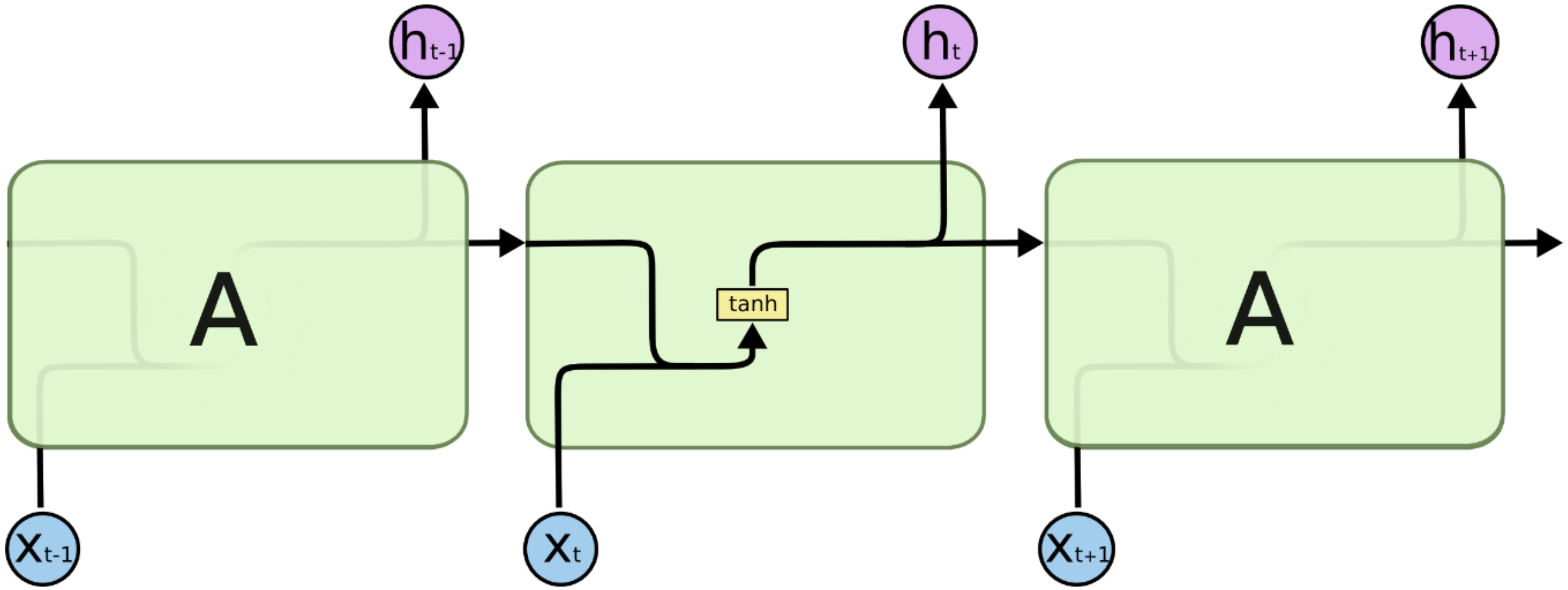
When these are small, the gradient signal gets smaller and smaller as it back-propagates further

Model weights are updated only with respect to short-term effect rather than long-term effect

# Long Short-Term Memory (LSTM)

- Short-term memory: hidden state $h_t$
- Long-term memory: cell state $c_t$
- Key idea
  - Turn multiplication into addition (partially reduce gradient vanishing)
  - use gates to control how much information to add/erase

# Recurrent Neural Network (RNN)

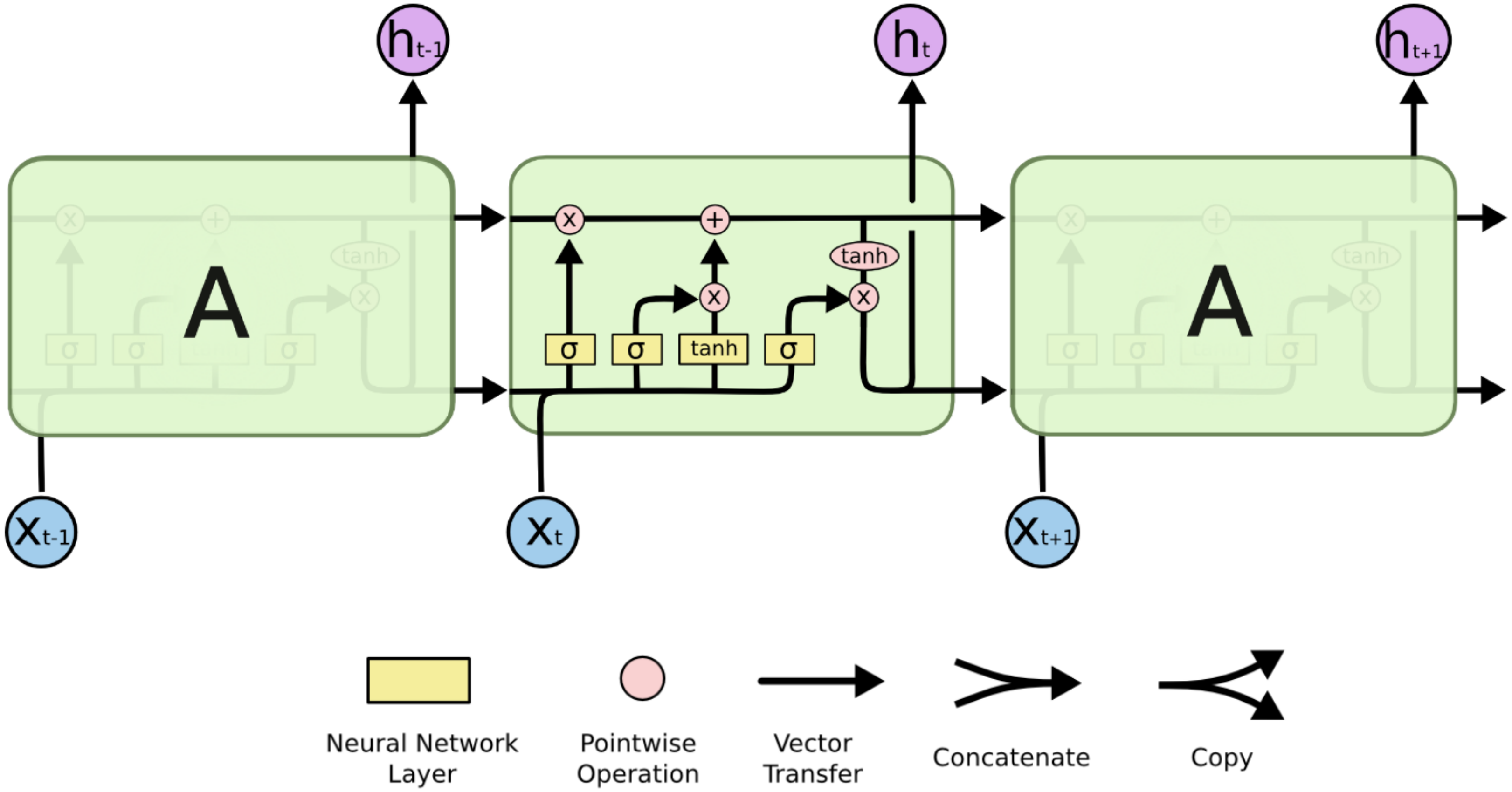



$$h_t = \sigma(Wh_{t-1} + Ux_t + b)$$



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

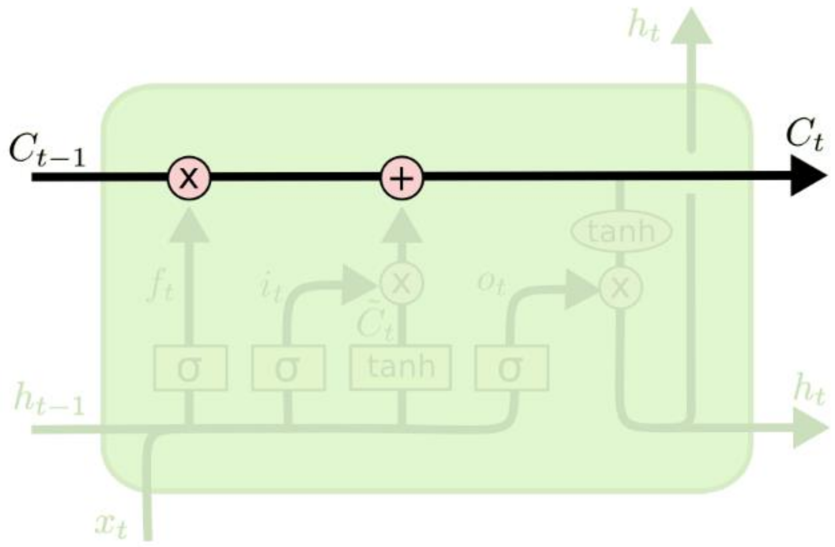
# Long Short-Term Memory (LSTM)






https://colah.github.io/posts/2015-08-Understanding-LSTMs/

# Long Short-Term Memory (LSTM)

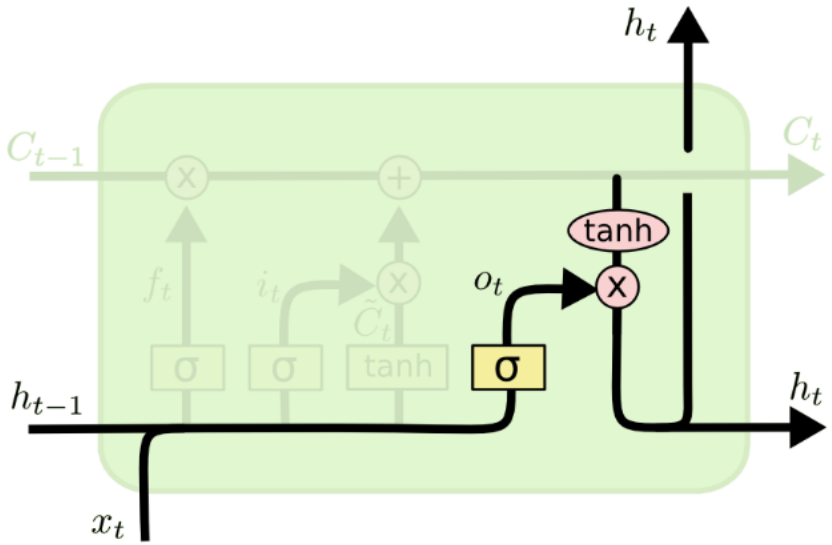



The cell state stores long-term information



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

# Long Short-Term Memory (LSTM)

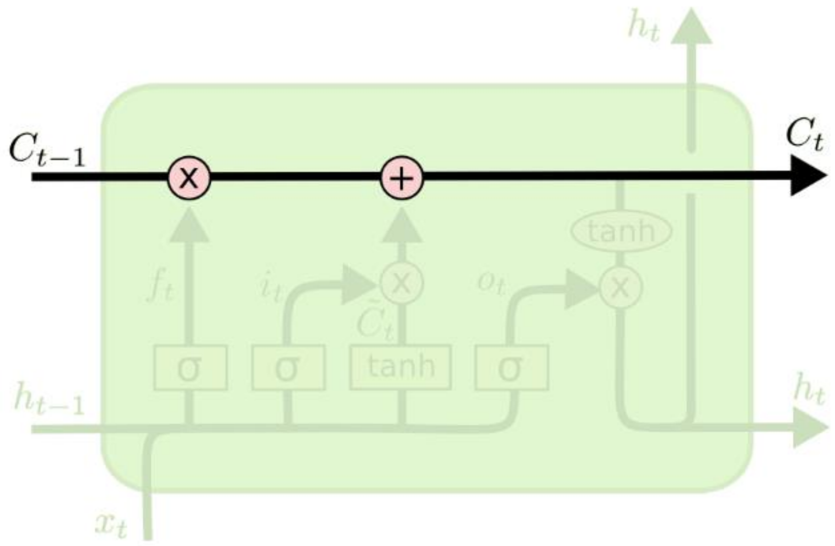



The hidden state stores short-term information



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

# Long Short-Term Memory (LSTM)



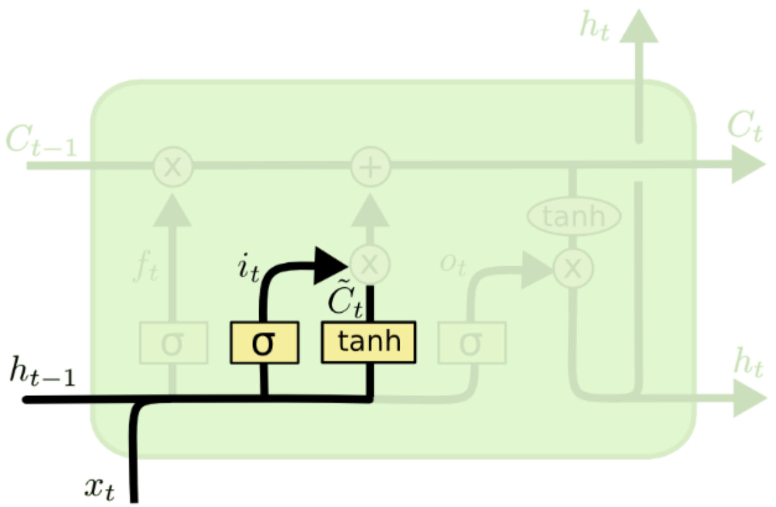

The cell state stores long-term information

Whenever reading a word, we will write/forget information to the cell state



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

# Long Short-Term Memory (LSTM)



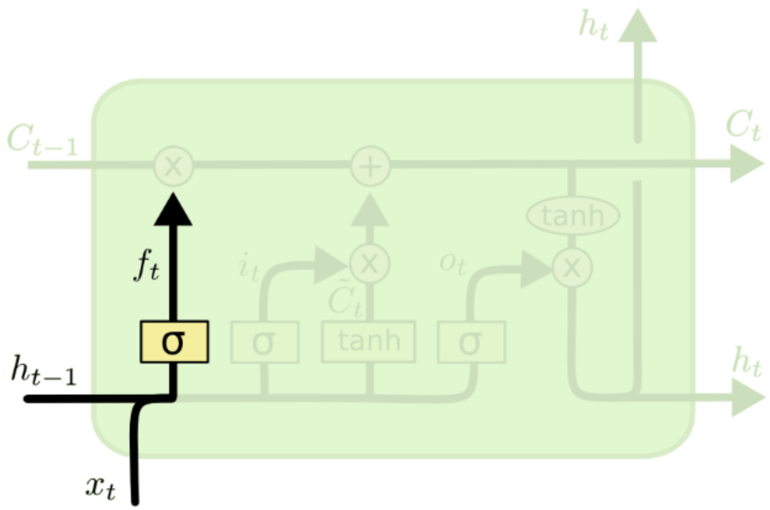

Sigmoid function: gate values are between 0 and 1

Input gate $\quad i_t = \sigma\left(W^{(i)} h_{t-1} + U^{(i)} x_t + b^{(i)}\right) \quad$ How much we should write

New information $\quad \tilde{C}_t = \tanh\left(W^{(C)} h_{t-1} + U^{(C)} x_t + b^{(C)}\right) \quad$ What we should write

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

# Long Short-Term Memory (LSTM)



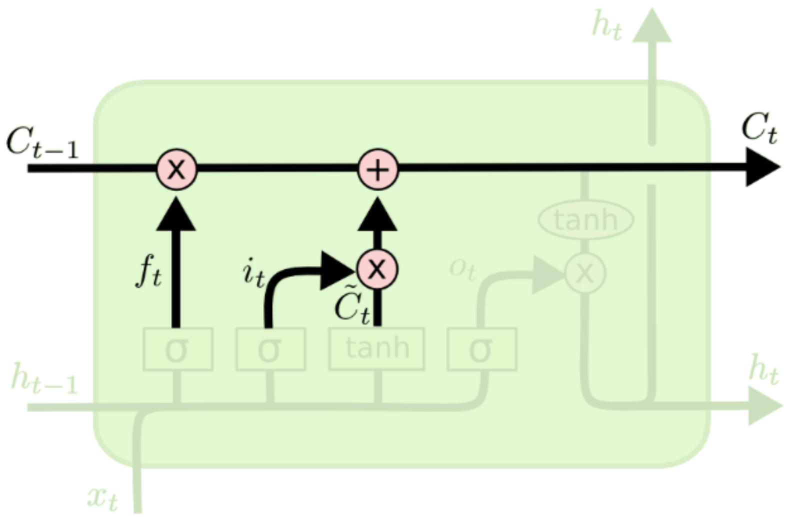

Sigmoid function: gate values are between 0 and 1

Forget gate $f_t = \sigma\left(W^{(f)}h_{t-1} + U^{(f)}x_t + b^{(f)}\right)$ How much we should erase



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

# Long Short-Term Memory (LSTM)




What we should write

Update cell state $$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

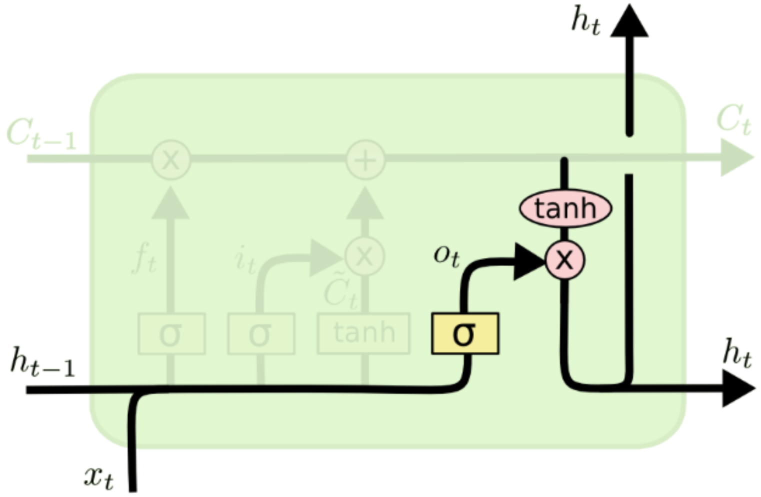
How much we should erase

How much we should write



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

# Long Short-Term Memory (LSTM)




Update hidden state

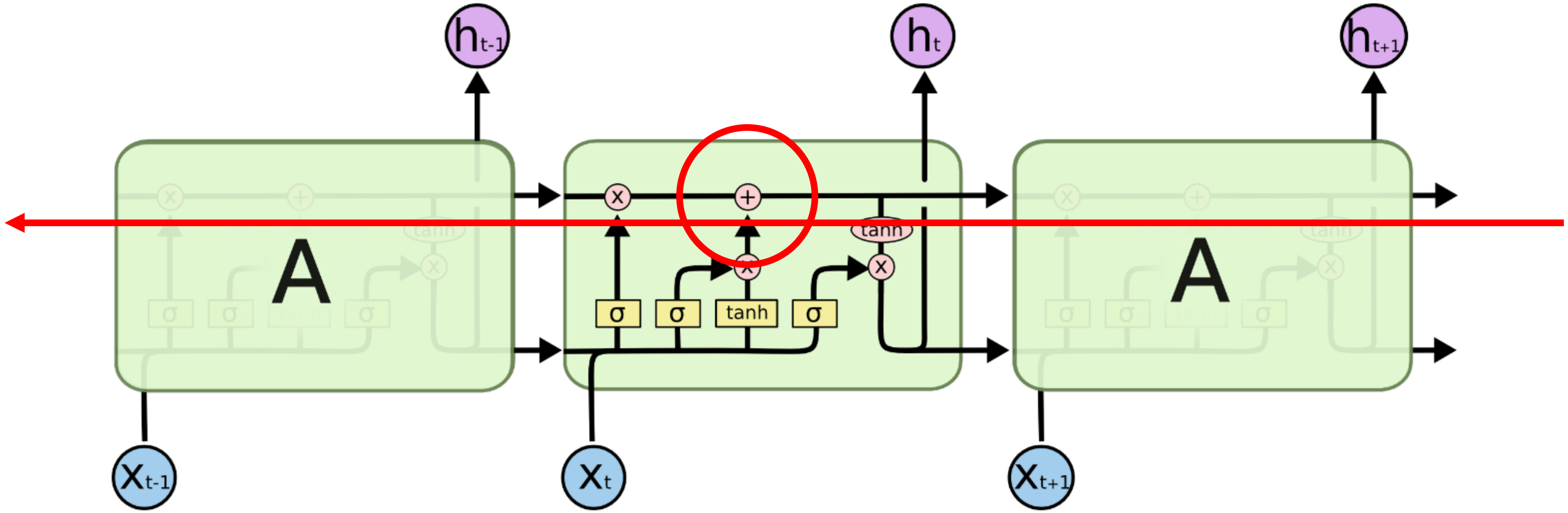
$$o_t = \sigma\left(W^{(o)} h_{t-1} + U^{(o)} x_t + b^{(o)}\right)$$

$$h_t = o_t * \tanh(C_t)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

# Long Short-Term Memory (LSTM)

Uninterrupted gradient flow




The addition is the key

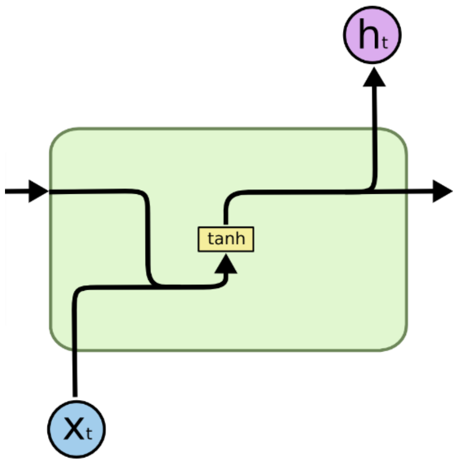
LSTM does not guarantee that there is no vanishing gradient
but it does provide an easier way to learn long-distance dependencies



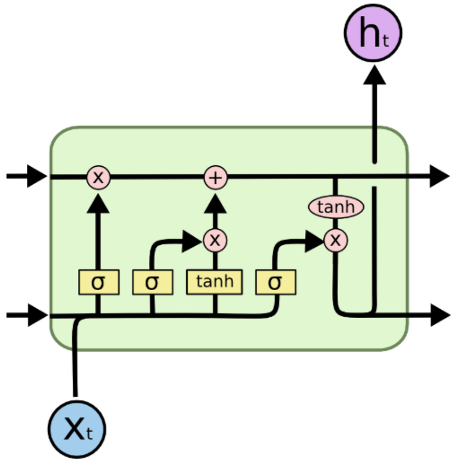
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

# RNN vs. LSTM

RNN




LSTM




$$h_t = \sigma(W h_{t-1} + U x_t + b)$$

$$i_t = \sigma\left(W^{(i)} h_{t-1} + U^{(i)} x_t + b^{(i)}\right)$$

$$f_t = \sigma\left(W^{(f)} h_{t-1} + U^{(f)} x_t + b^{(f)}\right)$$

$$o_t = \sigma\left(W^{(o)} h_{t-1} + U^{(o)} x_t + b^{(o)}\right)$$

$$\tilde{C}_t = \tanh\left(W^{(C)} h_{t-1} + U^{(C)} x_t + b^{(C)}\right)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

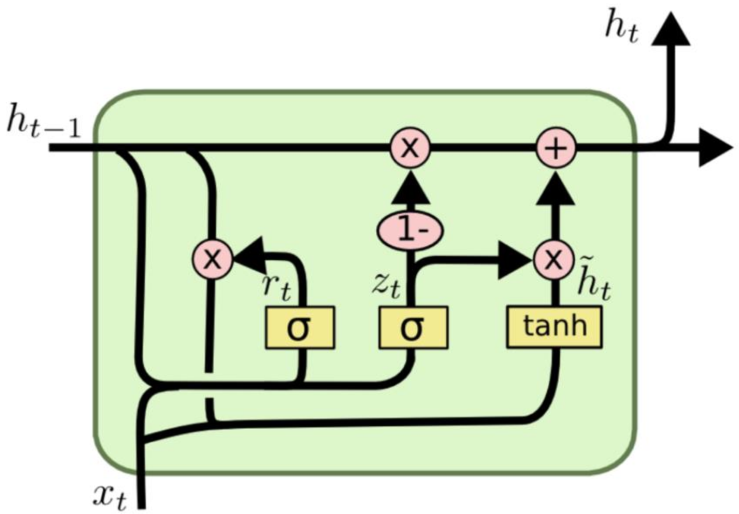
$$h_t = o_t * \tanh(C_t)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

# Gated Recurrent Units (GRU)

- Simplify 3 gates to 2 gates
  - Reset gate and update gate
- No explicit cell state
- More training-efficient

# Gated Recurrent Units (GRU)




Reset gate $r_t = \sigma\left(W^{(r)} h_{t-1} + U^{(r)} x_t + b^{(r)}\right)$

Update gate $z_t = \tanh\left(W^{(z)} h_{t-1} + U^{(z)} x_t + b^{(z)}\right)$

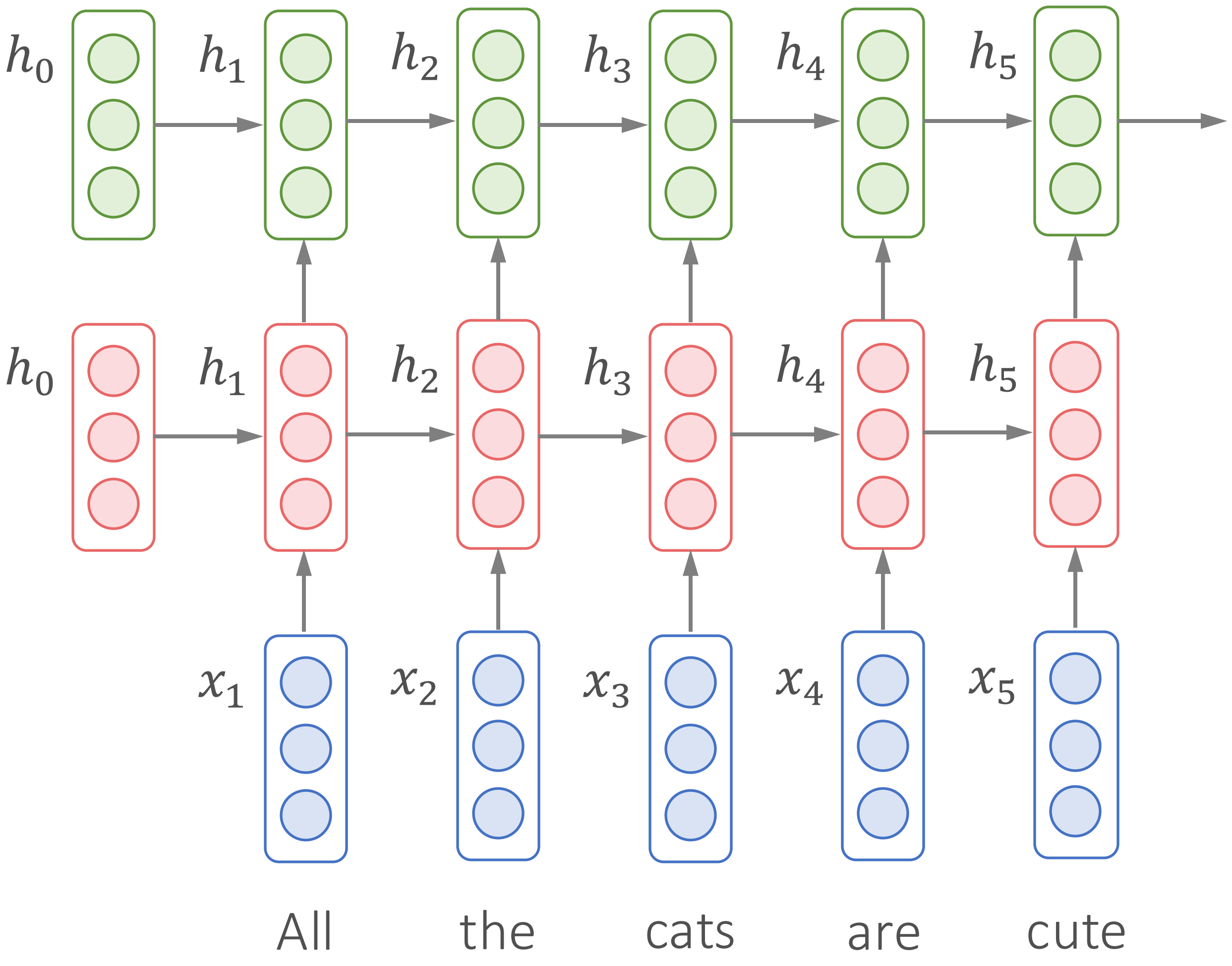
New hidden state

$$\tilde{h}_t = \tanh(W(r_t * h_{t-1}) + U x_t + b)$$

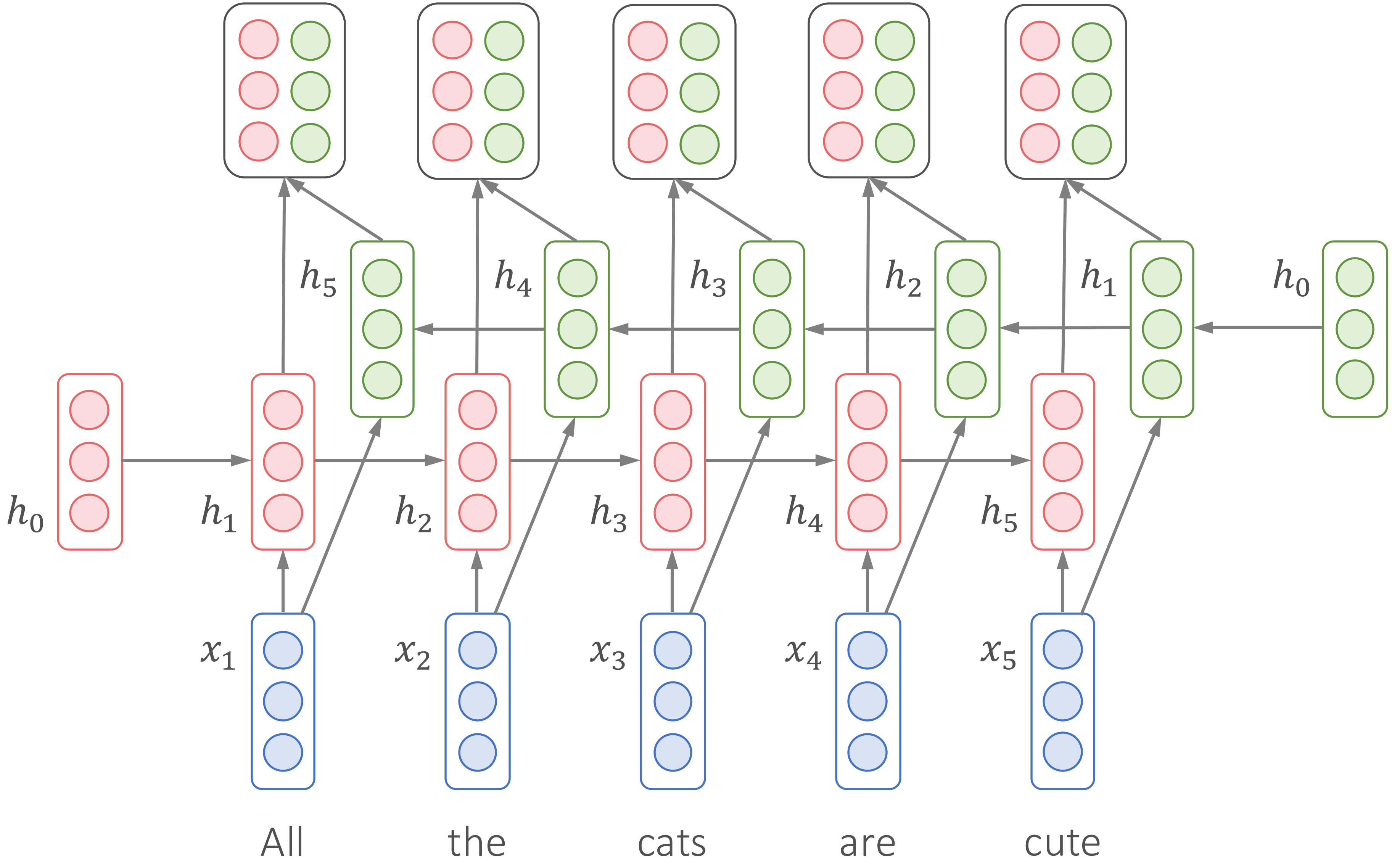
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Merge input and forget gate

# Multi-Layer RNN (Stacked RNN)



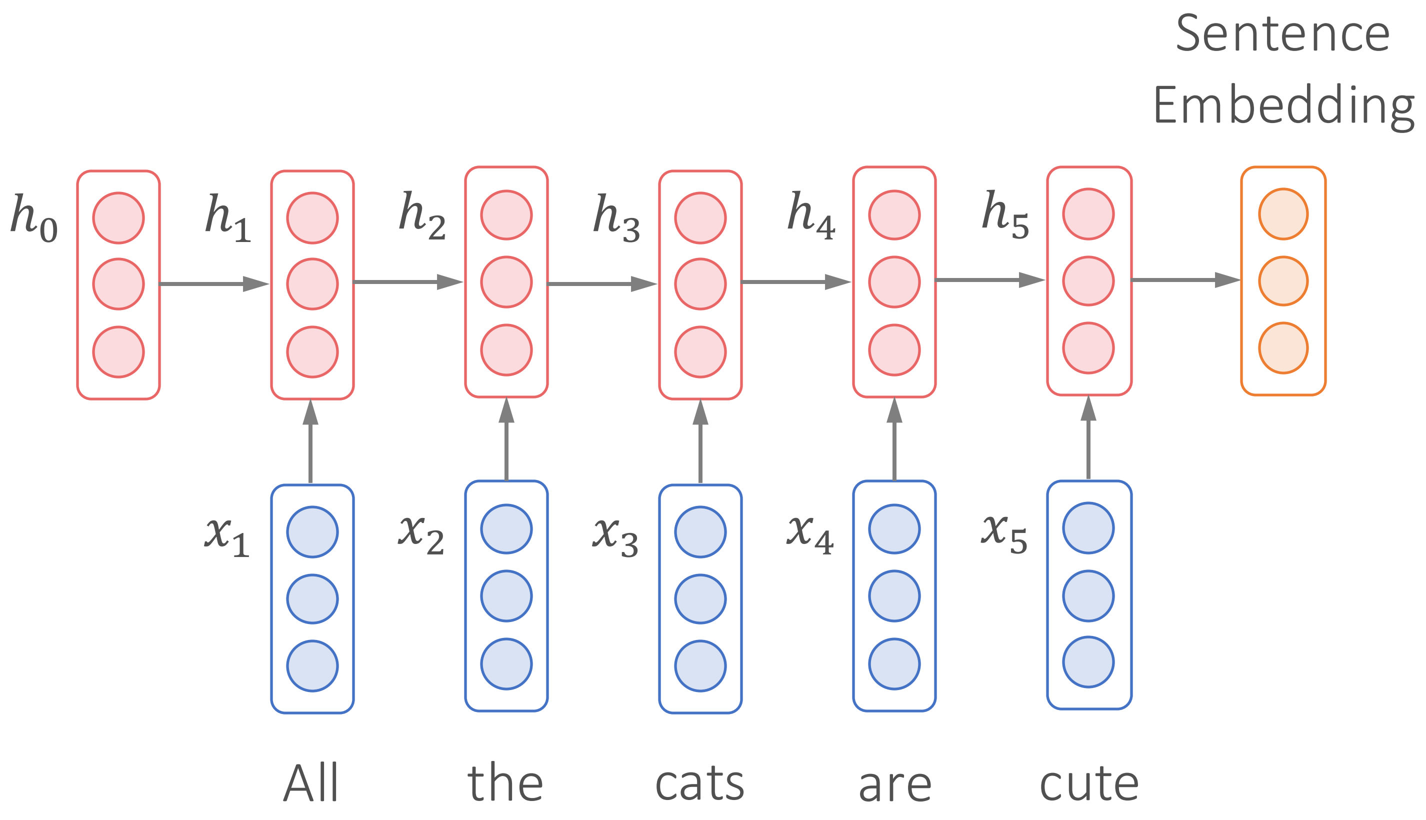
# Bidirectional RNN



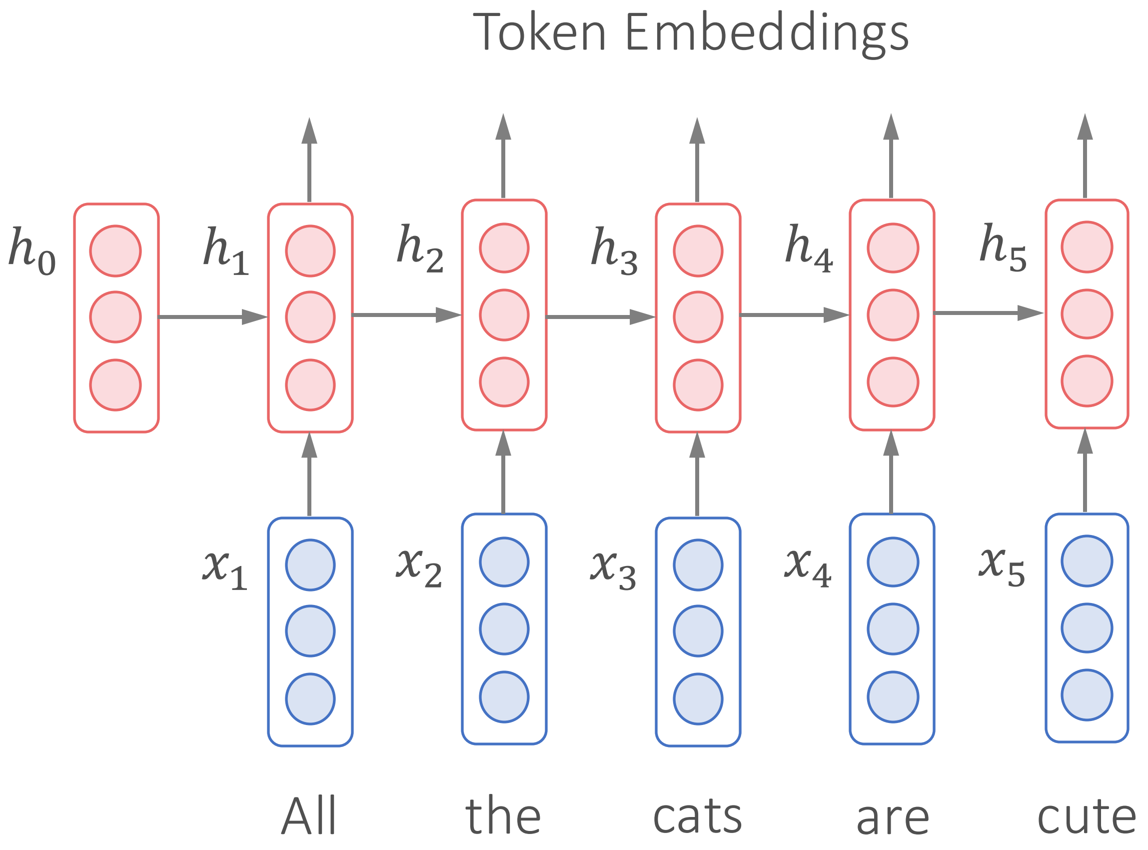
# RNN is Flexible

- Can be used for both classification and generation
  - Encoder
  - Decoder
  - Encoder-decoder

# RNN as Sentence-Level Encoder



# RNN as Token-Level Encoder




The embeddings are contextualized

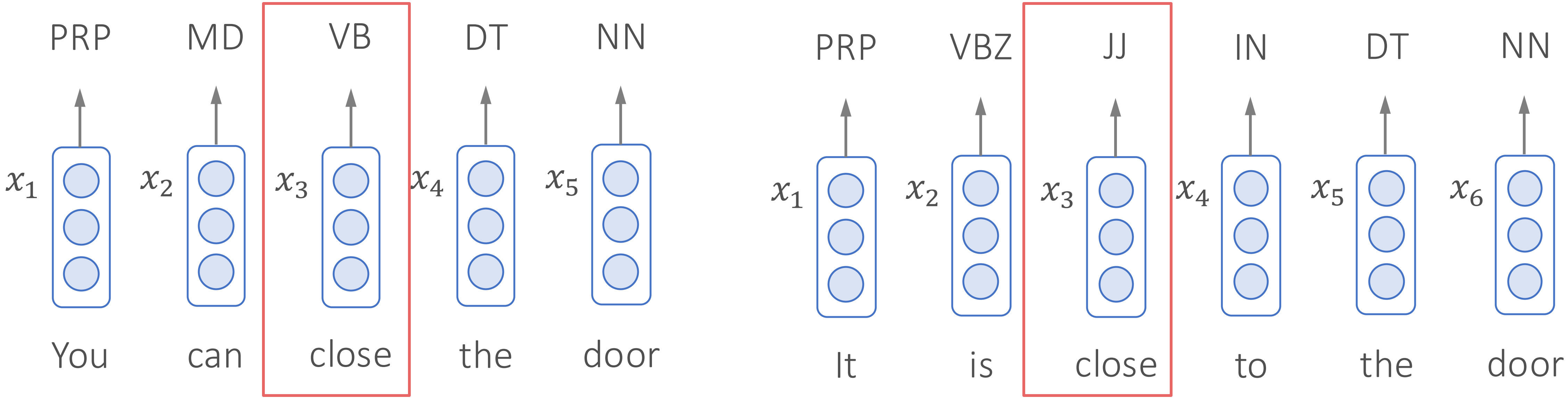
# Part-of-Speech (POS) Tagging

| | | | | |
|---|---|---|---|---|
| *You* | *can* | *close* | *the* | *door* |
| PRP | MD | VB | DT | NN |

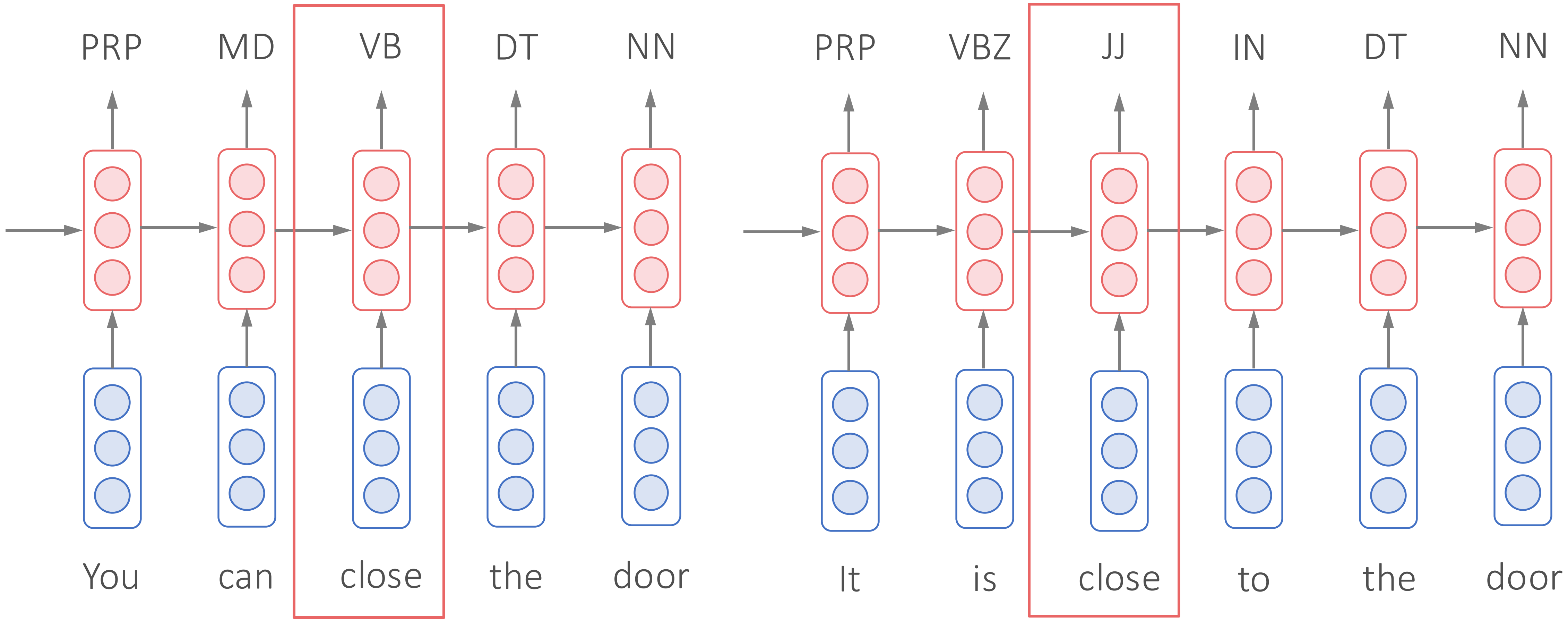
| | | | | | |
|---|---|---|---|---|---|
| *It* | *is* | *close* | *to* | *the* | *door* |
| PRP | VBZ | JJ | IN | DT | NN |

It’s a structed prediction problem

# POS Tagging with Word Embeddings



# POS Tagging with Sequential Labeling



# Named Entity Recognition

*John* went to *New York City* to visit *Kuan-Hao Huang*

Entity Entity Entity

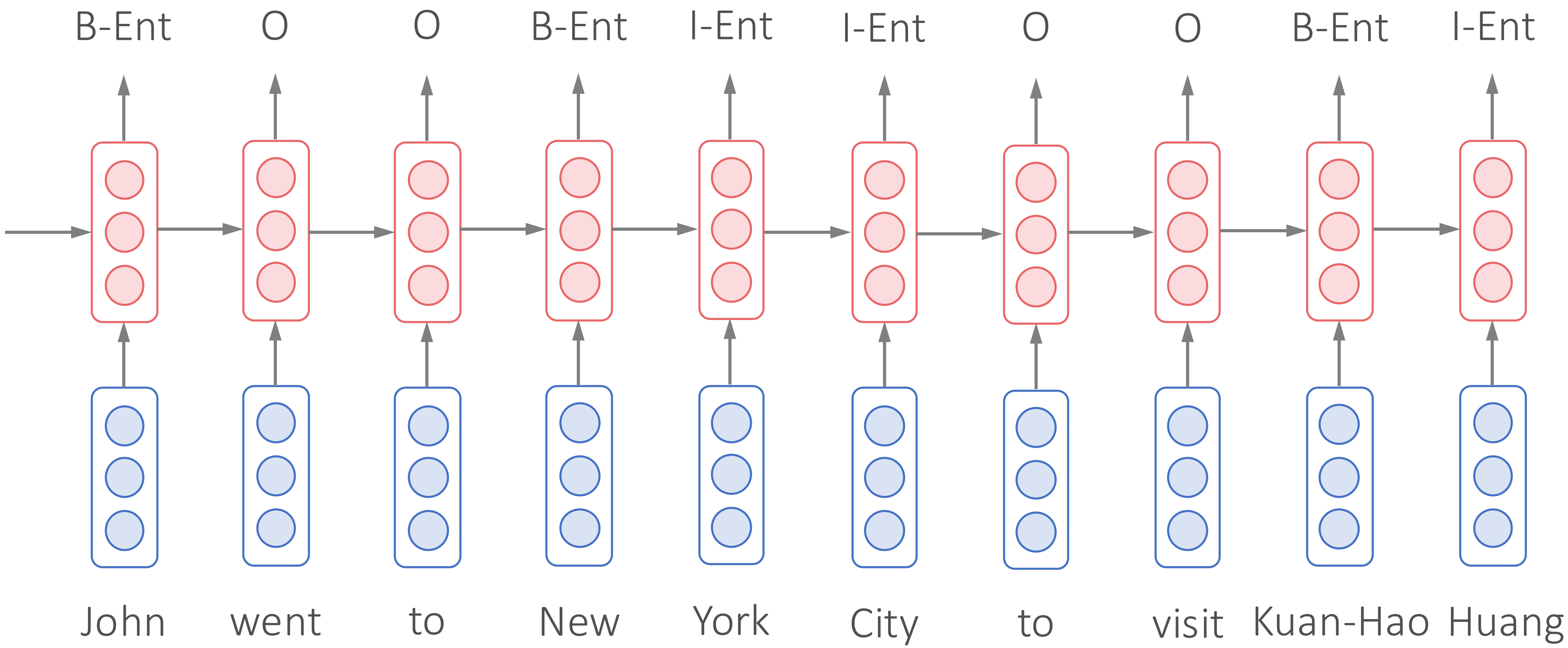
**BIO Sequence**

| *John* | *went* | *to* | *New* | *York* | *City* | *to* | *visit* | *Kuan-Hao* | *Huang* |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B-Entity | Other | Other | B-Entity | I-Entity | I-Entity | Other | Other | B-Entity | I-Entity |

B-Entity: Begin of an entity span, I-Entity: Inside of an entity span

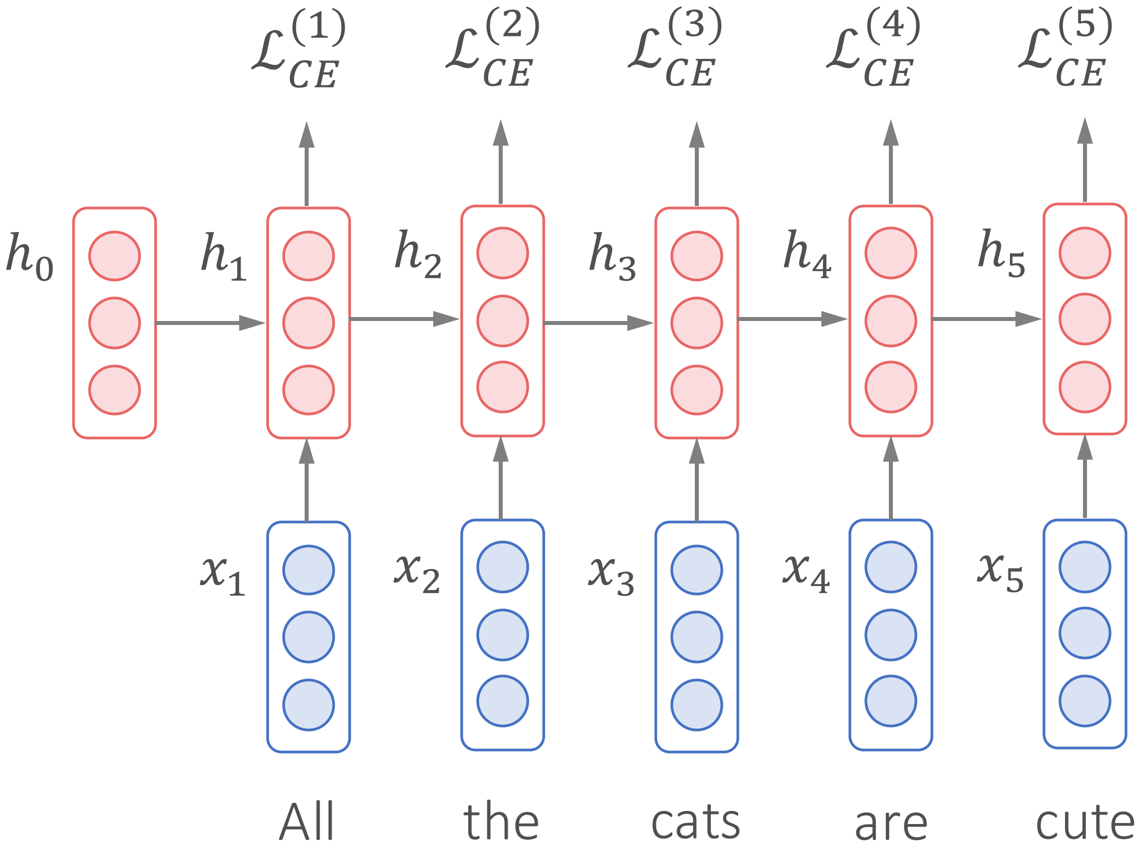
It's a structed prediction problem

# Named Entity Recognition as Sequential Labeling



# Sequential Labeling

- A sequence of dependent classification




$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_{CE}^{(i)}$$

# RNN as Decoder (Generator)

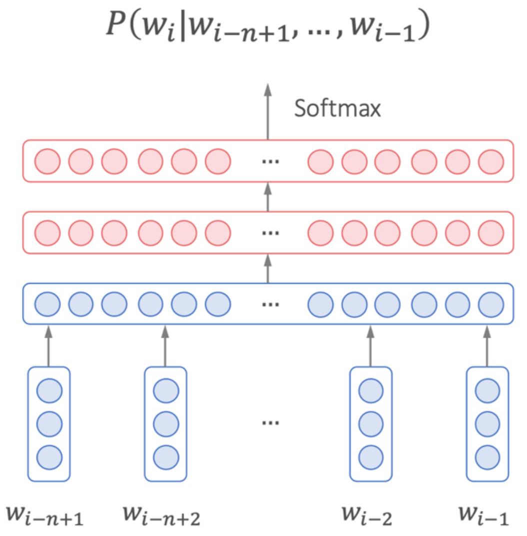
- RNN Language Modeling
  - Generation is a sequence of word classification

$$P(w_1, w_2, w_3, \dots, w_l) = P(w_1)P(w_2, w_3, \dots, w_l | w_1)$$

$$= P(w_1)P(w_2|w_1)(w_3, \dots, w_l | w_1, w_2)$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)(w_4, \dots, w_l | w_1, w_2, w_3)$$

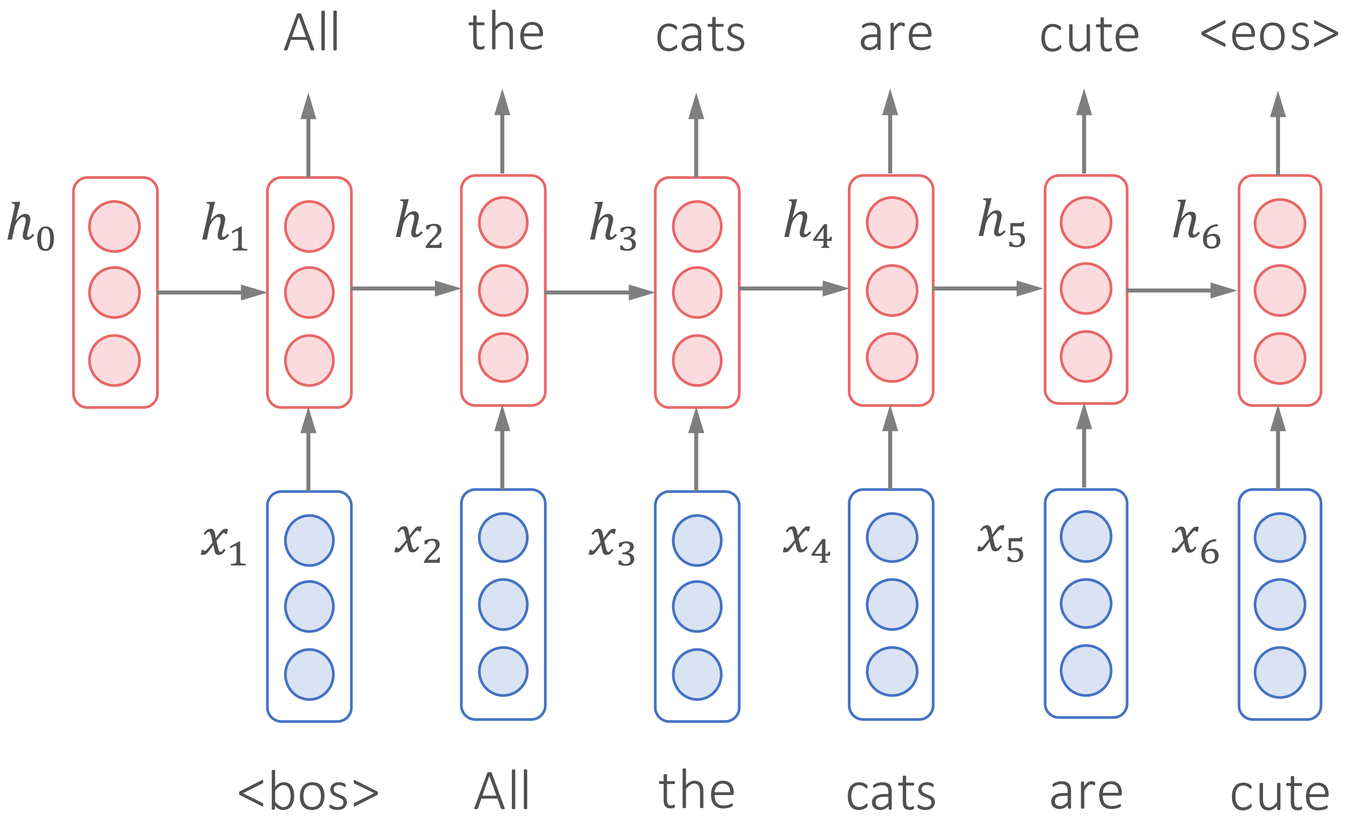
$$= \prod_{i=1}^{l} P(w_i | w_1, w_2, \dots, w_{i-1})$$



Neural language models with context window

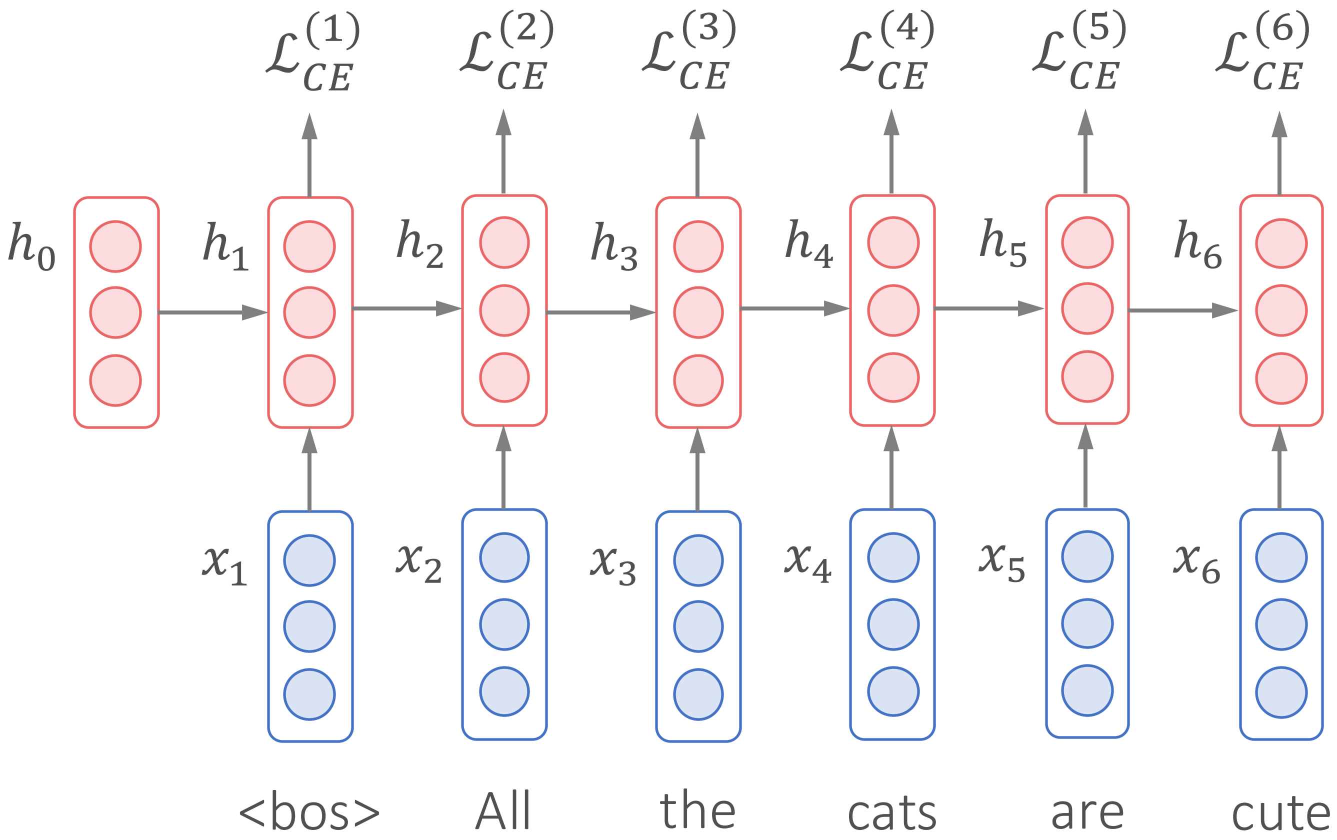
# RNN as Decoder (Generator)

- RNN Language Modeling
  - Generation is a sequence of word classification



# RNN as Decoder (Generator)

- RNN Language Modeling
  - Generation is a sequence of word classification

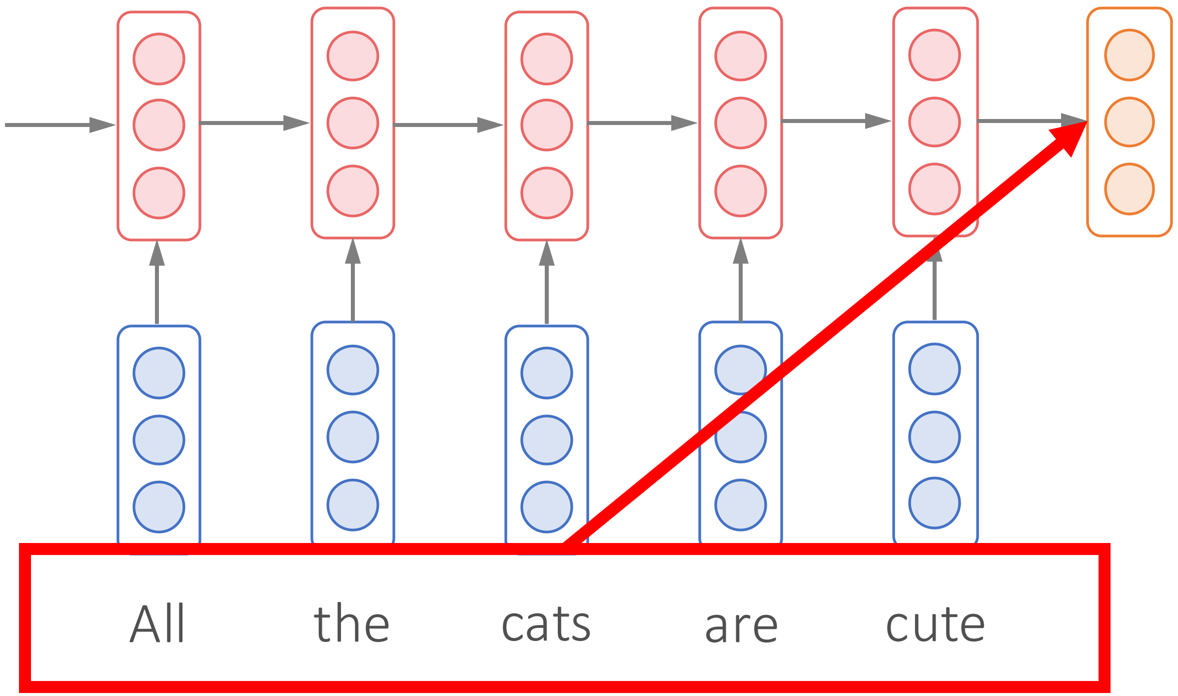



$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_{CE}^{(i)}$$

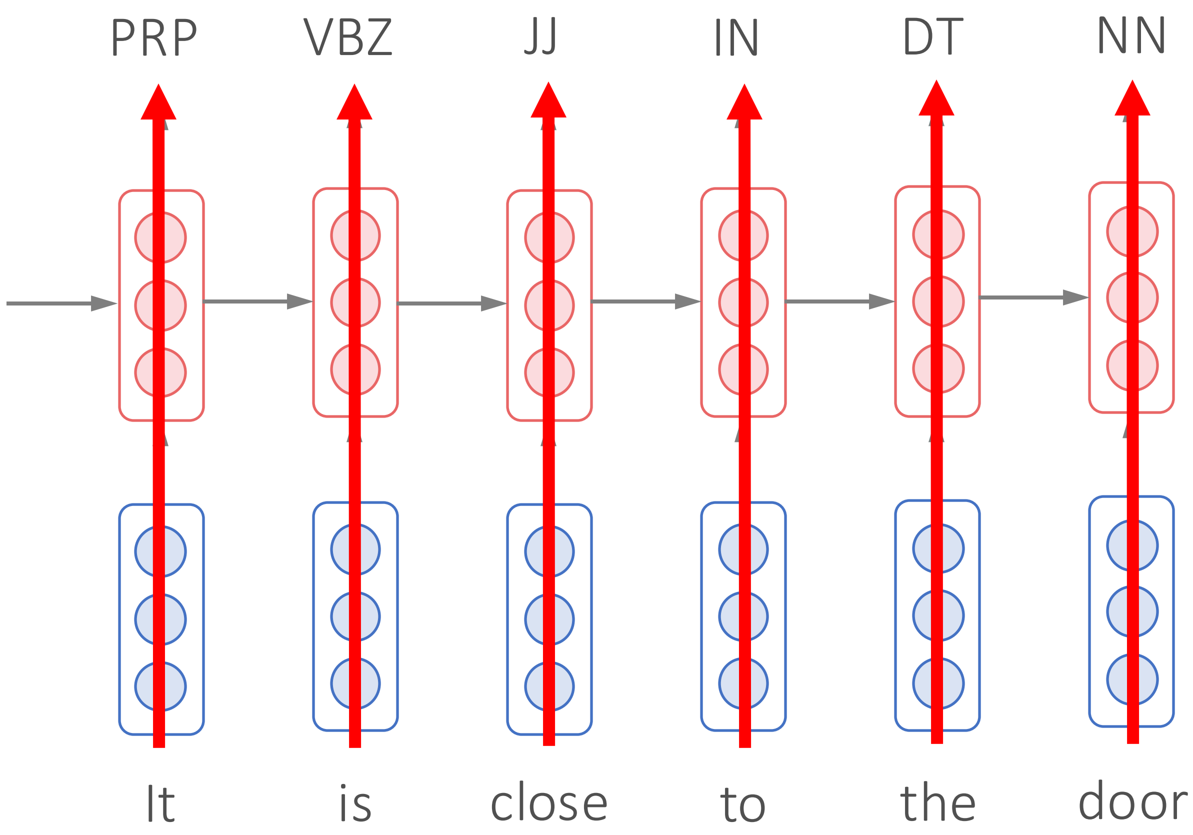
# Encoder vs. Decoder

- Encoder
  - Focus more on representations and understanding
- Decoder
  - Focus on generation

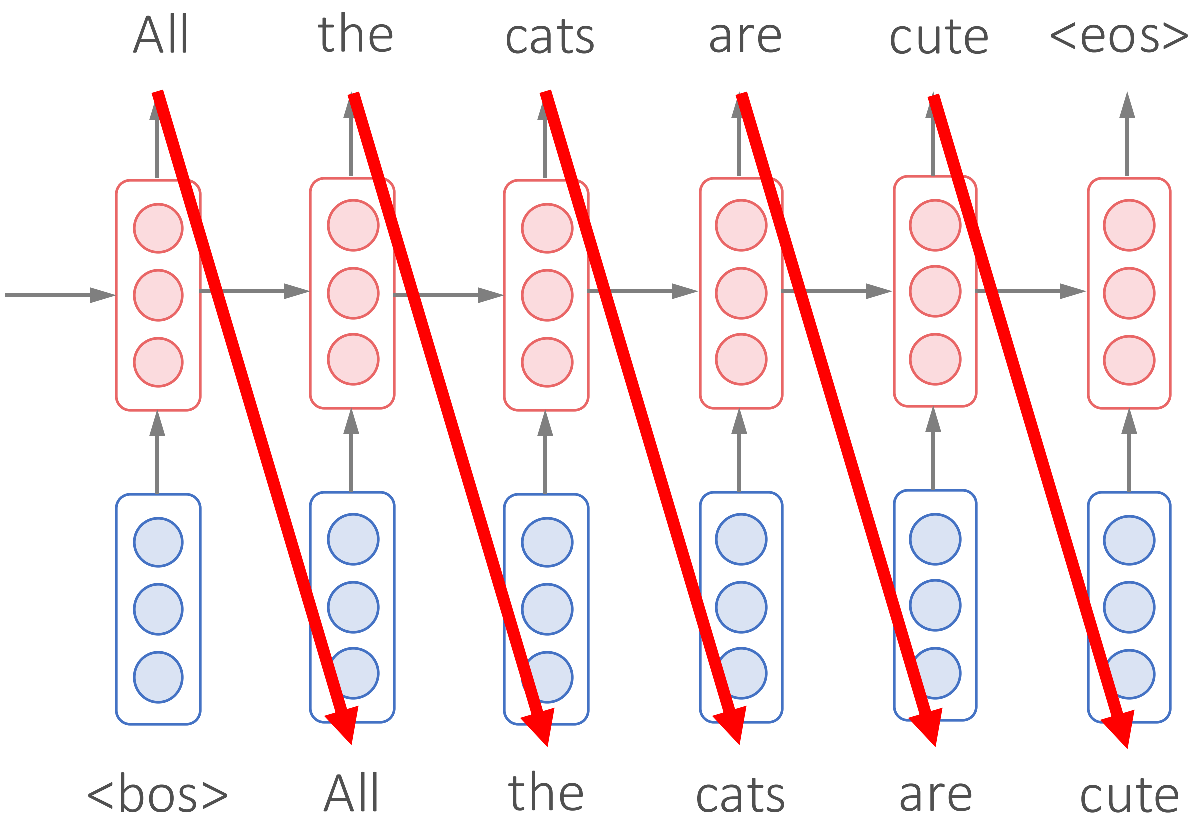
# Encoder



# Encoder

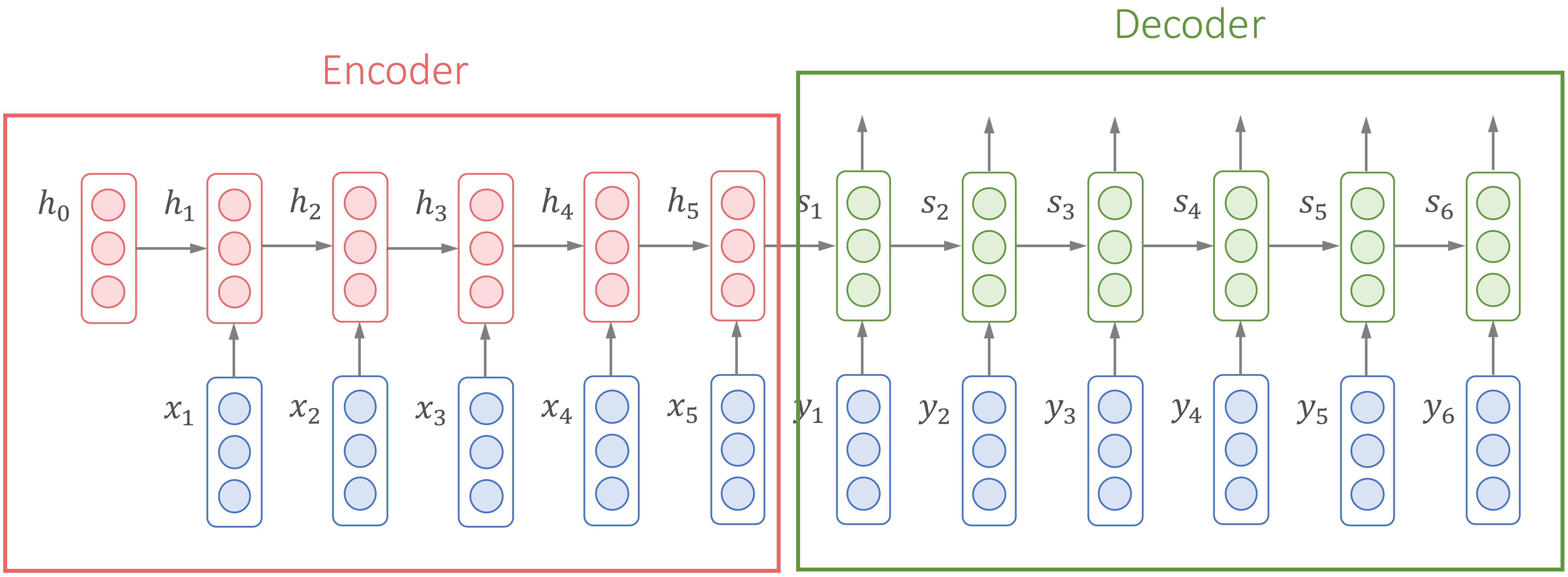


# Decoder

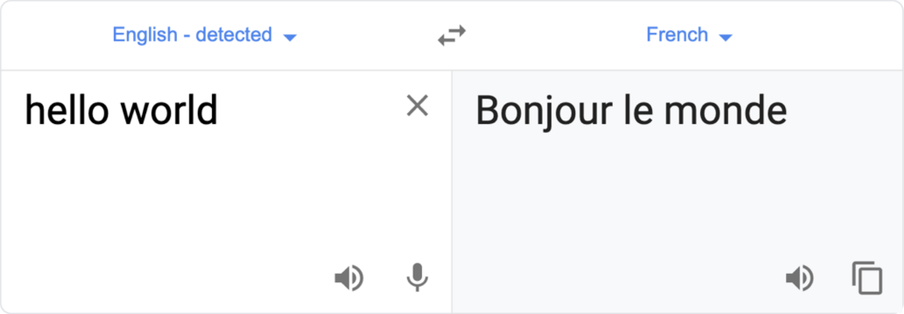


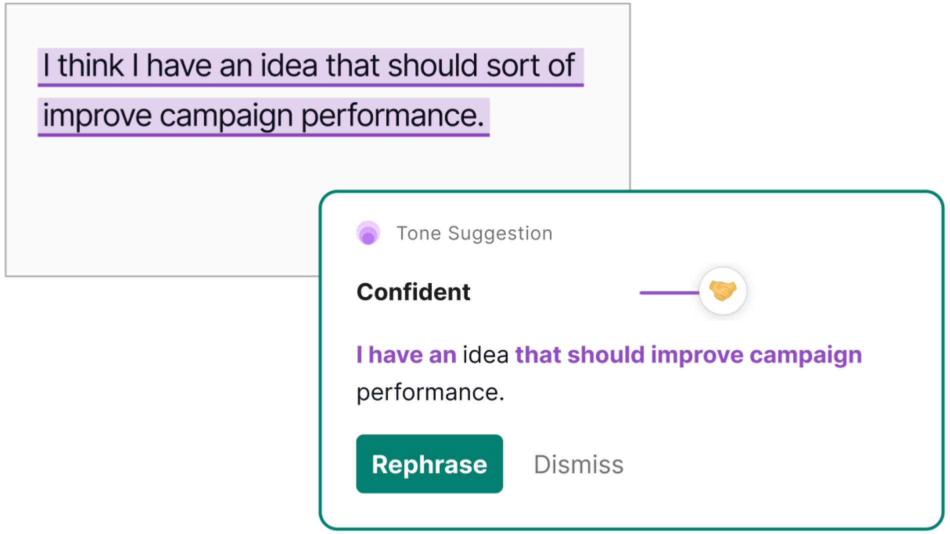
# Sequence-to-Sequence Models (Seq2Seq)

- When we need understanding and generation at the same time



# Sequence-to-Sequence Tasks




Provided proper attribution is provided, Google hereby grants permission to reproduce the tables and figures in this paper solely for use in journalistic or scholarly works.

**Attention Is All You Need**

| | | | |
|---|---|---|---|
| **Ashish Vaswani*** Google Brain avaswani@google.com | **Noam Shazeer*** Google Brain noam@google.com | **Niki Parmar*** Google Research nikip@google.com | **Jakob Uszkoreit*** Google Research usz@google.com |
| **Llion Jones*** Google Research llion@google.com | **Aidan N. Gomez*** † University of Toronto aidan@cs.toronto.edu | **Łukasz Kaiser*** Google Brain lukaszkaiser@google.com | |

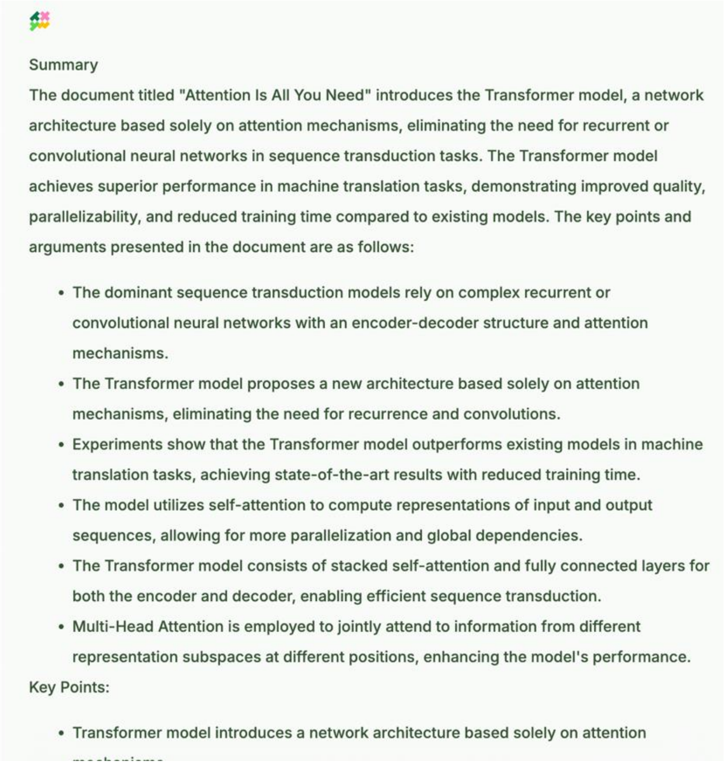
**Illia Polosukhin*** ‡
illia.polosukhin@gmail.com

**Abstract**

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain



Summary

The document titled "Attention Is All You Need" introduces the Transformer model, a network architecture based solely on attention mechanisms, eliminating the need for recurrent or convolutional neural networks in sequence transduction tasks. The Transformer model achieves superior performance in machine translation tasks, demonstrating improved quality, parallelizability, and reduced training time compared to existing models. The key points and arguments presented in the document are as follows:

- The dominant sequence transduction models rely on complex recurrent or convolutional neural networks with an encoder-decoder structure and attention mechanisms.
- The Transformer model proposes a new architecture based solely on attention mechanisms, eliminating the need for recurrence and convolutions.
- Experiments show that the Transformer model outperforms existing models in machine translation tasks, achieving state-of-the-art results with reduced training time.
- The model utilizes self-attention to compute representations of input and output sequences, allowing for more parallelization and global dependencies.
- The Transformer model consists of stacked self-attention and fully connected layers for both the encoder and decoder, enabling efficient sequence transduction.
- Multi-Head Attention is employed to jointly attend to information from different representation subspaces at different positions, enhancing the model's performance.

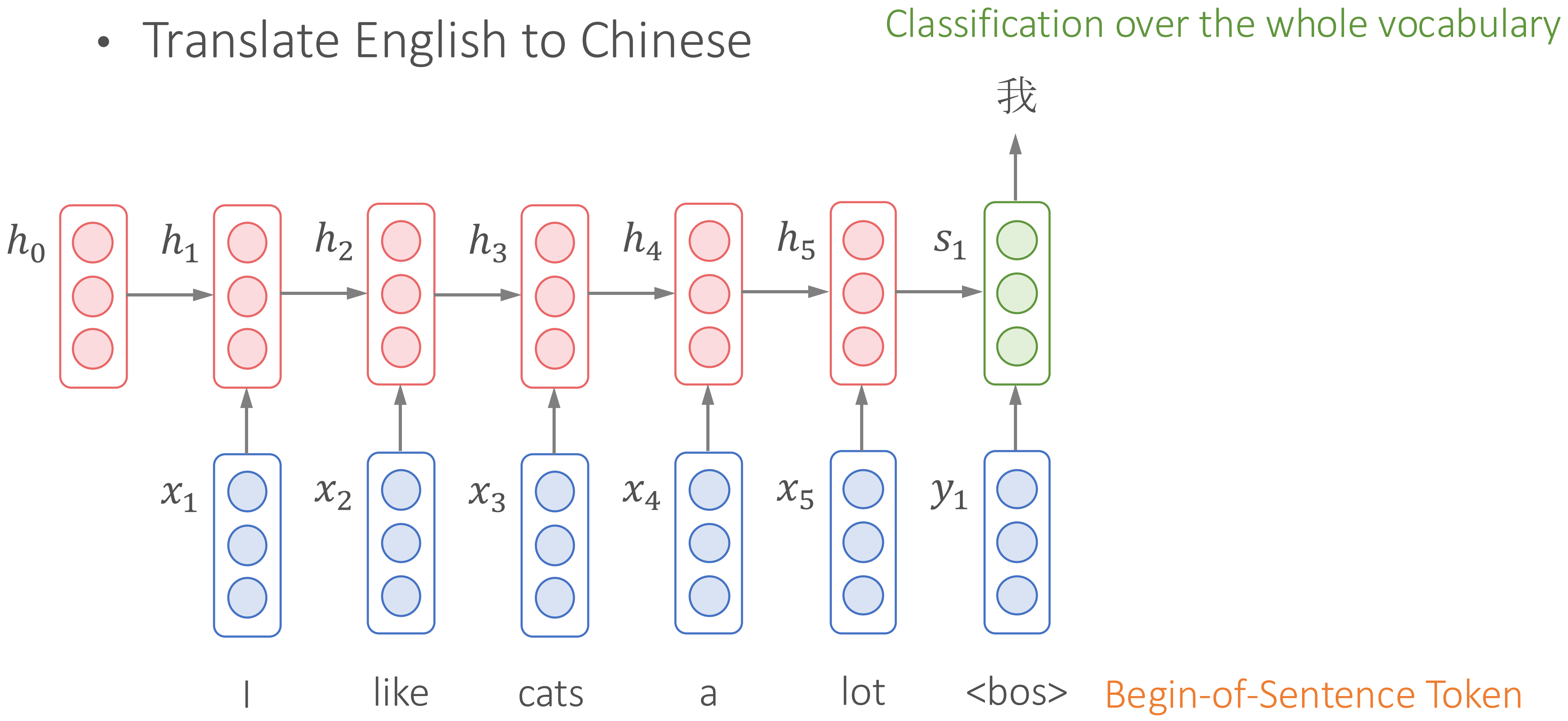
Key Points:

- Transformer model introduces a network architecture based solely on attention

https://www.txyz.ai/
https://www.grammarly.com/grammar-check
https://translate.google.com/

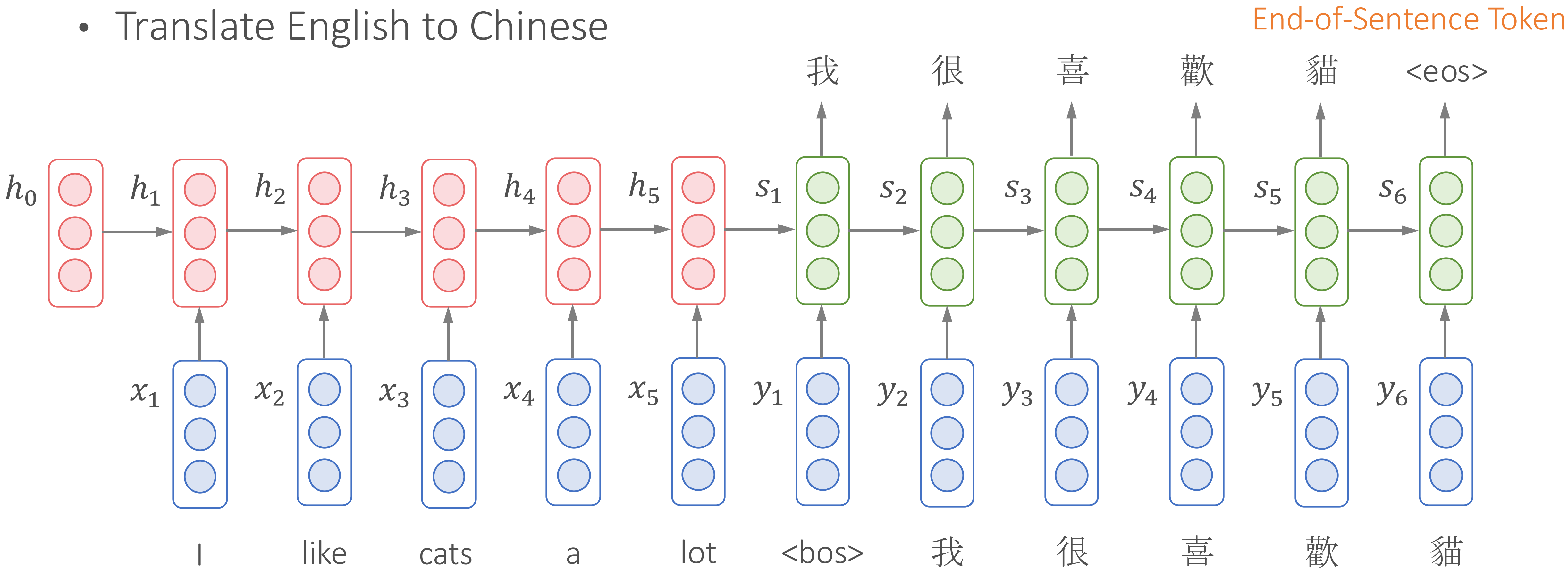
# Translation

- Translate English to Chinese



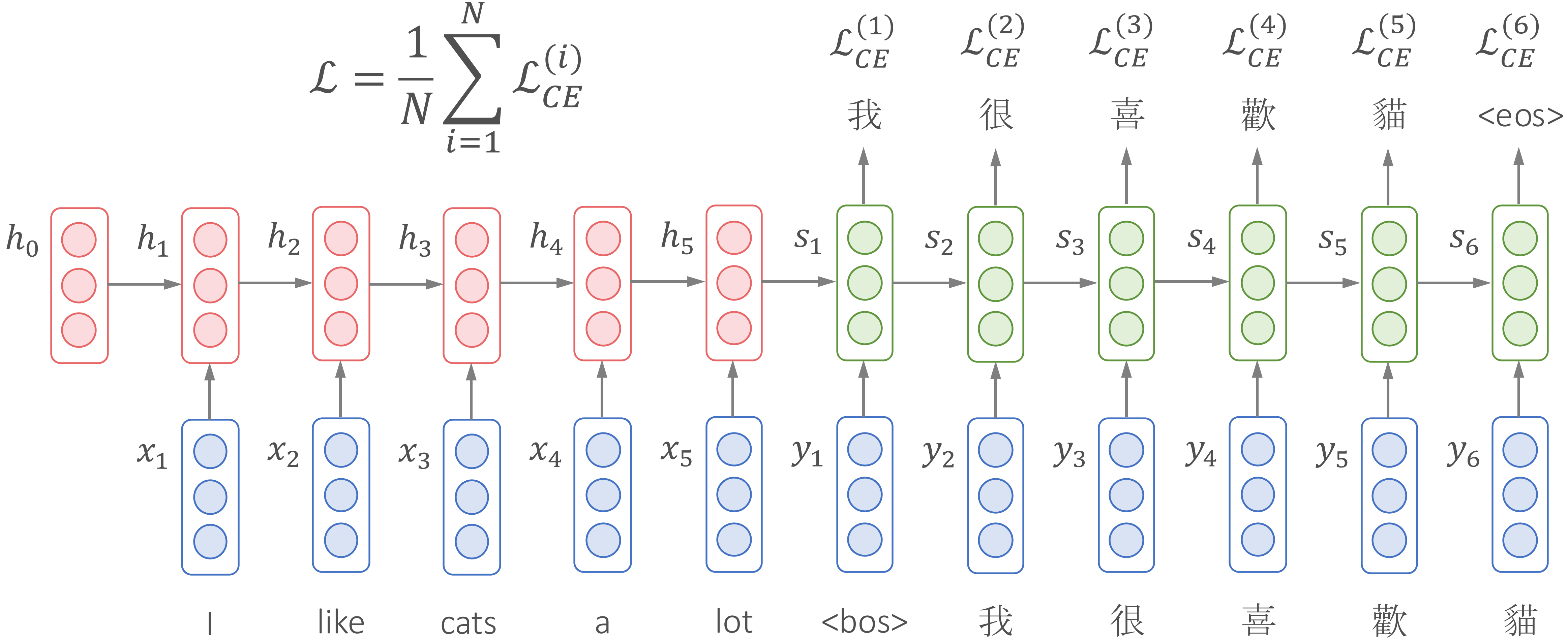
# Translation

- Translate English to Chinese



# Sequence-to-Sequence Model Loss

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_{CE}^{(i)}$$

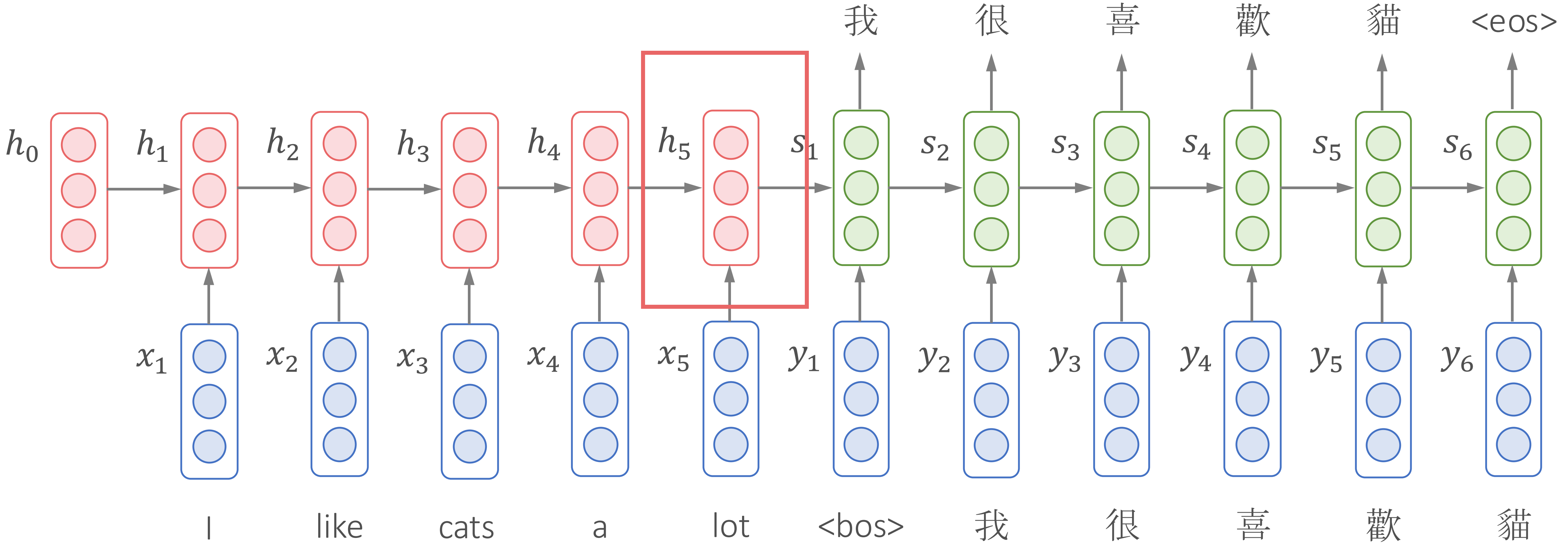


# Decoder-Only Models vs. Seq2Seq Models

- Decoder-only models with prompting
  - Continue writing
- Seq2Seq models
  - Encode first, then generate
- The difference becomes larger when we talk about Transformers!

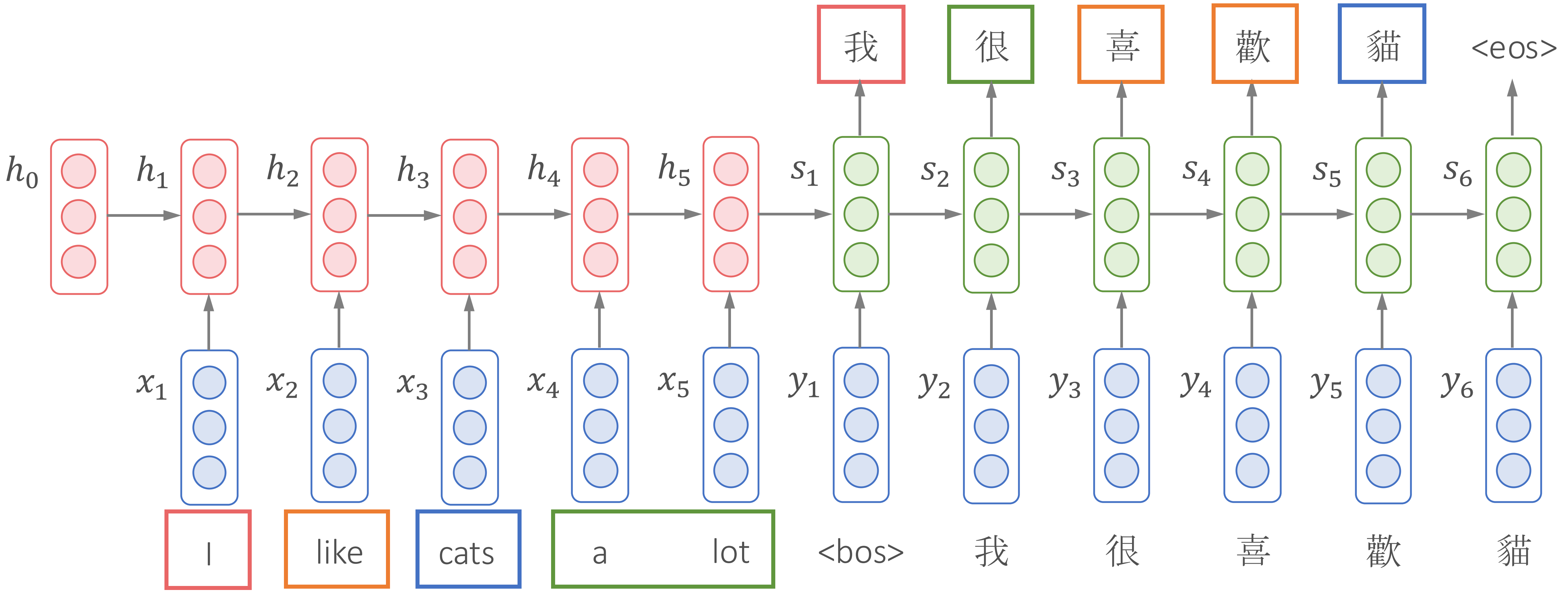
# Seq2Seq: Bottleneck

- A single vector needs to capture all the information about source sentence
- Longer sequences can still lead to vanishing gradients



# Focus on A Particular Part When Decoding

- Each token classification requires different part of information from source sentence



# Next: Attention

- Attention provides a solution to the bottleneck problem
- Key idea: At each time step during decoding, focus on a particular part of source sentence